ERTS 2022 proceedings

Philippe Cuenot, Marie de Roquemaurel, Kevin Delmas, Jean-Marc Gabriel, Adrien Gauffriau, Christophe Grand, Éric Jenn, Mohamed Kaâniche, Benoît Morgan

11th EDITION EUROPEAN CONGRESS

EMBEDDED REAL TIME SYSTEMS

ERTS 2022

1-2 JUNE, TOULOUSE - FRANCE
DIAGORA CONGRESS CENTER

ORGANIZED BY

3AF

see since 1883

PROCEEDINGS

Editors

Mohamed Kaaniche	LAAS-CNRS - France
Philippe Cuenot	Continental Automotive, IRT Saint Exupéry
Marie de Roquemaurel	Airbus Defence & Space
Kevin Delmas	Onera
Jean-Marc Gabriel	Renault Software Labs
Adrien Gauffriau	Airbus
Christophe Grand	Onera
Eric Jenn	Thales Avionics, IRT St Exupéry
Benoît Morgan	IRIT

Contents

Program Committee

Ahiad, Samia	France	VALEO
Ainhauser, Christoph	Germany	BMW AG
Anguenot, Yves	France	Aerospace Valley
Armengaud, Eric	Austria	Armengaud Innovate GmbH
Asselin, Eric	France	Collins Aerospace
Baron, Claude	France	LAAS-CNRS, INSA, Toulouse
Baufreton, Philippe	France	Safran Electronics & Defense
Belmonte, Fabien	France	Alstom Transport SA
Bieder, Corinne	France	ENAC
Boissé, Sébastien	France	Thales Group
Boyer, Marc	France	ONERA
Braband, Jens	Germany	Siemens AG
Carsten, Thomas	Germany	Siemens
Cazorla, Francisco J	Spain	Barcelona Supercomputing Center
Chave, Olivier	France	TechnicAtome
Claraz, Denis	France	Vitesco Technologies France SAS
Comar, Cyrille	France	Adacore
Cormery, Patrick	France	ArianeGroup
Cuenot, Philippe	France	Continental
Cunha, Alcino	Portugal	University of Minho
de Roquemaurel, Marie	France	Airbus Defence & Space
Delmas, Kevin	France	ONERA
Dreiseitel, Stefan	Germany	Continental Teves AG & Co. oHG
Ducoffe, Mélanie	France	Airbus
Erts, Admin	France	NONE
Faucou, Sebastien	France	Université de Nantes
Florent, Meurville	France	Valeo
Frezouls, Benoit	France	CNES
Fuerst, Simon	Germany	BMW Group
Gabriel, Jean-Marc	France	Renault
Gallina, Barbara	Sweden	Mälardalen University
Gauffriau, Adrien	France	Airbus
Grand, Christophe	France	ONERA
Guetta, Olivier	France	Renault
Guiochet, Jérémie	France	LAAS-CNRS
Habli, Ibrahim	United Kingdom	University of York
Hochgeschwender, Nico	Luxembourg	German Aerospace Center (DLR)
Jan, Mathieu	France	CEA LIST
Jean-Louis, Boulanger	France	certifer

Jenn, Eric	France	IRT Saint Exupéry
Johnson, Chris	United Kingdom	Queen's University Belfast
Kaaniche, Mohamed	France	LAAS
Kuehne, Uwe	Germany	Airbus Defence and Space
Laarouchi, Youssef	France	EDF R&D
Lanusse, Agnes	France	CEA LIST
Le Calvez, Gilles	France	VALEO
Lecomte, Thierry	France	CLEARSY
Leconte, Bertrand	France	Airbus Operations SAS
Ledinot, Emmanuel	France	THALES Research & Technology
Lonn, Henrik	Sweden	Volvo Group
Mader, Ralph	Germany	Vitesco Technologies GmbH
Maillet-Contoz, Laurent	France	STMicroelectronics
Malenfant, Jacques	France	Sorbonne Université - LIP6
Mamalet, Franck	France	IRT Saint Exupery
Mangane, Laurent	France	3AF
Mekki-Mokhtar, Amina	France	ANSYS
Moreno, Christophe	France	Thales Alenia Space
Morgan, Benoît	France	IRIT
Mouy, Patricia	France	ANSSI
Mraidha, Chokri	France	CEA LIST
Nadjm-Tehrani, Simin	Sweden	Linköping university
Nanya, Takashi	Japan	The University of Tokyo
Navet, Nicolas	Luxembourg	University of Luxembourg
Niemetz, Michael	Germany	OTH Regensburg
Pagetti, Claire	France	ONERA
Palanque, Philippe	France	ICS-IRIT, University Toulouse 3
Parissis, Ioannis	France	Univ. Grenoble Alpes - Grenoble INP
Paulitsch, Michael	Germany	Intel
Pfeifer, Holger	Germany	fortiss GmbH
Picard, Celia	France	ENAC
Pinot, Frédéric	France	Hitachi rail STS
Pons, Philippe	France	Aerospace Valley
Popa, Mircea	Romania	Politehnica University of Timisoara
Poyet, Eric	France	Kalray
Prof. Dr. Mottok, Juergen	Germany	LaS^3, OTH Regensburg
Quéré, Philippe	France	Stellantis
Rochange, Christine	France	IRIT - Université de Toulouse
Rodríguez, Rafael	Spain	GTD Sistemas de Información, S.A.
Saez, Estelle	France	LIEBHERR
Shagdar, Oyunchimeg	France	VEDECOM
Stea, Giovanni	Italy	University of Pisa
Terraillon, Jean-Loup	Netherlands	European Space Agency
Totel, Eric	France	Supelec
Trapp, Mario	Germany	Fraunhofer
Traverse, Pascal	France	AIRBUS
Troubitsyna, Elena	Sweden	KTH
Van Der Linden, Frank	Netherlands	Philips Healthcare
Verdier, Damien	France	Vitesco Technologies France SAS
Vigouroux, David	France	IRT Saint-Exupery
Voget, Stefan	Germany	Continental Automotive GmbH
Warns, Timo	Germany	Airbus
Wartel, Franck	France	AIRBUS Defence and Space
Zennou, Sarah	France	Airbus

Session We.1.A

GPU

Wednesday 1st June

11:30

–

Amphithéâtre

Real-time high performance computing using a Jetson Xavier AGX

Cyril Cetre[1, 2], Florian Ferreira[2], Arnaud Sevin[2], Rémi Barrere[1] and Damien Gratadour[2, 3]

[1]Thales Research & Technology
[2]LESIA, Observatoire de Paris, Université PSL, CNRS, Sorbonne Université, Université de Paris, 5 place Jules Janssen, 92195 Meudon, France
[3]RSAA, Australian National University, Cotter Road, Weston, ACT2600, Australia

Abstract

While general purpose graphics processing units now embark tremendous amount of computing power, their use in real time applications is still a challenge. The COSMIC platform, developed in the context of adaptive optics control for giant astronomical telescopes is a demonstrated solution to perform real time computations using discrete GPUs while maintaining a high level of abstraction and modularity. An implementation on embedded platforms with the goal to reach an acceptable level of time-determinism would enable new real-time use cases in other application domains. In this regard, NVIDIA is offering a broad range of embedded systems on chip delivering great performance and compatible with the CUDA ecosystem. However, specific hardware and software features bring uncertainties regarding real-time performance. The approaches presented in this paper rely on COSMIC recipes to expose part of the underlying unconventional GPU programming model to reach real-time performance. It shows how Jetson Xavier performs on sub-millisecond complex pipelines made of several compute kernels, considering the limitations engendered by missing CUDA features as compared to discrete devices, leveraging unified memory to work around these hurdles and enabling several strategies for implementing real-time workflows on embedded GPU platforms.

Keywords: Real-time systems, graphics processing units, High Performance computing, embedded software, CUDA

Introduction

Real-time computing for adaptive optics

Adaptive Optics (AO) systems are used to compensate aberrations in real-time on optical systems. They are a core component of extremely large telescopes for astronomy. They aim at making partial compensation of image distortions induced by atmospheric turbulence in real-time using a set of computer controlled actuators, under the reflective surface of so-called deformable mirrors, to observe astronomical objects at high angular resolution and high contrast. As distortions and external constraints on AO are fluctuating on very short time scales, AO controllers, working in closed loop, need to infere the best actuators commands with minimum and stable time-to-solution (in the range of 1 ms or less). In addition, the complexity of AO systems exacerbates the need for a modular solution which provides the ability to realize flexible computing pipelines.

The COSMIC [1] platform was designed to cover the requirements of a wide variety of AO instruments considering these constraints. This platform relies on off-the-shelf high performance libraries and a modular approach, in order to minimize implementation cost and complexity while maximizing performance and throughput of applications requiring efficient GPU computations. It provides abstraction layers handling data transfers, inter-process communications and synchronisations between computation units of a given pipeline. Each computation of the pipeline is turned into an independent process, allowing individual monitoring and real-time pipeline modifications. The multi-process mechanisms are thoroughly optimized to obtain the best possible performance given the hardware configuration.

While COSMIC was originally designed for AO, its application to other domains remains perfectly valid as it is aiming at providing a framework for real-time computation regardless the pipeline. Therefore, evaluating the use of its programming methods on embedded platforms is meaningful, although opening the door to new challenges. An embedded implementation will have to ensure that performance is preserved with scarce resources while proposing possible solutions to overcome technical limitations. A stable implementation on systems on chip (SoC) could prove useful to implement simpler pipelines with specific use cases such as smart cameras working as wavefront sensors.

Embedded systems and real-time computing

With the exponential growth of the use of deep learning in various domains, SoC with integrated GPU (iGPU) are becoming very popular thanks to their rather high performance per watt. In particular, NVIDIA provides a broad range of consumer products using custom ARM CPU and iGPU. These SoC have many interesting features such as supporting the CUDA ecosystem. This makes cross-platform programming pretty straightforward with few differences between SoC and discrete GPUs (dGPU) [2]. As opposed to a dGPU, which is a separate device from the processor with dedicated memory, one of the main features of iGPU is to share memory with the CPU, which allows to reduce

the need for data transfers, although it opens the doors to a number of challenges as well considering the CPU activity may interfere negatively with the GPU computations.

In any case, achieving real-time GPU computing can be challenging since this technology offers very few safeguards on execution time determinism. That is why average jitter and Maximum Measured Execution Time (MMET) must be assessed carefully. Ensuring such features, combined with a high throughput is a significant hurdle to overcome before bringing GPU into time critical applications.

In addition, enabling workflows involving multiple processes on a GPU brings uncertainties regarding the end-to-end response time. Some functionalities like CUDA Multi Process Service (MPS) are offered to allow kernels from different processes to execute concurrently. However, MPS and some other features are not available on embedded NVIDIA SoC, making multi-process application portability unpredictable on such platforms.

Unconventional programming model for GPU

Ensuring time-to-solution repeatability with very low jitter for a complex pipeline can be particularly tough when relying on multi-process asynchronous GPU computations. While achieving a high throughput is possible when taking into account intrinsic overheads of GPU computing, such as memory transfers, kernel launches or synchronizations between the host and the device, reaching low performance jitter usually requires unconventional programming approaches, such as using persistent kernels [3] [4]. As the name implies, these kernels are not terminated between each iteration and just wait on the arrival of more data to be triggered again. However, such technique heavily depends on hardware features and the CUDA grid/block dimensioning must be handled within the kernel. As a consequence, persistent kernels must be redesigned when either the pipeline or the targeted hardware changes. A proposed trade-off between persistent kernels and traditional programming models is the use of a combination of GPU busy wait kernels and look ahead jobs scheduling, relying on the GPU scheduler to launch new kernels while avoiding CPU/GPU synchronization overheads [1].

This paper exposes this approach through a set of out-of-the box use cases, comparing the discrete and embedded behaviour in order to provide guidance about how to perform real-time multi-process computations with complex applications in an embedded environment.

Focus of this paper

This paper relies on previous work done with the COSMIC platform and extends the best practices implemented therein to provide a suitable solution for critical real-time applications running on embedded platforms in terms of average jitter, worst observed execution time and throughput.

Porting these recipes on Jetson Xavier AGX provides an overview of how architectural and software differences between embedded platforms and integrated GPUs can be worked around to achieve real-time performance.

We highlight these differences and show why it could be a serious impediment to determinism. We also propose workarounds using multi-thread or multi-process implementations of GPU inter-process communication through busy-waiting.

Finally, we evaluate through benchmarking the performance and overall behaviour obtained using different methods for efficient inter-process communication

Related Work

Using GPUs for real-time applications is not straightforward, as it has not necessarily been designed to minimise MMET, although modern GPUs are slowly overcoming technological limitations with increased features and capabilities. [5, 6].

Unlike dGPUs, the CPU and the iGPU share the same SoC DRAM on Tegra devices. As a consequence, CPU activity may interfere negatively with GPU computation and conversely [7]. That is why some authors have proposed to improve determinism by protecting GPU applications from memory throttling through custom schedulers [8, 9]. In addition, several studies are aiming to unveil closed-source details of GPU schedulers to get a better understanding of their behaviour. [10, 11].

On a positive note, shared DRAM implies that data transfers from host to device are avoidable. In such circumstances, pinned host memory accessed from GPU will not have copy overheads and will be bounded by the same bandwidth as memory allocated from device code. Combined with CPU shared memory, it is used in this contribution as a workaround to bypass the lack of CUDA Inter-Process Communication (IPC) features. In addition, previous work has shown that using pinned host memory on embedded systems can be a way to increase determinism by reducing memory requirements of GPU programs [12].

Real-time multi-process computing on GPU

Inherited from a project having strong requirements for scalability, maintainability and modularity [3], COSMIC was designed as a framework able to overcome the underlying limitations by supporting separate kernels that can run either concurrently or sequentially, with an efficient synchronization mechanism. However, a complex software architecture does not necessarily scale well when GPU computations are involved depending on the exact setting.

For instance, the default behaviour of CUDA is not suited for real-time multi-process applications. As every process has its own CUDA context only one context can run at the same time on the GPU, the device has to switch constantly between them, leading to extra jitter. In addition, running kernels with a small number of threads will lead to poor GPU performance as another kernel could have run concurrently. For a single process (and single context) application, the CUDA streams were created to overcome this issue and enable the overlapping of kernels executions.

In the case of multi-process applications, NVIDIA has made various tools available to improve the device management. In this regard, the next sub-section offers an overview

of these CUDA features providing a good understanding of their implication and why they are critical for real-time.

CUDA MultiProcess Service (MPS)

CUDA MPS is a binary-compatible client-server runtime implementation of the CUDA API. MPS is especially useful when multiple processes are making use of the GPU, in particular when these processes underutilize GPU resources. Its main benefits are the following :

- Kernels and memcopy operations from different processes may overlap on the GPU.
- MPS handles only one GPU context and set of scheduling resources between its clients instead of one context for each process. This removes the need for context switching between two GPU kernels.

CUDA Interprocess Communication

Part of the CUDA Toolkit and available since compute capability 2.0, CUDA IPC enables sharing device buffers between multiple processes. While the COSMIC memory manager implementation relies on CUDA IPC to share device buffer between processes, these calls are not supported on Tegra device.

GPU busy wait synchronization

At some point, a process will have to wait for another to complete, thus needing to synchronize. Previous work on the COSMIC framework provided an assessment on which kind of synchronization gives the best response time in order to maximize performance [1]. From this study, it appears that busy wait synchronization (meaning different processes actively wait for data to be written on the GPU through memory polling) shows a significant latency improvement compared to a regular CPU based POSIX semaphore synchronization. It has several benefits :

- The pipeline processing is now fully asynchronous from the CPU, allowing the user to hide kernel launch latency by accumulating jobs on the GPU scheduler in advance.
- A CPU, even isolated is still more likely to get non preemptible interruption request from the OS kernel compared to a GPU which is dedicated to computations.

On the other hand, each busy wait process is using a GPU streaming multiprocessor to spin on a given location in memory which is increasing system occupancy and resource consumption.

Figure 1 and 2 shows how busy waiting with CUDA MPS enabled performs compared to POSIX semaphore implementation, revealing non negligible latency improvement on a NVIDIA DGX server (286 μs of average execution time for semaphore and 235 μ s for GPU Busy wait). Even though the semaphore synchronization shows a better average jitter (2.1 μs for semaphore and 4.4 μs for GPU busy wait), this synchronization process is more prone to jitter peaks (semaphore : 35 μs of peak to valley and 30 μs for GPU busy wait). In addition, the NVIDIA DGX server performs especially well to minimize this kind of interference, which tends

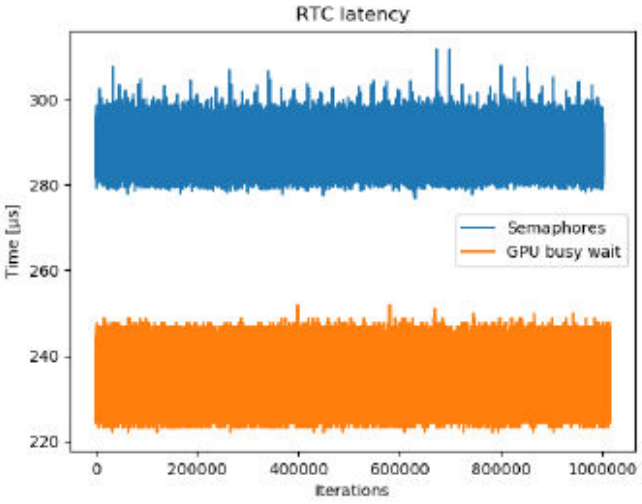


Figure 1 Time-to-solution execution profiles obtained with COSMIC in the context of the AO application, using 2 different synchronization mechanisms over 1M iterations [1]

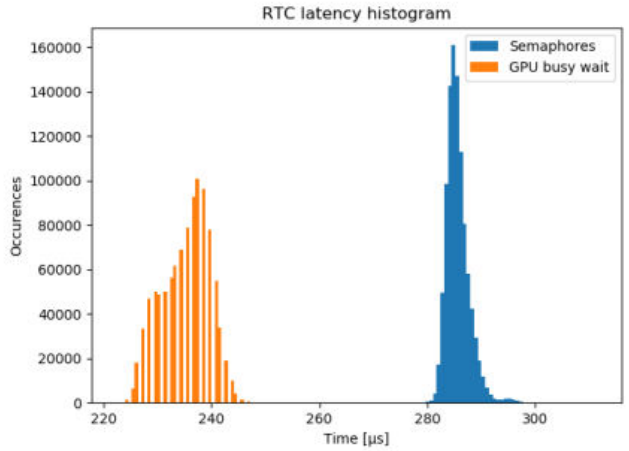


Figure 2 Time-to-solution histogram obtained with COSMIC in the context of the AO application, using 2 different synchronization mechanisms over 1M iterations [1]

to show higher peak to valley in the case of semaphore synchronization as compared to other GPU equipped servers. As a consequence, the GPU busy wait is preferable both in terms of latency and jitter stability.

Considering the sub-millisecond response time requirement, such gains represents a great asset and are hardly dispensable. Reproducing such results on an embedded SoC would bring us closer to hard real-time embedded GPU computing, although the road ahead is full of obstacles.

The challenge of embedded GPU computing

As GPUs are growing in complexity with features including dedicated cores for specific operations including tensor

cores for matrix multiplication or RT cores for raytracing, NVIDIA is regularly proposing new SoCs featuring the latest innovations. As a consequence, each SoC has a specific hardware architecture (CPU and GPU) which makes it difficult to predict performance on the targeted board without comprehensive testing. That is why this paper will focus on NVIDIA Jetson Xavier AGX, which is providing some features (for instance I/O coherency) which were not available on previous NVIDIA boards.

The Jetson Xavier AGX

Released in 2018, the Jetson Xavier AGX is currently the most powerful embedded GPU platform available on the market. It is featuring within its Xavier Tegra SoC a Volta GPU with 512 CUDA cores and 8 streaming multiprocessors.

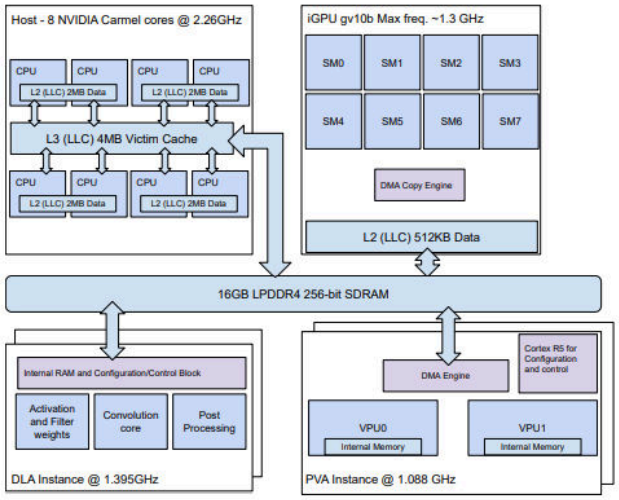


Figure 3 Block diagram of Jetson Xavier complex (credits : [13]

The CPU is composed of 8 ARM 8.2 architecture compliant cores. Regarding the cache hierarchy, it is important to notice that the CPU complex is composed of 4 islands. Although each core has its own private L1 cache, the CPU L2 cache is shared between two threads. Two cores from the same island will interfere if both are executing highly bandwidth dependant tasks. As a consequence, the combined bandwidth of two CPUs of the same island is worse compared to a single core doing heavy bandwidth computations paired with an idle one. [13].

This is why it is recommended to always isolate CPUs by island to avoid extra interference from shared L2 cache on this platform.

Unlike discrete GPU (dGPU), CPU and integrated GPU (iGPU) share 16GB of DRAM clocked at most at 2133 MHz, reaching a theoretical bandwidth of 137 GB/s.

Shared CPU/GPU memory

From a software point of view, shared memory does not mean that a CPU pointer is accessible from GPU and conversely. Depending on how memory is allocated, the cache behaviour differs and memory might not be accessible from both the host and the device. Similarly to dGPU, memory allocated with *cudaMalloc* is not accessible from the host and a buffer allocated with *cudaMallocManaged* will return a buffer accessible from both sides. The main difference with dGPU is that there is no hidden memory copies between the host and the device and the source code should be adapted to make copies only when unavoidable. NVIDIA provides detailed documentation about the cache behaviour depending on how the memory was allocated [2]. Starting with the Xavier architecture, Tegra boards are featuring I/O Coherency which allow I/O devices such as the GPU to read the latest updates in CPU caches. This is beneficial in our case as it allows to cache CPU pinned host memory and to register an existing host memory range for use by CUDA, which is crucial in our proposal to bypass the lack of CUDA IPC.

CUDA IPC/MPS unavailable on Jetson

At the time this paper was written, aforementioned multi-process features are not available on NVIDIA embedded systems. As a consequence, multi-process GPU busy waiting is not viable as it is introducing strong jitter interference due to GPU context switching. We will see in the next sections various options we explored to get performance similar to the classical multi-process approach, followed by experimental results.

Multi-process real-time computing on Jetson embedded platforms

GPU buffer sharing on Jetson Xavier

When trying to reproduce complex GPU pipelines on Jetson, the first obstacle lies in being able to share GPU buffers among processes which is usually done on a discrete device through CUDA IPC. So far, this feature is not available on Tegra SoC although sharing device memory is a core feature of the platform.

Fortunately, on Tegra devices with I/O coherency [2] (starting with the Jetson Xavier, not applicable on previous devices) it is possible to register an existing host memory range for use by CUDA. The identified workaround takes advantage of the CPU and GPU unified memory to keep decent performance while using a mix of POSIX shared memory and zero-copy mechanism which allows a GPU to fetch pinned host data.

Using GPU kernels to access pinned host memory is coming with drawbacks. First, it will result in separated pointers for host and device calls, although using the same memory space. Second, pinned host memory accessed by the GPU will not be cached. As a consequence, user must take extra care when accessing memory to avoid any performance drop.

GPU busy-waiting efficiency on Jetson Xavier

Although the aforementioned shared GPU buffers enables GPU busy wait synchronization, results showed really poor performance using a straightforward implementation.

Recording sub-milliseconds computations through a non-intrusive timer (meaning that it is using a separate process in order to avoid interference with the main pipeline) requires responsive measurements.

Unfortunately, measuring those events on GPU turns out impossible as the GPU timer is unable to start and stop at the right moment while performing sub-millisecond measurements. Considering that this timer has its own process and CUDA context, the GPU scheduler will not necessarily give the hand back to the timer process right after the computation process sends the stopping signal, thus providing misleading results.

Although CUDA preemption mechanism and context loading are not publicly disclosed, we know that the GPU is handling multiple clients (i.e., multiple processes requesting GPU resources) with a time-slicing behaviour. In addition, a GPU always serialise two kernels from different processes, even though the resources to run them in parallel are available.

As a consequence, the scheduler regularly interrupts the pipeline computations to perform GPU busy waiting from other computing units, leading to unintended performance and jitter degradation. Considering that multi-process busy-wait synchronization relies heavily on launching small kernels from different processes, it is impossible to achieve a satisfactory level of determinism in highly constrained environments using regular busy-waiting without CUDA MPS.

Possible workarounds

Thanks to unified memory, potential workarounds are available although they don't completely overcome the lack of CUDA MPS.

Multiple-context CPU active-wait strategy

It is possible to keep going with multiple CUDA contexts. In this case, the important aspect is to avoid doing active waits on GPU at all costs as explained in the previous subsection. Active waiting with CPU on a GPU data buffer is working, although forcing CPU thread spinning. This avoids a synchronization between the host and the GPU as notifications are sent through the device. However, the GPU won't be able to schedule kernels ahead of time as the CPU is now blocking the execution. Adding context switching, this strategy will introduce some overhead compared to straightforward GPU busy-wait.

Semaphore strategy

While having an overhead compared to active wait strategy, implementing POSIX semaphore based signals remains perfectly viable and will suffice in most cases.

Multi-thread strategy for real-time computing

After establishing that the principal constraint is to keep only one CUDA context to get best results when using the GPU busy waiting technique, it seems that an approach relying on both multi-thread and multiple streams is an approach to consider. CUDA streams were specifically designed to allow concurrency on a single process, which is a requirement for this kind of busy-wait.

On the positive side, a multi-threaded implementation removes software dependency to inter-process features (CUDA MPS and CUDA IPC). As a consequence, it is deployable on both discrete and integrated GPU environments as opposed to the multi-process approach.

However, it must be pointed out that the GPU behaviour is not strictly identical. For instance, considering the GPU busy-wait mechanism, it is very important to synchronize the CPU with the CUDA execution at some point. A CUDA context has a limited queue depth in terms of kernel scheduling. When this limit is reached, a forced synchronization occurs on the CPU before it can be able to enqueue more kernels. It is thus very easy to end up with a deadlock, with busy wait kernels exceeding the queue depth leading to the impossibility to send notifications. This limitation must be taken into account by the implementation.

The described behaviour may not be the only difference between multi-thread and multi-process strategies. Unfortunately, as for the multi-process preempting behaviour, CUDA stream scheduling is scarcely documented by NVIDIA although several studies are working toward exposing it in more details [14]. Thus, it is likely that new behaviour specificities will be empirically discovered in the future.

Regarding performance, results with a multi-thread implementation appeared to be very similar to regular MPS multi-processing, which is encouraging.

Environment setup and results

Use Case

For the sake of reproducibility and modularity, the decision was taken to implement a simplified pipeline while keeping only COSMIC core synchronization mechanisms. It allows us to propose multiple test cases, with both embedded and iGPUs whenever necessary.

Figure 4 shows an example of how a given sample test behaves regardless of the synchronization approach and its kind of parallelization, being either multi-threaded or multi-processed. The basic architecture of the pipeline is the following :

- A camera emulator which sends notifications at a given framerate. The notification to start computation is sent to both the timer and the first computation unit.
- A pipeline that can be composed of one or more compute units. In our test environment, it essentially performs matrix vector multiplications (MVM). It sends a notification to the timer once it is done.
- A timer which gets notifications from the camera emulator to start and from the last computation unit to stop.

Each test is performed first in a multi-processed environment, meaning each unit (timer, camera, first MVM, second MVM and so on) are launched as separated processes. The

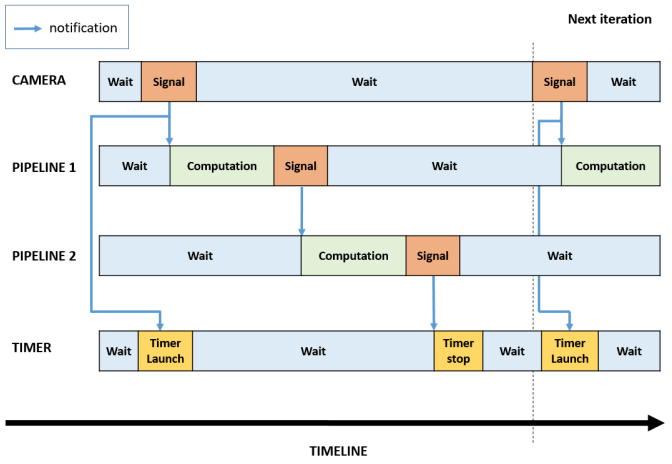


Figure 4 Timeline of a test with two computing units

only way these units have to communicate is through shared memory and signals to synchronize, which is done either through POSIX semaphores or GPU busy-wait. In a second experiment, the test is performed with the multi-threaded approach : the computation units are launched in the same process. However, each unit is given a distinct CUDA stream which allows kernel execution overlapping. Both synchronization methods are performed as well.

WCET and MMET

The Worst Case Execution Time is a term typically used in critical real-time systems where reliability and safety is paramount. As of today, it is very unlikely that an embedded environment such as a Jetson can allow a demonstrated WCET such as it is meant in avionics. In order to avoid misconception about the testing process, this paper is addressing Maximum Measured Execution Time (MMET), meaning the worst case measured in the test session with a representative number of executions (1 million). Figure 5 shows how MMET does not necessarily shows the worse possible case, compared to the WCET.

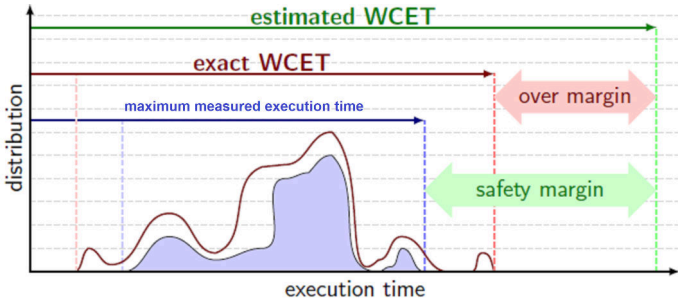


Figure 5 This paper uses maximum measured execution time (MMET) figure credit : Thales Research Technology

Timing measurement

Timing events with asynchronous CPU and GPU interactions can be surprisingly difficult. In order to record overheads between a computation unit sending a signal and another receiving it, the timer must stay in a separate thread. It is also a great asset as a timer needs to synchronize with CPU and GPU in order to get execution time, which is unavoidably introducing overhead. This kind of timer needs the first pipeline unit to send a notification to launch measurement and the last one to send a another one to stop it. The downside is that the timer is not bounded to computation anymore and might miss some iterations at some point if is not awaiting for a signal when a notification is sent.

Making the timer independent of computation units will not introduce interference to computation or forcing unintended CPU/GPU synchronization, ensuring measurement closest to the actual computation time.

The semaphore synchronization approach is measured through the C++ high_resolution_clock API while the GPU busy wait approach is using CUDA events.

Results will be shown through two different displays that highlights different kind of information :

- Histograms are helpful to get a grasp of an approach latency and jitter.
- Execution profiles include each iteration execution time. It is useful to show how an approach is affected by jitter. The wider the plot "line", the more jitter this approach gets. it also shows infrequent outliers, that would not appear on histograms.

Environment

Table 1 and 2 show both configuration used in the following tests.

OS	CentOS 8
Linux kernel	4.18.0-193.19.1.el8 2.x86 64
CPU	15 Intel(R) Xeon(R) CPU E5-2698 v4 @ 2.20GHz
CUDA version	11.5
GPU	Tesla V100

Table 1 Benchmark environment setup with dGPU

OS	Ubuntu 18.04
Linux kernel	4.9.253-rt168-tegra
CPU	8-core ARM v8.2 64-bit
CUDA version	10.2
GPU	512-core Volta integrated GPU

Table 2 Benchmark environment setup on Jetson Xavier AGX

Optimizations for real-time

Several optimisations are done to achieve real-time performance. This includes :

- CPU isolation at boot.
- Processes scheduling set to FIFO with high priority.
- Running CUDA MPS server (for dGPU only).

In order to get comparable results across all experiments for a given setup (dGPU or Jetson), MVM are based on a fixed size. The framerate of the camera is set to always let the pipeline finish its job before notifying for the arrival of a new image.

To reach sub-millisecond computation time with both settings, the matrix size is reduced with the Jetson Xavier, giving the following sizes :

- 4096 X 12500 with the Tesla V100.
- 4096 x 1562 with the jetson Xavier.

Experimental results with the Tesla V100

The first step requires to study how different implementations behave with dGPUs compared to known results (Multiprocess with Active wait synchronization and CUDA MPS). Figures 6, 7 and Table 3 shows an experiment with 1M iterations of the same test performed with different parallelization and synchronization techniques. The pipeline is the simplest possible, composed of one camera, one processing unit performing a matrix-vector multiplication (MVM) and a non intrusive timer.

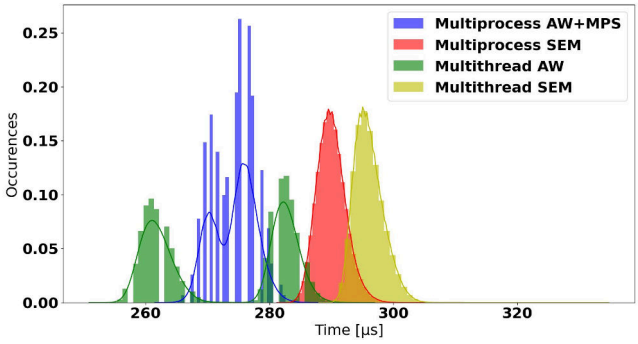


Figure 6 Normalized histograms with their estimated probability density function (solid lines) of a pipeline composed of one MVM unit with various synchronization approaches (SEMaphore and Active Wait) over 1M iteration on DGX server

The best results in terms of latency come from active wait synchronization (AW) regardless if using multi-thread or multi-process approach with 274.3 μs of Mean Execution Time (MET) for multi-process and 272.4 μs for multi-thread. Even with this extremely simple pipeline, it still shows noticeable improvement regarding the worse case compared to semaphore based synchronization (gap < 25µs between MMET and MET). However, the histogram of both active

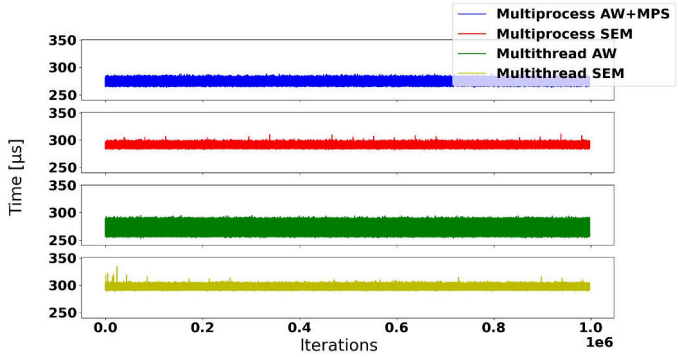


Figure 7 Execution profiles over 1M iterations for various scheduling and synchronization approaches on the DGX server

Compute type	Mean execution time (μs)	MMET (μs)	Average jitter (μs)	Jitter Peak-to-Valley (μs)
Multiprocess AW + MPS	274.3	288.8	3.5	25.6
Multiprocess SEM	290.0	310.8	2.3	29.2
Multithread AW	272.4	294.9	10.7	42.0
Multithread SEM	296.0	334.3	2.3	45.9

Table 3 Summarised results of different synchronization approaches on DGX server

wait synchronization approaches shows that results are scattered across two Gaussian distributions. That is why the average jitter of multi-thread active wait is considerably higher compared to the three other approaches. This behaviour is not yet explained as it doesn't look like an external interference, which would have been highlighted with semaphore approach as well. The multi-thread approach shows slightly higher Peak-To-Valley jitter (P2V – MMET minus the minimum measured time) with 42.0 μs with AW synchronization and 45.9 μs with semaphore synchronization. The multi-process approach (25.6 μs for AW and 29.2 μs) is slightly more stable.

With these considerations, the active-wait based synchronization is still outperforming semaphores in terms of latency and MMET and remains the preferred option with a dGPU. The multi-process approach shows slightly more stable results, although these gains are not sufficient to discard the multi-threaded approach.

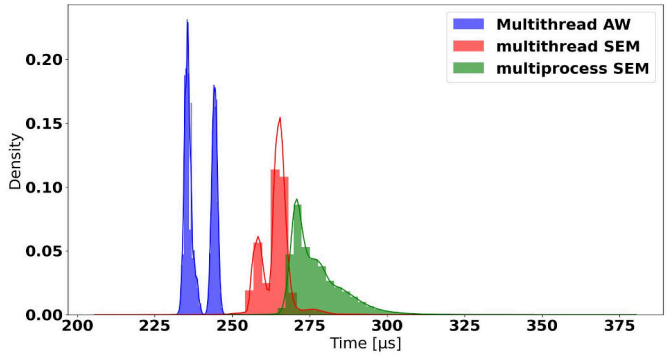


Figure 8 Normalized histograms with their estimated probability density function of a pipeline composed of one MVM unit with various synchronization approaches (SEMaphore and Active Wait) over 1M iterations on the Jetson Xavier

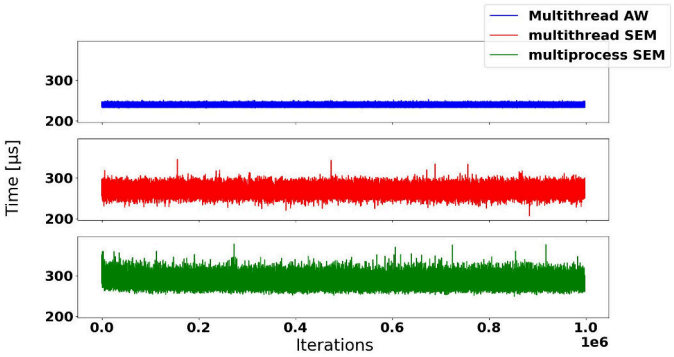


Figure 9 Execution profiles over 1M iterations for various scheduling and synchronization approaches on the Jetson Xavier

Compute type	Mean execution time (μs)	MMET (μs)	Average jitter (μs)	Jitter P2V (μs)
Multithread AW	239.4	252.5	4.4	21.5
Multithread SEM	263.9	346.4	4.8	139.8
Multiprocess SEM	277.4	378.8	8.4	129.7

Table 4 Summarised results of synchronization techniques on the Jetson Xavier

Experimental results on Jetson Xavier

As MPS is not available on Jetson Xavier, only 3 cases can be assessed. We performed the exact same experiment on this hardware and the results are presented in figures 8, 9 and Table 4.

When comparing results obtained on both the dGPU and iGPU platforms, a noticeable difference is that semaphore synchronization approaches lead to less stable results on the iGPU case with a up to 101.4μs gap between the MET and the MMET for multi-process and 82.5 for multi-threaded based approach. The multi-threaded active wait synchronization is performing best with only 13.1 μs difference. The P2V jitter shows the same kind of outcome with 21.5 μs for multi-thread AW, 139.8 μs for multi-thread SEM and 129.7 μs for multi-process SEM. Even though CPU cores are isolated on Jetson, semaphore synchronization is more prone to interference compared to the dGPU case. Multi-thread results are still considered satisfactory results, both in terms of MMET and average time to solution.

Regarding semaphore performance, it appears that multi-thread performs slightly better compared to multi-process. When comparing to the dGPU (Figure 6), the reverse occurs with multi-process semaphore delivering slightly better execution time. It may be caused by how CPU contexts are handled on both platforms (introducing CPU context switch) and probably some CPU interference, more pronounced on Jetson Xavier. We will investigate further such behavior in future work.

Multiprocess with hybrid active wait

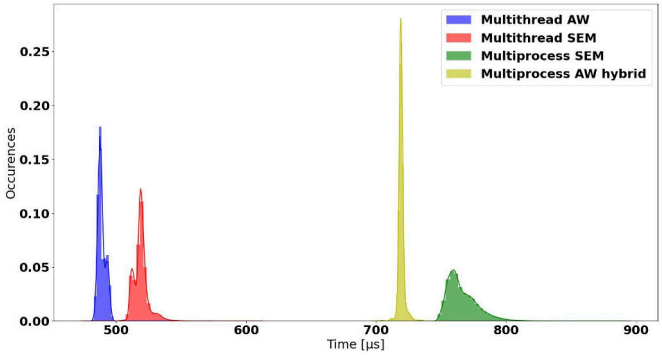


Figure 10 Normalized histograms with their estimated probability density function of a pipeline composed of two MVM units with the proposed hybrid (CPU/GPU) active wait compared to other synchronization approaches (Semaphore and active wait) on the Jetson Xavier

As aforementioned, the way the GPU handles multiple processes and CUDA contexts is not suitable for GPU busy waiting synchronization. A proposed workaround is to take advantage of shared memory to avoid busy waiting with GPU threads.

This hybrid active-wait strategy relies on letting the CPU do the busy waiting on data shared with the GPU while notifications are sent through GPU kernels. This allows the GPU scheduler to stop constantly switching between multiple CUDA contexts requiring GPU usage.

Although this implementation should not benefit from the latency hiding obtained with look ahead kernel launch and

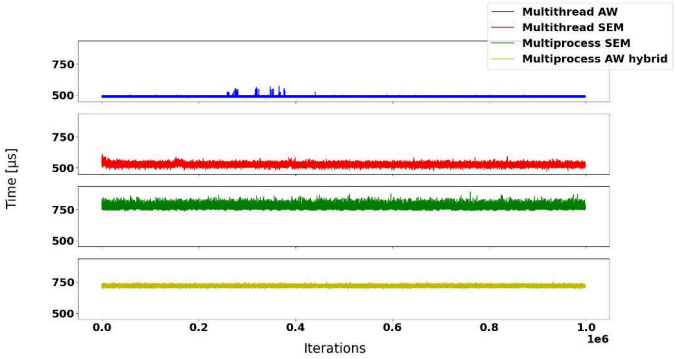


Figure 11 Execution profiles over 1M iterations of the hybrid active wait compared to other synchronization approaches on the Jetson Xavier

Compute type	Mean execution time (μs)	MMET (μs)	Average jitter (μs)	Jitter P2V (μs)
Multithread AW	488.9	572.4	3.0	93.2
Multithread SEM	519.0	611.1	5.8	116.9
Multiprocess SEM	766.16	893.3	11.6	158.9
Multiprocess AW hybrid	719.3	752.5	2.4	56.1

Table 5 Summarised results of hybrid synchronization techniques on Jetson

still partly suffers from context switching overheads, using the GPU for notification may help to reduce noxious interference coming from the CPU, plus avoiding some CPU/GPU synchronizations compared to semaphore approach.

It is important to note that such implementation is possible only with a sequential pipeline as concurrent kernels have to be implemented through CUDA streams, otherwise the resulting performance and determinism will be severely affected.

In this test case (Figure 9, 10 and Table 5), the new mechanisms described are implemented (results named Multiprocess AW hybrid), resulting in a new approach for multi-process synchronization. In order to expose how different kinds of synchronization affect performance, two sequential computing units are instantiated thus reproducing exactly the setting for Figure 4. The first noticeable thing is that the gap between multi-threaded and multi-processed synchronization increases as the pipeline complexity grows. Regarding latency, the multi-thread approach shows best results regardless the kind of synchronization (488.9 μs for active wait and 519.0 μs for semaphore) where the multi-process approach is taking an extra 200 μs for the exact same computations (766.16 μs for semaphore and 719.3 μs for the new hybrid synchronization). This brings to light the cost of GPU context switch which never happens in a multi-threaded application.

However, it also shows that synchronization on the GPU is an environment less prone to interference. The difference between the MMET and MET of the hybrid active wait approach is 33,2 μs where the second best being multi-thread AW with a difference of 83,5 μs. Finally, semaphore synchronization with multi-thread gives a difference of 92,1 μs and the multi-process approach comes last with a 127,14 μs difference. The P2V jitter confirms this result with 56.1 μs for the multi-process hybrid AW, 93.2 μs for multi-thread AW, 116.9 μs for multi-thread SEM and finally 158.9 μs for multi-process SEM.

As a consequence, the hybrid active waiting, although showing marginal latency improvement compared to using multi-process semaphore is still far from reaching multi-threaded approaches performance and cannot be proposed as a universal replacement for active wait with CUDA MPS. Still, it is a great replacement for pipelines that need to communicate through multi-process and is a good solution in terms of determinism.

Conclusion

The best practices introduced in the COSMIC framework enable the realization of complex compute intensive pipelines while keeping a high level of modularity thanks to its efficient synchronization mechanism.

It delivers latency improvements while keeping a stable jitter with discrete GPUs, but its use on embedded platforms needs to be adapted in order to work around missing features on such hardware and supporting ecosystem. Best results are obtained when reproducing the same mechanism with a multi-thread implementation, instead of multi-process. This allows the CUDA environment to keep a single context, avoiding both kernel preemption and context switching. However, the behaviour may differ between multiple processes and multiple streams implementations depending on how the scheduler handles kernels.

The embedded Jetson CUDA package is still missing some critical features of multi-process programming. However, it is possible to get it working using using the hybrid active wait strategy detailed in this paper. While the user needs to be aware of the corresponding performance trade-off, making sure only one context is executed at a given point in time, a pipeline can be built with processes sending notifications using the GPU by taking advantage of shared memory between CPU and GPU. Such approach performs better than synchronization through CPU semaphores both in terms of jitter and latency.

Future work will focus on further testing these new approaches to get a better understanding of embedded platforms behaviour in order to get closer to true GPU real-time determinism.

Acknowledgements

This work is sponsored through a grant from project 873120, a.k.a. Rising STARS, funded by European Commission under program H2020-EU.1.3.3 coordinated in H2020-MSCA-RISE-2019.

References

[1] F. Ferreira. Hard real-time core software of the AO RTC COSMIC platform: architecture and performance. *Proceedings of SPIE - The International Society for Optical Engineering*, 2020.

[2] NVIDIA. Cuda for tegra. https://docs.nvidia.com/cuda/cuda-for-tegra-appnote/, 2021.

[3] Julien Bernard. Design and performance of a scalable GPU-based AO RTC prototype. *Proceedings of SPIE - The International Society for Optical Engineering*, 2018.

[4] Allen Todd. Improving Real-Time Performance with CUDA Persistent Threads (CuPer) on the Jetson TX2. *Concurrent Real-Time White Paper*, 2018.

[5] Yang Ming. Avoiding Pitfalls when Using NVIDIA GPUs for Real-Time Tasks in Autonomous Systems. *ECRTS 2018*, 2018.

[6] Paweł Czarnul. Investigation of parallel data processing using hybrid high performance CPU+GPU systems and CUDA streams. *Computing and informatics*, 2020.

[7] Ali Waqar. Protecting Real-Time GPU Kernels on Integrated CPU-GPU SoC Platforms. *ECRTS 2018*, 2018.

[8] Kenjić Dušan. One Solution for Deterministic Scheduling on GPU for Automotive Algorithms. *2021 Zooming Innovation in Consumer Technologies Conference*, 2021.

[9] Jason Baietto. Real-Time Linux: The RedHawk Approach. *Concurrent Real-Time whitepaper*, 2019.

[10] Tanya Amert. Gpu scheduling on the NVIDIA TX2: hidden details revealed. *IEEE Real-Time Systems Symposium*, 2017.

[11] Ignacio Sanudo Olmedo. Dissecting the CUDA scheduling hierarchy: a Performance and Predictability Perspective. *RTAS*, 2020.

[12] Vance Miller. Determinism in GPU Programs, Real Time Applications on the NVIDIA Jetson TK1. *Senior Honors Thesis, University of North Carolina*, 2016.

[13] Nicola Capodieci. Contending memory in heterogeneous SoCs: Evolution in NVIDIA Tegra embedded platforms. *IEEE Real-Time Systems Symposium*, 2020.

[14] Nathan Otterness. Inferring Scheduling Policies of an Embedded CUDA GPU. *Department of Computer Science, University of North Carolina*, 2017.

PasTiS: building an NVIDIA Pascal GPU simulator for embedded AI applications

Michaël Adalbert*†, Thomas Carle†, Christine Rochange†
*IRT SystemX, Palaiseau, France
†IRIT - Univ. Toulouse III - CNRS, Toulouse, France, name.surname@irit.fr

Index Terms—GPU, cycle-accurate simulator, timing analysis

***Abstract*—We present PasTiS, a simulator for the NVIDIA Pascal GPU architecture family, with a focus on timing simulation. PasTiS supports a subset of the Pascal ISA, sufficient to simulate the execution of neural networks. We present this subset, as well as the underlying microarchitecture that we modelled using information available from NVIDIA, from scientific publications, and from our own experiments. We demonstrate the precision of the simulator by comparing it to measurements on the NVIDIA Jetson TX2 development board, on neural network applications.**

I. INTRODUCTION

Real-time systems are increasingly embedding machine learning software which requires a huge computing power. For example, the control systems of autonomous vehicles rely on neural networks (NNs) to detect roads and objects, compute trajectories and plan the actions to be performed. These algorithms are computation-intensive and inherently parallel, which drives the industry to adopt massively parallel hardware such as many-core and GPU accelerators. In particular, GPUs have received a lot of attention these last years, both from the industry and from the real-time research community.

Scheduling tasks in a timing-critical system, so as to ensure that they will meet their timing constraints, requires being able to determine their respective worst-case execution time (WCET). Various approaches to WCET analysis exist and are based on static analysis techniques and/or measurements [10]. The estimated WCET can then be deterministic (i.e. expressed as a single upper bound) or probabilistic (i.e. several WCET values are produced, each associated to a probability of being exceeded) [9]. In this paper, we focus on systems where strict upper bounds on execution times are needed, and thus consider static WCET analysis approaches.

Static WCET analysis aims at determining invariants on the code of the task under analysis. Some invariants are related to the software (e.g. loop bounds), others to the state of the hardware (processor, cache memories, etc.). They are all used to build an integer linear program (ILP) that maximizes the execution time of the task over all the possible paths in the control flow graph (CFG) [17]. OTAWA is an open-source framework that offers many built-in facilities to generate WCET analysis tools [4].

This work was partially supported by the ANR LabEx CIMI (grant ANR-11-LABX-0040) within the French State Programme "Investissements d'Avenir."

Computing hardware-related invariants requires modeling the behaviour of the computing platform. This is usually done manually by translating the knowledge we have of the hardware (from documentation provided by the processor manufacturer or designer) into a formal model, although automatic translation from a VHDL specification of the processor (when available) has also been considered [24].

However, most of existing work considers CPU-only platforms. The very specific execution model of GPUs (Single Instruction Multiple Threads - SIMT) requires a substantial revisiting of hardware models. First, the lockstep execution of batches (*warps*) of threads requires rethinking the concept of basic block and instruction sequence due to possible branch divergence: all the threads in a warp do not necessarily follow the same control flow. Second, GPUs implement hardware scheduling schemes for warps and blocks of threads that must be accounted for in WCET estimations. Third, their memory system is noticeably different from that of standard CPUs and requires specific analyses.

The closed nature of most GPUs and the lack of official documentation on their micro-architecture have slowed down the understanding and modeling of their micro-architectural behaviour, which plays a crucial role in the execution time. These reasons explain that so far, no decent (industrial or academic) WCET analyzer for GPU-accelerated code is available.

As part of the French national *Confiance.ai* [1] program that brings together industry and academic researchers to build trustworthy Artificial Intelligence, we ambition to extend static WCET analysis techniques to GPUs. The work presented in this paper was mainly supported by the Labex CIMI[2] through the AVATAr project, and is a first step towards this objective: we show how we were able to conduct experiments to uncover some of the execution mechanisms and hardware parameters of an NVIDIA Pascal GPU and how we have built a cycle-level simulator that is able to run CUDA programs. We compare the execution times evaluated by the simulator to those measured on a Jetson TX2 board and demonstrate the accuracy of our model.

The insights provided in the paper can be used by industrial actors to assess the viability of using a GPU in their embedded systems: they give a clearer view of how a GPU program is actually executed and of the specificity with respect to CPU

[1] https://www.confiance.ai/en/
[2] https://cimi.univ-toulouse.fr/en/

execution. Although we focus on a particular NVIDIA GPU, the general mechanisms that we describe are common to all GPUs.

Contributions: In this paper, we present how we have proceeded to understand the behavior of an NVIDIA Pascal GPU and how we have used this knowledge to develop a cycle-level simulator called PasTiS (Pascal Timing Simulator).

Section II describes the general organization and execution model of GPUs, with a focus on our target (NVIDIA Pascal architecture). In Section III, we detail two key aspects of the GPU behaviour (thread divergence and accesses to the shared memory), and how we proceeded to understand them. We then introduce the PasTiS simulator in Section IV, and evaluate its performance in Section V. Related work is discussed in Section VI. Section VII concludes the paper.

II. GPU ORGANIZATION AND EXECUTION MODEL

In this section we present the global organization and execution model of GPUs. We borrow the NVIDIA terminology which is widely accepted in the community, but similar concepts exist in GPUs from other manufacturers.

A. Heterogeneous computation

At the highest level, a GPU is an accelerator on which a CPU can offload functions called *kernels*. A kernel is specified in a particular language (or a language extension, such as CUDA or OpenCL) that enables the description of parallel computation. The offloading of a kernel to a GPU generates threads that all execute the same code. The number of threads is specified by the programmer, as a number of *blocks* and a number of threads per block, and should be as high as possible in order to fully benefit from the GPU's resources. This is illustrated in Figure 3. In this example, a CPU function (*lines 2-11*) modifies each element in an array, depending on its initial value. Instead of processing elements sequentially, it invokes a GPU kernel (*line 7*) for a number of threads equal to the array size (`n × BLK`). The code of the kernel is given on *lines 12-18*. Each thread executing this code determines its identifier (*line 13*) – a value between 0 and `n × BLK`, and according to the initial value of the element (*line 14*), it computes its new value (*lines 15* and *17*). The CPU and the GPU have distinct main memories and the GPU cannot access the CPU memory. For this reason, the CPU function has to allocate space in the GPU memory (*line 4*) and to transfer input data from the CPU memory to the GPU memory (*line 5*) and output data from the GPU memory to the CPU memory (*line 8*).

B. General organization

On the hardware side, a GPU is a collection of clusters called *Streaming Multiprocessors* (SMs). Figure 1.a shows that the NVIDIA Pascal GPU of the Jetson TX2 contains two SMs, that share an L2 cache. Each SM contains in turn four processing blocks, denoted SMPs, that share an L1 instruction cache and two L1 data caches (each shared by two SMPs), as well as a fast multi-banked memory referred to as the *shared memory* (Figure 1.b). An SMP includes 32 *Cuda Cores* (CCs) that perform ALU and floating-point (32- and 16-bit) operations, a 64-bit FP unit, 8 Special Function Units and 8 Load/Store Units (Figure 1.c). It is then able to execute several operations in parallel, e.g. 32 integer instructions or 8 memory loads. In addition, the GPU contains resources to host thousands of active threads so that context switching is extremely fast.

When a kernel is offloaded to a GPU, each block of threads (e.g. 1024 threads in the example of Figure 3) is mapped to one SM. This mapping is performed by the hardware, following a heuristic that tries to maximize the use of the SMs [21].

C. Execution model

At the lowest level, GPUs implement the Single Instruction Multiple Threads (SIMT) execution model, which can be seen as a mix between simultaneous multithreading and the SIMD[3] model. A block of threads is organized into fixed pools of 32 threads called *warps*[4]. All the threads inside a warp are executed in lockstep: in a given clock cycle, they all execute the same instruction. This reduces the complexity of the instruction fetch and decoding logic since these operations are shared between the 32 threads of a warp.

Warps are the smallest schedulable entities in a GPU. An SMP contains an instruction buffer for each active warp, which stores the next instructions to be executed by the warp. At each execution cycle, a hardware warp scheduler is responsible for electing a warp for execution among those that are ready, following a given scheduling policy [19]. A warp is ready for execution when all data dependencies have been resolved (which is checked using a scoreboard, as explained in Section II-D), when its next instruction is available in its instruction buffer and when the required functional units are available. The instruction to be executed by the elected warp is sent to a dispatch unit that pushes the instruction to the required functional units. Each SMP has two dispatch units: one for memory operations (which target LSUs) and one for the other instructions. The full processing of an instruction is depicted in Figure 2.

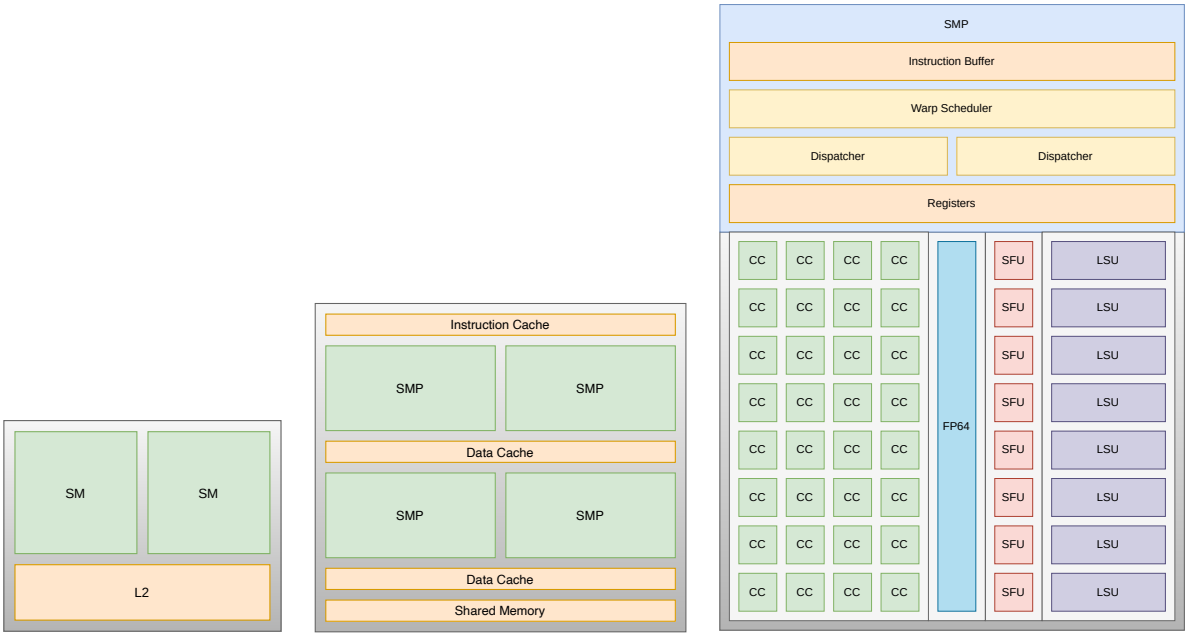
As long as all threads inside a warp agree on the control flow (i.e. they follow the same direction at conditional branches), the SIMT execution scheme is straightforward. However, when threads within the same warp disagree on whether or not to take a branch, both paths are executed *in sequence* one after the other, with only one part of the threads being active on each path. This phenomenon is known as *thread divergence* [8].We explain in Section III-A how thread divergence is handled in Pascal GPUs.

D. Scoreboard update and scheduling instructions

Recent NVIDIA GPUs handle pipeline hazards using a software programmed scoreboard: in a Pascal GPU program, the compiler inserts so-called *scheduling instructions* [11]

[3]Single Instruction Multiple Data

[4]The number of threads in a warp is not the same in all GPUs depending on the vendors. In NVIDIA GPUs, warps are always composed of 32 threads.



(a) GPU (b) SM (c) SMP

Fig. 1. Architecture of a Pascal GPU

between each group of three successive instructions. These are responsible for updating the scoreboard with information needed to enforce data dependencies (minimum instruction latencies, setup of local fences).

E. Memory hierarchy

All the SMs share a *global memory* that can be used by their threads to communicate between SMs, and a constant memory which stores constants. The global memory is also used to communicate between the CPU and the GPU through the use of a hardware *copy engine* that transfers data and instructions between the CPU and GPU memory spaces. Threads can also access a private *local memory*, which is in practice an area of the global memory. The global, constant and local memories are accessed through a cache hierarchy composed of an L2 cache, shared by all the SMs, and L1 caches, local to each SM. The memory hierarchy of the Pascal GPU is displayed in Figure 4.

In addition, each SM features a so-called *shared memory*. In the GPUs terminology, shared memory refers to a fast memory local to each SM, that is shared by threads that belong to the same block. NVIDIA reports a 100x lower latency for shared memory accesses compared to uncached global memory accesses, making it a key factor in the acceleration of kernel execution. It is implemented as an interleaved multi-banked SRAM with 32 banks storing 32-bit words. In the ideal case, threads of a given warp access either words from different banks or the same word from a given bank. The hardware is optimized to serve such requests with minimal latency and maximal throughput by grouping them into a single transaction. However, whenever two or more threads from the same warp try to access different addresses in the same bank at the same time, this results in a conflict. The bank then serves each request in sequence as part of a separated transaction, and all the threads in the warp wait until all requests have been served before executing the next instruction. This obviously degrades performance.

III. Reverse engineering the Pascal GPU execution

A. Thread divergence

Thread divergence occurs when threads within a warp do not all follow the same direction after a conditional branch. For example, in the code in Figure 3, threads execute either *line 15* or *line 17* depending on their own evaluation of the condition on *line 14*. As mentioned earlier, the SIMT execution model does not allow threads that belong to the same warp to execute different instructions. As a result, each path (i.e. *line 15* or *line 17*) is executed one after the other one, and each thread is active along one of these paths only.

According to [2], thread divergence is handled using a hardware mask mechanism that temporarily deactivates the threads that must not execute one path. In order to deal with nested conditional branches, a stack (called SIMT stack in the remainder of the paper) holds the activation masks, along with some additional information. In order to better understand the behavior of the SIMT stack, we read a NVIDIA patent dedicated to this mechanism [20]. In this section we present a model of the divergence handling mechanisms of the Pascal GPU, based on this patent and on experiments that we conducted in order to verify and clarify the behaviour of the GPU on conditional branches.

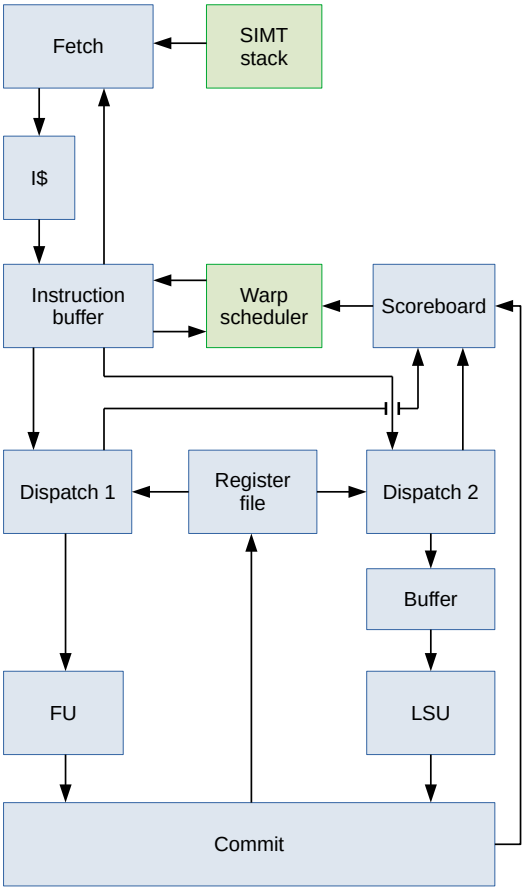


Fig. 2. Detailed view of the modeled elements for one SMP in the PasTiS simulator. In blue, the elements related to the non-functional semantics of instructions. In green, the elements related to the warp-level semantics.

```
1   #define BLK 1024
2   void fun(int *a, int n){
3       int *d_a;
4       cudaMalloc((void **)&d_a,n*BLK*sizeof(int));
5       cudaMemcpy(d_a, a, n*BLK*sizeof(int),
6                  cudaMemcpyHostToDevice);
7       kern<<<n,BLK>>>(d_a, n);
8       cudaMemcpy(a, d_a, n*BLK*sizeof(int),
9                  cudaMemcpyDeviceToHost);
10      cudaFree(d_a);
11  }
12  __global__ void kern(int *t, int n){
13      int tid=blockIdx.x*blockDim.x+threadIdx.x;
14      if (t[tid] < 0)
15          t[tid] = 0;
16      else
17          t[tid] *= 2;
18  }
19
```

Fig. 3. Example Cuda program

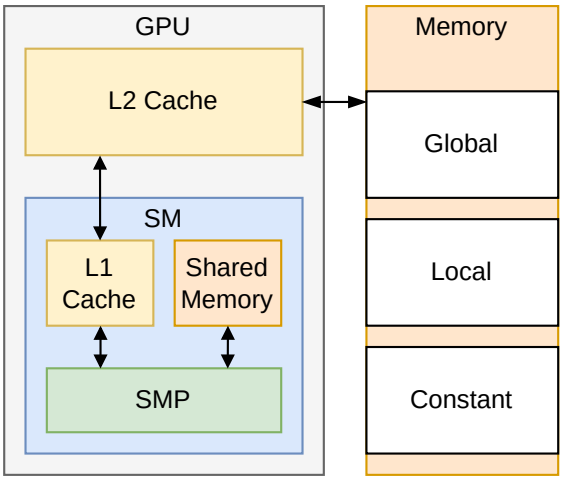


Fig. 4. Memory hierarchy

When a kernel is launched, a stack is allocated for each corresponding warp. In the stacks, a 32-bit mask is stored along with the address of the next instruction to execute (next program counter or *npc*) and of the address of the instruction at which the threads must wait for reconvergence (reconvergence program counter or *rpc*). Each stack is initialized with an entry composed of: a mask in which all the threads in the warp are active, the start address of the program as *npc* and the last address in memory as *rpc* (note that it does nos need to be a valid instruction address). The left part of Figure 5 shows the state of the SIMT stack for a warp executing the kernel of Figure 3: the next instruction to be executed is at address `0x8`, the end of the program is at address `0xFFFF` and all the threads of the warp are currently active. The GPU handles divergence and reconvergence through the use of dedicated instructions which are automatically inserted in the code by the compiler. In practice these instructions are responsible for modifying the SIMT stack of their warp. The `SSY` and `PBK` instructions prepare the stack for a possible divergence; they contain the reconvergence address. When `SSY @addr` or `PBK @addr` is executed, the top entry of the stack is popped. A new entry is pushed to handle the execution after reconvergence: the *npc* is the reconvergence address (`@addr`), and the *rpc* and the mask are copied from the popped entry. A second new entry is then pushed to handle the potentially diverging portion of code: the *npc* is the actual next instruction (the *npc* of the popped entry + 8, since instructions are encoded on 64 bits), the *rpc* is the reconvergence address (`@addr`) and the mask is the same as in the popped entry. In the example of Figure 5, we consider that the instruction at address `0x8` is a `SSY` or `PBK` instruction that prepares the stack for a future conditional branch (corresponding to the *if-else* construct in the code of Figure 3), with a reconvergence of the control flow at address `0x70`. The stack is updated (in the right part) with two entries. The bottom entry will be used after reconvergence: the *npc* is the address of reconvergence of the control flow (`0x70`). The top entry is used to let all active threads execute the next instruction (`0x10`) naturally. Note that these instructions do not create divergence but instead prepare the stack for a possible upcoming divergence, which is why the mask remains unchanged at this point. In our experiments, we encountered

the SSY instruction before branch instructions that correspond to either an IF or a SWITCH construct, and the PBK instruction before branch instructions corresponding to a loop, although the official documentation from NVIDIA indicates that PBK instructions are not supported by Pascal (and next-generation) GPUs.

Actual divergence happens when a branch (BRA @addr) instruction is executed conditionally by only a subset of the threads of a warp. When this happens, the top entry of the corresponding stack is popped. Two new entries are then pushed:

- the first pushed entry concerns the threads which do not take the branch: the *rpc* is the same as in the popped entry, the *npc* corresponds to the next instruction in the code (i.e. the current *npc* + 8) and the mask activates only the threads that do not take the branch (the ones for which the condition is false);
- the second entry has the target address of the branch as *npc* and the same *rpc* as the popped entry; its mask activates only the threads that take the branch (the ones for which the condition is true).

As a consequence, the GPU first executes the threads that take the branch, until they reach a reconvergence instruction which is added by the compiler: SYNC of BRK (depending on whether a SSY or a PBK was executed before). This process is illustrated in Figure 6. In this example, the threads reach a BRA 0x48 instruction at address 0x10. We assume that threads 0 and 1 of the considered warp execute the *else* part of the code because the elements they access in table t are positive, while the rest of the threads execute the *if* part. As a consequence, only threads 0 and 1 take the branch instruction while all other threads in the warp continue in sequence. As displayed in the right part of the figure, the top entry is replaced by two new entries: one for threads 0 and 1 (activation mask 0xC0000000) and one for the rest of the threads (mask 0x3FFFFFFF). To the best of our knowledge, no documentation explicitly describes the order in which the entries are pushed on the stack when divergence occurs (and thus the corresponding execution order of the branches). In our experiments, the first threads to execute are always the ones taking the branch.

The reconvergence instructions pop the top entry from the stack, as illustrated in Figure 7. In the example, threads 0 and 1 reach a SYNC (or BRK) instruction: their entry is popped from the stack. The GPU then resumes the execution with the group of threads active in the mask of the new entry at the top of the stack: the threads that do not take the branch. When they reach a SYNC (or BRK) instruction, their corresponding entry is popped from the stack: the reconvergence is done and the execution flow resumes at the reconvergence address (which is the *npc* of the entry at the top of the stack at this point).

B. Shared memory accesses

We needed to characterize the timing behavior of the shared memory in order to implement it in the simulator. The major difficulty was to derive the number of transactions generated by the hardware. It depends on (i) the memory addresses accessed by threads in a warp, and (ii) the size of the transferred data. To determine this number, we wrote a microbenchmark in which we control the size of the transferred data and the target memory addresses, so as to tune the number of conflicts in the memory banks. We then executed the benchmark on the Jetson TX2 board and observed the number of transactions using the NVIDIA nvprof profiling tool.

Before:

0x8	0xFFFF	0xFFFFFFFF
npc	*rpc*	*mask*

After:

0x10	0x70	0xFFFFFFFF
0x70	0xFFFF	0xFFFFFFFF
npc	*rpc*	*mask*

Fig. 5. SIMT stack when executing SSY/PBK 0x70

Before:

0x10	0x70	0xFFFFFFFF
0x70	0xFFFF	0xFFFFFFFF
npc	*rpc*	*mask*

After:

0x48	0x70	0xC0000000
0x18	0x70	0x3FFFFFFF
0x70	0xFFFF	0xFFFFFFFF
npc	*rpc*	*mask*

Fig. 6. SIMT stack when executing BRA 0x48 by the first 2 threads

Before:

0x48	0x70	0xC0000000
0x18	0x70	0x3FFFFFFF
0x70	0xFFFF	0xFFFFFFFF
npc	*rpc*	*mask*

After:

0x18	0x70	0x3FFFFFFF
0x70	0xFFFF	0xFFFFFFFF
npc	*rpc*	*mask*

Fig. 7. SIMT stack when executing SYNC/BRK by threads 0 and 1

	size of accesses (# bits)		
mask	32	64	128
000000FF	1	2	4
0000FFFF	1	2	4
00FFFFFF	1	2	4
FFFFFFFF	1	2	4

TABLE I
NUMBER OF TRANSACTIONS FOR CONSECUTIVE ACCESSES

We report in Table I the number of observed transactions when threads in a warp make accesses to consecutive areas of the memory (e.g. thread 0 accesses a 32-bit word at address 0, thread 1 accesses a 32-bit word at address 4, etc.). Table I also displays the activation mask that we use in order to control which threads are active. For 32-bit accesses, a single

#	mask	size of accesses (# bits) 32	64	128
1	00000001	1	2	4
2	00000003	2	3	5
3	00000007	3	4	6
4	0000000F	4	5	7
5	0000001F	5	6	8
6	0000003F	6	7	9
7	0000007F	7	8	10
8	000000FF	8	9	11
9	000001FF	9	10	11
10	000003FF	10	11	12
11	000007FF	11	12	13
12	00000FFF	12	13	14
13	00001FFF	13	14	15
14	00003FFF	14	15	16
15	00007FFF	15	16	17
16	0000FFFF	16	17	18
17	0001FFFF	17	17	18
18	0003FFFF	18	18	19
19	0007FFFF	19	19	20
20	000FFFFF	20	20	21
21	001FFFFF	21	21	22
22	003FFFFF	22	22	23
23	007FFFFF	23	23	24
24	00FFFFFF	24	24	25
25	01FFFFFF	25	25	25
26	03FFFFFF	26	26	26
27	07FFFFFF	27	27	27
28	0FFFFFFF	28	28	28
29	1FFFFFFF	29	29	29
30	3FFFFFFF	30	30	30
31	7FFFFFFF	31	31	31
32	FFFFFFFF	32	32	32

TABLE II
TRANSACTIONS ISSUED WHEN ALL THREADS ARE IN CONFLICT

transaction is issued by the hardware, regardless of the number of active threads: each memory bank composing the shared memory is accessed by one and only one thread, and the transaction accounts for up to 32 non-conflicting accesses to 32-bit words. For 64-bit accesses, two transactions are issued, regardless of the number of active threads. In this setting, when more than 16 threads are active, at least one bank is accessed by two threads that request different words: this conflict is handled by issuing two transactions. However, when less than 17 threads are active and are consecutive, the accesses made by the active threads are non-conflicting, and we initially expected to measure a single transaction, as in the case of 32-bit accesses. For 128-bit accesses, four transactions are issued, regardless of the number of threads. Once again, we expected this behavior when more than 24 threads are active, but based on the number of conflicting accesses in the banks, we expected a single transaction when less than 9 (consecutive) threads are active, two transactions when between 9 and 15 consecutive threads are active, and three transactions when between 16 and 23 consecutive threads are active.

We designed a first experiment in which all active threads access different words in the same bank at the same time, and in which we vary the number of active threads from 1 to 32 and the size of the access from 32 to 128 bits. For each

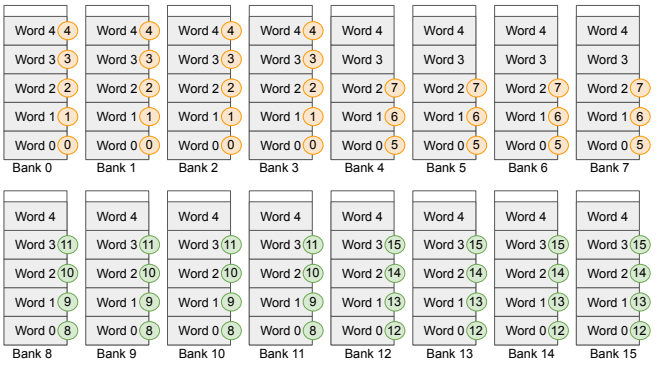


Fig. 8. Conflicts between shared memory accesses

combination of active threads and access size, we measured the number of issued transactions using `nvprof`. We discovered that threads of a warp that perform memory accesses are grouped in one pool of 32 threads when the accesses performed are 32-bit wide, in two pools of 16 (threads 0 to 15 and 16 to 31) when the accesses are 64-bit wide and in four pools of 8 (threads 0 to 7, 8 to 15, 16 to 23 and 24 to 31) when the accesses are 128-bit wide. The results of the experiment are displayed in table II. By default, one transaction is issued for each pool of threads when a warp executes a memory access. As reported in Table II we measured that even when all the threads in a given pool are inactive, a transaction is still issued for the pool. Since transactions are issued pool-wise, only threads from the same pool can generate conflicting accesses to the memory banks. We ran some additional experiments to confirm that hypothesis. In these experiments we forced conflicts to happen between threads from the same pool on two separate banks. The banks accessed by threads of different pools are not the same, in order to evaluate if the hardware tries to coalesce accesses from different groups in transactions. Figure 8 displays an example of such an experiment. For space reasons, we only display the first 5 words of the first 16 banks of the shared memory. In the example, threads 0 to 15 perform 128-bit accesses to the memory: each thread thus performs accesses to 4 consecutive banks. Since accesses are of size 128 bits, threads 0 to 7 are in a pool (shown with orange disks) while threads 8 to 15 are in another pool (shown with green disks). Threads 16 to 31 are not displayed, once again for space reasons, and without loss of generality we can assume that they are not active in the example. The threads are programmed so that in the orange pool, threads 0 to 4 are in conflict on banks 0 to 3 and threads 5 to 7 are in conflict on banks 4 to 7, and in the green pool threads 8 to 11 are in conflict and threads 12 to 15 also. Running the corresponding program on the GPU generates a total of 9 transactions.

After multiple experiments in which we varied the configuration of the conflicts, we observed that transactions were generated in order to deal with the conflicts of each pool separately, and that for each pool, the number of generated transactions corresponds to one plus the maximum number

of conflicts on any bank between threads of the pool. In the example of Figure 8, the maximum number of conflicts on any bank is 4 for the orange pool, and 3 for the green pool. In our hypothesis, the GPU is thus expected to generate $(4+1)+(3+1)=9$ transactions which is exactly what we measured. Following our observations, we derived the following formula to compute the number of transactions issued by the hardware:

$$nTrans = \sum_{i=1}^{nPools} (1 + \max_{j \in [|0,31|]} (conflict(pool_i, bank_j)))$$

where $nTrans$ is the number of issued transactions, $nPools$ is the number of thread pools (1, 2 or 4) and $conflict(p, b)$ is the number of conflicting accesses by threads of pool p on bank b.

We used the same benchmarks and measured the number of cycles for the accesses instead of the number of transactions. We could derive the following formula:

$$dur = 22 + base + 2 \times \sum_{i=1}^{nPools} (\max_{j \in [|0,31|]} (conflict(pool_i, bank_j)))$$

where dur is the duration in cycles of the execution of the memory instruction and $base$ is a constant duration which depends on the size of the accesses: 1 cycle for 32-bit accesses, 8 cycles for 64-bit accesses and 16 cycles for 128-bit accesses.

IV. The PasTiS simulator

A. Global architecture of the simulator

The simulation of a CUDA program with PasTiS mimicks a real execution shared between a CPU host and a GPU accelerator target, as illustrated in Figure 9. The CPU parts of the program (e.g. the `fun` function), are compiled and executed natively by the CPU of the machine that hosts the simulation, and the call to a GPU kernel is replaced by a call to PasTiS (which is developed in C language). The simulator code and the CPU part of the program are linked together and share the same memory space. As a result, the simulator can read and write in a memory zone accessible to the host program[5]. Before starting the cycle-level simulation, the `simulate` function stores the parameters in the constant memory and initializes special registers for each thread (e.g. with its identifiers, such as `blockIdx.x`), as expected by the GPU assembly code.

PasTiS is organized around three main modules that work together to enable a cycle-accurate simulation of GPU code:

1) an instruction set simulator: it is responsible for decoding and functionally simulating GPU binary code. This means maintaining the contents of registers and of the memory all along the execution, by emulating the semantics of the program's instructions. The instruction set simulator is agnostic of the timing aspects of the execution.

[5]PasTiS is intended to simulate and determine the duration of the execution on the GPU, but not of the operations of the copy engine.

```
1  #define BLK 1024
2  void fun(int *a, int n){
3      parameter_t par[2];
4      par[0].kind = PTR; par[0].val.ptr = a;
5      par[1].kind = INT; par[1].val.i = n;
6      simulate("kern.cubin", n, BLK, par);
7      // instead of kern<<<n,BLK>>>(a,n);
8  }
9
```

Fig. 9. Modification of the example in Figure 3 to invoke the simulator

2) a model of the warp-level execution semantics: it simulates warp selection and scheduling policies, as well as control flow and intra-warp thread divergence.
3) a model of the timing aspects of the execution: it maintains the state of the architectural components (e.g. contents of cache memories, occupation of functional units, contents of the scoreboard, etc.) and determines instruction latencies in such a way that it is able to determine the full state of the GPU after each simulated clock cycle. Thanks to this module, PasTiS is able to derive the total execution time of a kernel.

When the simulator is launched, elements modeling the memory, caches, scoreboard, registers and SIMT stacks are allocated. Then, a loop simulates the execution cycle by cycle: for each SMP, if a warp is ready for execution, the warp scheduler simulator selects the warp to execute and the instruction set simulator decodes and emulates the corresponding instruction. The latency of the instruction is computed depending on its nature, the state of the caches, and the potential memory access conflicts. Finally the state of the hardware simulator is updated according to the effect of the current instruction.

B. Instruction set simulator

The instruction set simulator is in charge of decoding the binary code of the program and emulating its execution from a functional perspective. For this purpose, we rely on GLISS [23], a tool that accepts the description of an ISA (its encoding and its semantics) and generates a library of functions that can be invoked to decode an instruction or to evaluate its impact onto the processor's state (registers and memory contents). We have described a large part of the Pascal ISA in the GLISS format.

The Pascal GPUs use the same ISA as the Maxwell generation. Unfortunately, the low-level (SASS)[6] language is not documented by NVIDIA: only the instruction names are provided, but not their encoding nor their exact semantics. We thus had to conduct reverse engineering in order to determine the missing information. The work reported in [11] was a valuable starting point for this task, which is really tedious and which we have limited to the subset of instructions that appear in our benchmark applications (in particular matrix products and

[6]PTX is a common ISA for all Nvidia GPUs, while SASS is a micro-architecture specific ISA. SASS code is generated by JIT or AOT compilation of PTX sources

a feed-forward fully-connected neural network). The Maxas tool[7] was very useful to uncover the entire encoding scheme for some of the instructions. To understand the semantics of the instructions, we carefully analyzed the mapping between source and binary code, and we could clarify uncertainties based on NVIDIA-related forums.

The method for deriving the encoding of instructions is the following:

1) compile a GPU kernel to a .cubin binary file using the nvcc compiler
2) disassemble the .cubin file using nvdisasm
3) select an instruction to investigate
4) modify the disassembled program to generate multiple instances of the instruction with varying operands and options
5) re-assemble the program using maxas
6) disassemble again, and compare the binary encodings of the instructions

The second step to building an instruction set simulator is to describe the semantics of the instructions. Once again some reverse engineering had to be performed for this step. To do so we once again start by writing a kernel (e.g. a matrix product), which we compile and disassemble. We then proceed as follows, in a systematic manner:

1) figure out the most probable semantics of the instruction using its name
2) run the program on the GPU and on the simulator, verify if the results are equal
3) if the results are not the same, execute the program step by step in the simulator and on the GPU with cuda-gdb
4) find the first instruction for which the state (register and memory values) in the simulated and executed program differ
5) collect results and source operands on both sides
6) find the purpose of this instruction in the executed program
7) implement this semantics in the simulator
8) try again from 2)

C. State of the implementation and current limitations

As of today, our implementation of PasTiS supports integer and floating point arithmetic instructions, memory accesses to all memories of the GPU, conditional branching and instructions for thread divergence/reconvergence handling, scheduling instructions, memory fences and synchronization barriers. Functionally speaking, the floating point arithmetic is handled natively by the machine that performs the simulation, so results may differ from the ones obtained on the Jetson TX2.

We currently simulate a direct-mapped cache hierarchy. In our experiments, this policy is enough to get precise results in terms of cache misses, but we expect this precision to fall as we experiment with ever larger kernels. As a consequence we are currently working on experimenting other policies, based on existing results [18].

[7]https://github.com/NervanaSystems/maxas

Another limitation is the warp scheduler whose policy remains unknown at this point. For now, a warp is simulated until its completion or until it reaches a synchronization barrier, at which point another warp is elected for simulation. Using this policy the simulation is functionally correct but does not faithfully represent the memory latency hiding mechanisms that the real GPU implements. As a consequence, our simulation timing results are close to the measurements on the actual GPU as long as at most one warp is active on each SMP, but the overhead is growing as we add more warps to the simulation. Determining a realistic warp scheduling policy is the next step in our research. To do so, we can rely on existing work [21] and on our simulator: once again we will work by trial and error, by first assuming a policy, implement it and validate it or not following the results of the simulations.

V. PERFORMANCE EVALUATION

We conducted two kinds of experiments to assess the functional correctness and the timing precision of PasTiS. Since our objective is the simulation (and in a second step the static analysis) of neural network applications, we started by experimenting PasTis on integer matrix multiplications, which represent the major building block of neural network computations. We then implemented a simple feed-forward neural network for handwritten numbers recognition which we trained on the MNIST dataset [1] and tested our simulator on the inference function of this network.

The integer matrix multiplication benchmark uses the shared memory as is usually the case in ”real-life” kernels: the program starts by a copy of the contents of the matrices to multiply from the global memory to the shared memory, then each thread performs the computation of one element of the result matrix and stores it directly in the global memory. We started the evaluation with a multiplication of two 4×4 integer matrices, which is handled by 16 threads of one warp (the 16 other threads are not active for this computation). The results are depicted in Figure 10. First of all the results are functionally correct: we obtain the same result matrix in the simulation as in the actual execution on the Jetson TX2. Second, the number of reported transactions to the shared memory is the same in the simulation and in the execution. Finally, the number of cycles reported in the simulation is 9.7% higher than in the measurements performed on the execution. We expected an overhead since we do not know all the implementation details of the GPU. In this experiment, we believe that the overhead is linked to the L2 cache of the GPU: in the Jetson TX2, the copy of the input data (the two input matrices) from the CPU to the GPU is performed through the L2 cache of the GPU. As a consequence, the cache is warm when the execution of the kernel starts on the GPU. Unfortunately, PasTiS does not simulate the copy of the inputs from the CPU to the GPU: it considers that the inputs are already in the global memory and thus starts the simulation of the kernel with a cold cache. We performed the same experiment on 8×8 (resp. 11×11 matrices), in order to test the simulator with 2 (resp. 4) active warps in parallel.

size : 4*4, block : 4*4 threads			
Measures	GPU	Simulator	Overhead
shared memory reads	8	8	0
shared memory writes	2	2	0
cycles	1131	1241	9.7%

Fig. 10. Matrix multiplication on one warp

size : 8*8, block : 8*8 threads			
Measures	GPU	Simulator	Overhead
shared memory reads	32	32	0
shared memory writes	4	4	0
cycles	1381	1491	7.9%

Fig. 11. Matrix multiplication on two warps

size : 11*11, block : 11*11 threads			
Measures	GPU	Simulator	Overhead
shared memory read	88	88	0
shared memory write	8	8	0
cycles	1580	1677	6.1%

Fig. 12. Matrix multiplication on four warps

Since our target GPU is composed of 4 SMP each capable of running one warp in parallel, this is the maximum number of warps that can be executed/simulated without influence from the warp schedulers policy. Once again, the results of the simulation are functionally correct, and we obtain an overhead of 7.9% (resp. 6.1%) in terms of execution cycles (reported in Figures 11 and 12 respectively). We consider that these results validate our model of the shared memory.

Our neural network benchmark is a simple feed-forward network composed of 3 dense layers which performs the classification of input images representing handwritten digits (taken from the MNIST dataset). The first (resp. second, third) layer is composed of 784 (resp. 128, 64) weights and 128 (resp. 64, 10) biases, and the activation function is ReLU for all three layers. Each layer is implemented as a separate kernel which performs a matrix multiplication analogous to the one of our first experiment, and then calls an activation function. The network was written in C/CUDA, trained on the Jetson TX2, and the trained parameters were exported so they could be used for inference in the simulation as well.

In the experiments we first measured the difference between the simulated and executed floating point operations and its influence on the results of the network. As expected, we measured a slight difference in the computed probabilities (the result layer). However the difference only appears after the 14th decimal digit, and does not affect the classification results: the classification precision of the network is 98.4375% both on the Jetson TX2 and on PasTiS. Unfortunately, this benchmark heavily relies on the 64-bit floating point unit and on some hardware prefetch optimizations whose behaviour is not correctly modeled yet in PasTiS. As a result the cycles count of the simulation sees a more than 100% overhead compared to the actual measurement. We are currently working on correcting this issue.

VI. Related work

With the ongoing adoption of GPUs in embedded and time-critical systems, the real-time community has started to work on the characterization of the timing aspects of GPUs. Since NVIDIA is currently the leader in the market of GPGPUs, and in particular for the embedded systems segment, most of the research has focused on understanding and taming the NVIDIA GPUs. The main problems with these GPUs lie in their closed nature: NVIDIA does not provide a complete documentation of its GPUs, and many aspects that are relevant to their timing behavior is undocumented. As a result, most of the scientific production yet has been dedicated to reverse-engineering various aspects of the execution of kernels on NVIDIA GPUs with three major drawbacks. First, it is extremely tedious and time-consuming. Second, we approach the GPUs as black boxes and rely on experiments and measurements to characterize their behavior, with no guarantee that a particular case was not missed. Finally, as NVIDIA builds new GPU families, there is no guarantee that a new-generation GPU will still behave in the same way than the previous ones: the reverse engineering experiments must be conducted again to validate the hypothesis again. The same is true for the CUDA API and drivers, as well as for the instruction set architecture which are also closed source.

At the higher level of description, the community has so-far covered the behavior of the stream queues which control the interface between the CPUs and the GPU [3], [25], as well as the timing aspects of the memory copies between the CPU and the GPU memory spaces [7]. The memory hierarchy and scheduler hierarchy (at the block and warp mapping levels) has also been documented for particular GPU families [16], [18], [21], although no reverse engineering has yet been performed to unravel the exact warp scheduling policy.

A noticeable exception is [22], in which the authors weigh the pros and cons of using AMD GPUs for research in the real-time domain. The authors report that the general organization of AMD GPUs does not differ from the NVIDIA ones, with mainly small dimensioning differences (number of SMs, number of threads per warp), and in the terminology. One important difference is the absence of integrated GPUs (iGPUs: SoCs in which the GPU and CPUs share the same memory) in the AMD offer, and a more flexible choice in the mapping of the thread blocks in the AMD GPUs. According to the authors, the main point in favor of AMD is that they decided to make their software stack (in particular their drivers and APIs) open source and to vastly document it. However the authors report that the vast amount of documentation is not centralized, making it tedious to find information (the same as for NVIDIA GPUs, although for other reasons), and that the API and drivers code is not yet stable enough to make it completely worthwhile to invest time in understanding its inner workings and in designing compatible real-time modules

since they may not be compatible to the next revision of the API (the equivalent for NVIDIA GPUs to reverse engineering a particular aspect of a particular GPU which may behave differently in the next family of GPUs).

Other works are dedicated to defining WCET analysis methods for more abstract GPUs (usually a simulated GPU e.g. [2] with simplifying assumptions) [5], [6], [12]–[15]. The main limitation in these works is the simplifying assumptions made on the execution behavior of the GPUs: although the methods work for the target simulated GPU, they are either incomplete (focusing on a particular aspect) or cannot be applied to COTS GPUs which implement more complex hardware optimization strategies.

VII. Conclusion

This paper presents how we reversed-engineered some aspects of the NVIDIA Pascal GPU of the Jetson TX2 board, and how we implemented them in the PasTiS simulator that we are currently developing. PasTiS supports the actual NVIDIA SASS assembly language, which makes it capable of simulating kernels that are compiled for Pascal GPU targets. We first explained the hidden mechanisms responsible for handling thread divergence and reconvergence in the presence of conditional branch instructions. Then we uncovered how transactions to the shared memory are handled by the hardware. In PasTiS, we have modeled these mechanisms as well as a consequent part of the Maxwell/Pascal ISA. We were able to demonstrate the current accuracy of the simulator on benchmarks implementing matrix products with the use of shared memory. In our experiments, the simulator cycle count suffers is overestimated by less than 10% compared to the actual measurements on the board, and this overhead is reduced as the size of the matrices grows. This confirms that our model of the shared memory is correct.

As part of future work we intend to apply further our methodology to other architectural elements (in particular cache hierarchy, FP64 units and warp scheduler) in order to complete our simulator, until we are able to simulate neural networks with a high temporal precision. We will then release the simulator code as open source. Once the model is precise enough, we will use this knowledge to build static analysis models in order to derive the worst-case execution time of kernels running on the Jetson TX2 board.

References

[1] Mnist handwritten digits data set. https://deepai.org/dataset/mnist.

[2] T. M. Aamodt, W. W. L. Fung, and T. G. Rogers. General-purpose graphics processor architectures. In *Synthesis lectures on computer architectures, Morgan & Claypool publishers*, 2018.

[3] T. Amert, N. Otterness, M. Yang, J. H. Anderson, and F. D. Smith. Gpu scheduling on the nvidia tx2: Hidden details revealed. In *2017 IEEE Real-Time Systems Symposium (RTSS)*, 2017.

[4] C. Ballabriga, H. Cassé, C. Rochange, and P. Sainrat. OTAWA: An Open Toolbox for Adaptive WCET Analysis. In *8th International Workshop on Software Technologies for Embedded and Ubiquitous Systems (SEUS)*, 2010.

[5] K. Berezovskyi. Timing analysis of general-purpose graphics processing units for real-time systems: Models and analyses. In *PhD dissertation, University of Porto*, 2015.

[6] K. Berezovskyi, K. Bletsas, and B. Andersson. Makespan computation for gpu threads running on a single streaming multiprocessor. In *2012 24th Euromicro Conference on Real-Time Systems*, 2012.

[7] A. J. Calderón, L. Kosmidis, C. F. Nicolás, F. J. Cazorla, and P. Onaindia. Understanding and exploiting the internals of gpu resource allocation for critical systems. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8, 2019.

[8] B. Coutinho, D. Sampaio, F. M. Q. Pereira, and W. Meira Jr. Divergence analysis and optimizations. In *2011 International Conference on Parallel Architectures and Compilation Techniques*, 2011.

[9] R. I. Davis and L. Cucu-Grosjean. A survey of probabilistic timing analysis techniques for real-time systems. *Leibniz Transactions on Embedded Systems*, 6(1), May 2019.

[10] R. Wilhelm et al. The worst-case execution-time problem—overview of methods and survey of tools. *ACM Transactions on Embedded Computing Systems*, 7(3), 2008.

[11] A. B. Hayes, F. Hua, J. Huang, Y. Chen, and E. Z. Zhang. Decoding cuda binary. In *Proceedings of the 2019 IEEE/ACM International Symposium on Code Generation and Optimization*, 2019.

[12] V. Hirvisalo. On static timing analysis of gpu kernels. In *Workshop in WCET Analysis*, 2014.

[13] Y. Huangfu and W. Zhang. Static wcet analysis of gpus with predictable warp scheduling. In *2017 IEEE 20th International Symposium on Real-Time Distributed Computing (ISORC)*, 2017.

[14] Y. Huangfu and W. Zhang. Wcet analysis of the shared data cache in integrated cpu-gpu architectures. In *2017 IEEE High Performance Extreme Computing Conference (HPEC)*, 2017.

[15] Y. Huangfu and W. Zhang. Wcet analysis of gpu l1 data caches. In *2018 IEEE High Performance extreme Computing Conference (HPEC)*, 2018.

[16] Z. Jia, M. Maggioni, B. Staiger, and D. P. Scarpazza. Dissecting the nvidia volta gpu architecture via microbenchmarking. *arXiv*, apr 2018.

[17] Y.-T.S. Li and S. Malik. Performance analysis of embedded software using implicit path enumeration. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 16(12), 1997.

[18] X. Mei and X. Chu. Dissecting gpu memory hierarchy through microbenchmarking. *IEEE Transactions on Parallel and Distributed Systems*, 28(1):72–86, 2017.

[19] Nvidia. Across thread out of order instruction dispatch in a multithreaded microprocessor. https://patentimages.storage.googleapis.com/ab/e4/d4/487e7e837f2ade/US7676657.pdf.

[20] Nvidia. Execution of divergent threads using a convergence barrier. https://patentimages.storage.googleapis.com/42/1d/77/18beae47a1fc64/US20160019066A1.pdf.

[21] I. S. Olmedo, N. Capodieci, J. L. Martinez, A. Marongiu, and M. Bertogna. Dissecting the cuda scheduling hierarchy: a performance and predictability perspective. In *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2020.

[22] N. Otterness and J. H. Anderson. AMD GPUs as an Alternative to NVIDIA for Supporting Real-Time Workloads. In *32nd Euromicro Conference on Real-Time Systems (ECRTS 2020)*, 2020.

[23] T. Ratsiambahotra, H. Casse, and P. Sainrat. A versatile generator of instruction set simulators and disassemblers. In *Int'l Symp. on Performance Evaluation of Computer Telecommunication Systems*, 2009.

[24] M. Schlickling and M. Pister. A Framework for Static Analysis of VHDL Code. In *7th International Workshop on Worst-Case Execution Time Analysis*, 2007.

[25] M. Yang, N. Otterness, T. Amert, J. Bakita, J. H. Anderson, and F. D. Smith. Avoiding Pitfalls when Using NVIDIA GPUs for Real-Time Tasks in Autonomous Systems. In *30th Euromicro Conference on Real-Time Systems (ECRTS 2018)*, 2018.

Session We.1.B

Model Driven Engineering I

Wednesday 1st June

11:30

–

Room Lauragais

Sizing a Drone Battery by coupling MBSE and MDAO

Ombeline Aïello
ISAE-SUPAERO, Université de Toulouse
Toulouse, France
Ombeline.Aiello@isae-supaero.fr

Olivier Poitou
ONERA
Toulouse, France
olivier.poitou@onera.fr

Jean-Charles Chaudemar
ISAE-SUPAERO, Université de Toulouse
Toulouse, France
jean-charles.chaudemar@isae-supaero.fr

Pierre de Saqui-Sannes
ISAE-SUPAERO, Université de Toulouse
Toulouse, France
pdss@isae-supaero.fr

Abstract

Drones have increasingly been used to assist Humans for rescue and surveillance missions. To do so, their design turns out to be a challenging issue, in particular when their autonomy is expected by sizing of batteries. Thus, solutions are to be sought to engineer a drone rigorously while specifying its main features. This paper introduces a work in progress aiming to bridge the gap between two engineering disciplines that have so far been developed separately: Model Based Systems Engineering (MBSE) and Multidisciplinary Design Analysis and Optimization (MDAO). Coupling of MBSE and MDAO is addressed in terms of language, tools, and methods.

1 Introduction

Over the past decades, drones have increasingly been used to assist Humans in hostile environments. Examples include high voltage electric line inspection in mountains, where hostility of the nature and the climate make missions life-critical. Of prime importance for these drones is to accomplish their inspection missions without power outage. Clearly, autonomy is a key issue for inspection drones.

Autonomy is a challenging issue for drones in general, and sizing of batteries raise complex design problems. Solutions may be sought in well-established engineering disciplines and associations of these disciplines.

This paper addresses the problem of drone batteries sizing by associating two families of engineering disciplines: Model Based Systems Engineering, or MBSE for short, and Multidisciplinary Design Analysis and Optimization, or MDAO for short. The benefits of using MBSE approaches for drone design have been acknowledged [ASV20]. By contrast, little work has been published [CS21] on joint use of MBSE and MDAO for revisiting the way drones are designed.

An early work [Aïe+21] by the authors of this paper has indicated that joint use of MBSE and MDAO opens promising avenues for sizing drone batteries and for drones design in general. Even if this work is still in progress, this paper goes beyond the concept proof presented in [Aïe+21], and formalizes the coupling of MBSE and MDAO in terms of language, tools and method.

The paper is organized as follows. Section 2 introduces MBSE and MDAO, and discusses the rationale behind the coupling of MBSE and MDAO. Section 3 surveys related work. Section 4 details the main contributions of the paper. Section 5 discusses a case study. Section 6 concludes the paper and outlines future work.

2 Rationale

2.1 MBSE

One of the current challenge is to develop innovative systems faster than ever, meeting ever higher expectations in terms of performance and safety. To take up this challenge, new methodologies are studied. One of the widely studied and promising approach to manage the complexity of systems is Model-Based Systems Engineering (MBSE) [Zha+21].

MBSE aims to facilitate the understanding of a system made up of parts interacting together. MBSE replaces document-centric approaches by model-centric ones. It improves communications between people involved in systems development. It also improves the understanding of the system under design, and reinforces testing and verification of the system throughout its life cycle [ZMT18].

The main asset of MBSE is its power of abstraction and its graphical representation. MBSE models may be classified into physical, geometric and mathematical models depending on their degree of abstraction [CS21]. These three different levels of abstraction ease the management of systems complexity, and make them understandable [CS21]. MBSE is increasingly used for the development of complex systems, for instance in [PF13] MBSE approach is used to design submarine subsystems. The authors of [PF13] propose to start modelling from the mission of the submarine to its components specifications. The interest of using

MBSE to design a submarine is that it contains more than 40 subsystems and a large number of functions. The subsystems of a submarine are highly integrated and cover a wide range of functionalities such as the navigation ones and the combat ones [PF13]. Another example in [Rim+] is to use MBSE approach to design a rover autonomous guidance, navigation, and control (GNC) and its collaborative drone. The authors of [Rim+] justify the choice of using MBSE because it allows to understand the general behavior of a complex system. According to [PF13], MBSE is comonly used in various industries such as aerospace and defence.

However some limitations are highlighted in the literature. In [CS21] the authors point out that MBSE still suffers from a lack of acceptance in terms of scientific and technical fields. Further, [Zha+21] highlights that it remains difficult to examine the design of the system under development from the conceptual design phase using domain-specific simulation. In addition models abstract the reality and address a simplified representation of the system. It becomes therefore almost impossible to include all the characteristics of the system into one model and a set of models may be needed. Moreover, existing MBSE methodologies are criticized for being too focused on high-level modeling which leads to a lack of precision in the design of the system [Zha+21]. On top of that, to achieve their objectives, each domain involved into the system development prefers to use its own methodology which does not allow to ensure consistency between models [Zha+21].

To ensure the consistency between domain-specific models, the authors of [Zha+21] propose to build an integrative system model which is linked to other domain-specific models. In this way, it is possible to describe the system with the accurate level expected. With regard to the methodology proposed in this paper, MBSE approach is also used to act as a guideline for the development of the system to which the MDAO will be integrated. The MBSE language used in this paper as well as in [Zha+21] is the Systems Modeling Language (SysML). SysML is a standard of the Object Management Group (OMG). It is also a multi-purpose language that allows to analyze, design, verify and validate a wide range of systems. Another benefits of using SysML is that it is not tool dependent and can therefore be supported by a wide range of MBSE tools.

[Par+21] reports that MBSE tools are currently used to describe the system baseline. [Par+21] mentioned also that only few papers use MBSE tools to look for design alternatives and to explore decision tradespace which forced engineers to evaluate the different alternatives at each life cycle stage. On the one hand, in which systems performance models is concerned, [Par+21] emphasizes the fact that MBSE tools would offer a way to include varying fidelity models and multiresolution models. On the other hand, in which definition and description of systems alternatives are concerned, [Par+21] points out that continuous design parameters are required to identify tradespace and perform Set-Based Design. Set-Based Design is a methodology allowing to take into account numerous design options and that eliminates poorer design choices throughout the development of the system [Sha+21]. The idea developed in this paper is to integrate MDAO approach into the MBSE one to precisely size a drone battery. The MDAO offers the possibility to create low and high fidelity models and to run different kind of analysis. The MDAO approach can also perform design space exploration, as well as trade-off analysis taking into account several design parameters distributed into different disciplines. For this reason, coupling MBSE and MDAO seems to be a good solution to reduce the limitations of MBSE tools above-mentioned.

2.2 MDAO

Unlike MBSE, MDAO is not intended to describe the system but rather to analyze a model in order to demonstrate the properties of the system using dynamical models based on mathematical theories [CS21]. MDAO is commonly used for designing engineering systems, such as airplanes and drones for instance, because their design requires to involve several disciplines [Joa12]. MDAO applies numerical optimization using algorithms that minimize or maximize an objective function. The problem allowing to find the minimum or maximum of the objective function can be constrained or not [Joa12]. For instance, a structural design optimization may consist in varying several parameters such as thickness in order to optimize the weight of the piece while taking the stress constraints into account.

A MDAO model describes in details an aspect of the system in a formal language. In [CS21], several strengths of MDAO are identified. Firstly, MDAO uses high computing performance for simulating and analyzing the models. But MDAO is also able to integrate high fidelity simulation tools, to deal with a large number of design variables and constraints, to have a wide range of efficient optimizers, and finally, to take into account model uncertainties [CS21]. Since its emergence, MDAO has demonstrated its effectiveness. For instance, MDAO can be used to design aircraft with minimum environment impact [Joa12], to design a CubeSat, and to achieve structural topology optimization [Gra+19].

OpenMDAO This paper presents and uses MDAO models built with OpenMDAO. OpenMDAO is an open-source framework that aims to solve design problems involving coupled numerical models [Gra+19]. OpenMDAO simulates complex systems such as satellites, drones, and aircrafts taking into account the interactions between all disciplines involved in the problem. Further, design variables of the problem are optimized simultaneously by paying attention to the interdisciplinary coupling. Several trade-offs are performed while running the analysis.

XDSM One difficulty in using MDAO to solve a problem is to choose the suitable architecture, that is to say, the strategy for organizing the analysis in order

to achieve an optimal design. [LM12] identified a lack of standard representation with respect to multidisciplinary design and optimization architectures. The solution proposed in [LM12] is to create a diagram offering a visual representation of the MDO architecture based on a common set of mathematical notations. This diagram is called an extended design structure matrix (XDSM) and its strength is to enhance the links between the elements of the diagram and the underlying mathematics.

WhatsOpt In this paper, we use the application WhatsOpt [LDL19] developed by French laboratory ONERA to create the XDSM of the MDAO models built. Its environment facilitates collaborative work providing a shared vision of the model under construction. In addition, WhatsOpt interfaces tools to conduct studies and generates skeleton code to facilitate analysis implementation [LDL19].

2.3 Why coupling MBSE with MDAO?

According to [CS21] MDAO model is restricted to a single aspect of the system. That is to say, all the components contained in the system are not identified by MDAO as well as their interconnection in terms of functions and data flow. Therefore, it is interesting to associate MDAO with MBSE given that designing system's models is a strength of MBSE. Moreover, it takes time to engineers to write and validate MDAO models because of the formal languages that support MDAO, whereas all these information are available into MBSE description. Conversely, the MBSE sometimes lacks precision in terms of analytical values. This is the reason why, this paper proposes coupling MBSE and MDAO in order to compensate the lack of one with the assets of the other. The following section of this paper focuses on the desire to improve requirements. To do so, we propose to start writing requirements with data provided by the stakeholders and the limitations of current technologies for sizing batteries. Then, to use this first set of data as input parameters of MDAO models. At the end of the MDAO analysis we expect to be able to improve several requirements thanks to MDAO results. Using more precise requirements from the conceptual phase will allow one to design the system in a safer and faster way since the design space to explore will already be reduced.

2.4 MBSE-MDAO coupling proposal

Figure 1 illustrates the main points covered by the MBSE-MDAO coupling method proposed in this paper. The coupling is split up into three distinct branches: Approach, Type and Tool. The *Approach* branch provides the two approaches concerned by the coupling and conveys the idea that MDAO results are used to populate MBSE description. The relation "Triggers" shows that it is the MBSE that initiate the request of using MDAO. Thanks to the coupling with MDAO, data contained in the MBSE description, and more precisely into the SysML diagrams modelling the system under development, are more accurate. The green arrow illustrates that point through the connection named "Refines" which associates MDAO to MBSE. The second branch entitled Type expresses that the coupling occurs at two different level: firstly there is a methodological coupling and secondly, there is a language coupling. A future goal of the coupling is to automate the communication and the data transmission between the two approaches. Finally, the third branch presents the main tools used to realize the coupling describe in this paper.

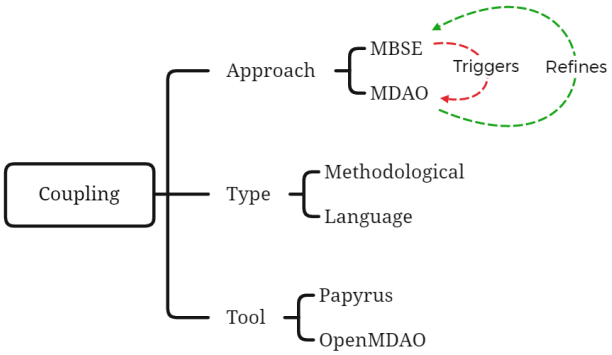


Figure 1: MBSE - MDAO coupling mind map

3 Related work

As mentioned in section 1, it is sometimes required to complete the MBSE approach by another one such as MDAO [CN21]. Or, by another tool, such as Matlab/Simulink (for instance, [Zha+21] which enables simulation of models developed through an MBSE approach [ZMT18]). According to [Nik+16], simulation is one of the preferred method to explore the performance of SysML. In the literature [Nik+15], several approaches aim to use simulation results to verify SysML models, that is to say, to ensure that requirements specified are satisfied by the system. Simulations are used to make trade-off analysis and take design decision.

In order to optimize the development of drones, it is indeed required to make decisions from the conceptual design phases. Modeling and simulations can help designers in making trade-off to select the most appropriate design solutions before building the drones [ZMT18]. According to [ZMT18] modeling and simulation are two different things: modeling helps understanding phenomena whereas simulation allows one to experience a model in different environments. One of the advantages of simulation is to better understand the behavior of a system simulating its associated model in different situations. In this paper, the SysML language is selected as the main modeling language. Simulating SysML models is a major step to validate models in terms of performance. Numerous researches are currently ongoing on the subject of generating simulation code from SysML models. For instance, [Nik+15] presents several approaches for automating

SysML models simulation process. The general process consists in exporting SysML models in XMI format and to transform them into another model readable by the simulation platform. Additional information such as constraints are transmitted to the simulation platform through profiles introduced in the SysML model. Thanks to the variety of diagrams offered by SysML, for instance use case diagrams and block definition diagrams, structure and behavior of the system can be defined. However, the behavior of the system is still described directly into the simulation platform using their libraries. The solution proposed by [Nik+15] to carry out the transformation from SysML models to simulation models is to define a meta-model describing simulation entities. Model transformation languages such as ATL and QVT are presented as good candidates to perform SysML to simulation models transformation.

In [Ker+13] the authors agree with the idea to develop models in SysML and to simulate these models with domain-specific tools. The approach developed in [Ker+13] consists in creating an integrated SysML model containing relevant data. From this integrated model, it is proposed to generate domain specific models. In this way, all domain-specific models are not developed from scratch and they depend on the same model (the integrated one). Such a method contributes to achieve consistency between all models. The low cost required to implement such a method, and the ease to set up simulation and test that allow one to obtain quick results, are its major advantages [Ker+13]. In terms of coupling, the SysML physical architecture of the system is directly simulated in tools such as Modelica where a code is generated from the logical architecture and tested on a real Programmable Logic Controller (PLC) [Ker+13].

The approach proposed in this paper is different from the ones presented above. Even if any model transformation is performed in the method proposed in this paper, two different approaches working on different tools are coupled. To be efficient, the coupling between MBSE and MDAO requires to set up communication between both of them. Using XML to exchange the data contained into SysML diagrams as it is done in [Nik+16] can be a solution for the MBSE-MDAO information sharing.

The objective of our work is, in the first instance, to refine the requirements in order to give as much details as possible about the expected features of the system. Writing good requirements is a crucial step of the systems engineering approach, it allows the definition of the system under development [BCN21].

The project Agile 4.0 also addresses MBSE-MDAO coupling and considers it as an enabler to accelerate the development of innovative products. The authors of [CN21] identified in the literature that deploying an MDAO collaborative design process could help in reducing the system development time [CN21]. The methodology proposed in [CN21] covers the entire life cycle of the system, starting from the stakeholders identification to the validation process as it is done in a systems engineering approach. The conceptual framework in [CN21] use systems engineering for upstream architecting phases such as system identification and system specification whereas they use MDAO processes for downstream product design phases. In our work, we propose to initiate the coupling between MBSE and MDAO from the requirement writing in order to add as details and accurate values characterizing the system as early as possible in the life cycle.

To write good requirements, that is to say requirements that are complete, consistent, understandable, unambiguous and traceable, [BCN21] proposes to judge their quality using measurable indicators. The authors of [BCN21] further present several viewpoints as modeling guidelines with stakeholders, needs, and requirements in mind. As far as requirements writing is concerned, 5 fields to be filled in are expected : Text, ID, Type, Author, and Version. The Text field is supposed to be filled in by following a requirement pattern corresponding to a standard sentence, according to the type of requirement to be defined. For example performance requirement, design constraints requirement or even environmental requirements.

Finally, certification of the drone developed is also part of the research spectrum of the Agile 4.0 project. A certification-driven process is presented in [Tor+21]. Since drones are embedding an ever increasing number of functionalities and become more complex, integrating the issue of certification as early as possible in the development of such system will facilitate its final certification.

4 Contributions

The work presented in this paper aims to size a drone battery in order to complete a given mission. To do this, an MBSE model and an MDAO analysis are done. The expected result is to have precise value of drone and battery characteristics that are required to go further in the design of both of them. Obtaining reliable values from the conceptual design of the system is an asset to ensure a good development of the battery and it reduces the design space to explore from the beginning of the project.

4.1 Process proposed

The process proposed in this paper is made up of 5 steps, as depicted by Figure 2.

The first step consists in writing requirements in a SysML requirement diagram using existing SysML stereotypes and the 3 additional stereotypes proposed in this paper (Figure 3). In this step the scalar that needs to be sized with the MDAO is identified with the Boolean identifier named $MDAOobjectiveFunction$. If the Boolean is true, the variable corresponds to the objective function. During the second step, the MDAO model that will optimize the objective function is built: this allows one to identify the design variables and parameters required to complete and run the MDAO model. The third step corresponds to the search for

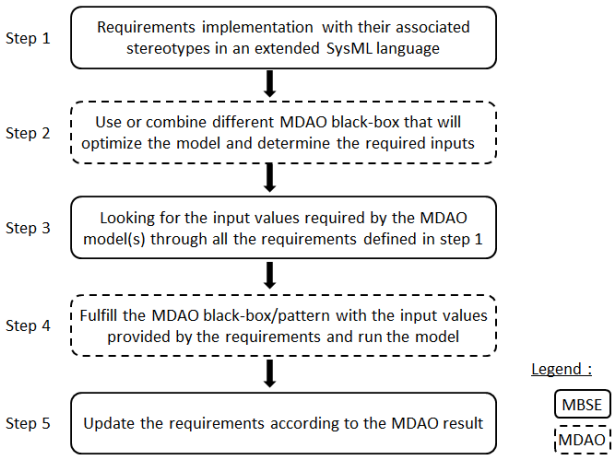


Figure 2: Requirement improvement process

these variables values in the requirements diagram using the stereotypes created in this paper (Figure 3). Then the input values are exchanged from the requirement diagram to the MDAO model. Finally, after running the MDAO analysis and optimization, outputs are added to the requirements diagram, which allows one to update it and improve its accuracy. An example is run in section 5.

4.2 MBSE modeling

To create a strong coupling between MBSE and MDAO, several changes are proposed and presented in the remainder of this paper. First, to highlight the new links between MDAO and MBSE models, new stereotypes were created in the Papyrus tool [DLG09]: MDAOproperties, MDAOinput, and MDAOoutput Figure 3.

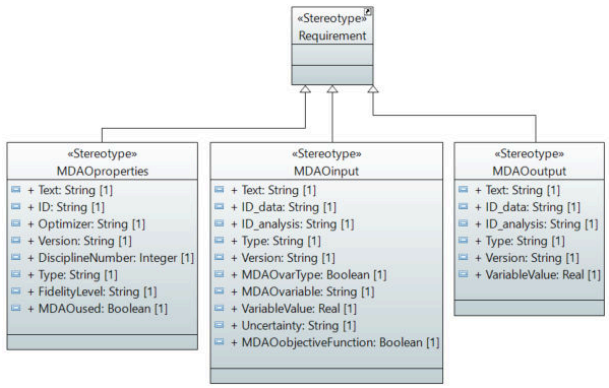


Figure 3: New stereotypes proposed for the requirement diagram

`MDAOproperties` contains all information related to the MDAO model definition used to improve associated requirements. Requirements with the stereotype ≪MDAOproperties≫ contains for example the solver used for the optimization in the Optimizer attribute, the number of disciplines that are part of the MDAO model, and the fidelity of the model in use. The Boolean MDAOused indicates if the MDAO described in the requirement with the stereotype ≪MDAOproperties≫ is used to improve another requirement with another stereotype or not.

`MDAOinput` contains data that can be used as input values by the MDAO model. These data are classified into 3 categories: "design variable", "parameter", and "objective function". The objective function is characterized by a true value of the Boolean MDAOobjectiveFunction. The difference between design variables and parameters is that a value of a design variable can be modified during the MDAO optimization process when parameters remain fixed values. The Boolean MDAOvarType returns the type of the variable. If the Boolean MDAOvarType is true, the variable is a design parameter; otherwise, it is a parameter. MDAOvariable indicates the name of the corresponding variable used in the MDAO model.

`MDAOoutput` allows one to integrate the outputs of the MDAO into the SysML requirement diagram. For this reason two relations are created: MDAOupdate and MDAOaddition. MDAOupdate represents the link between the original requirement and the requirement containing the variable value obtained with the MDAO. MDAOaddition represents the link between a new requirement given by the MDAO optimization and a requirement already present in the initial requirement diagram.

Several attributes created in this paper mean the same thing in different stereotypes. They are defined as follows. 'id_data' represents the identifier of the requirement which contained a data used or obtained by the MDAO. 'id_analysis' represents the MDAO solver that modifies or uses the value of the considered requirement. 'version' indicates the version of the requirement. If the requirement is modified, the version number will be incremented. 'type' indicates if the requirement is functional or non-functional. Finally, 'text' is a text field that supports controlled natural language (not addressed in this paper) and defines the variable provided into the requirement.

The 3 stereotypes ≪MDAOproperties≫, ≪MDAOinput≫ and ≪MDAOoutput≫ cover the needs of the drone battery sizing developed in section 5. Future work will allow one to complete and generalize them so that they can be used for other applications than battery sizing.

4.3 MDAO modeling

At this point, requirements are considered to be written, but they may not be accurate enough to restrict the possibilities of design of the future drone battery. For instance, this applies to the requirement named MaxWeight presented in the example section 5.

Another concern is to ensure that the drone will have enough autonomy to complete its given mission. For this reason, an MDAO model is built and run. It aims to help improving part of the requirements written in a SysML requirements diagram, and to verify if the drone is able to complete its mission with a battery developed that meets these requirements. The objective function

of the MDAO model used in this paper is the state of charge of the battery, deduced from the Bréguet range equation (1) for fully-electric UAV that is optimized by the MDAO.

$$R_{elec} = \frac{c_b}{g}\frac{C_L}{C_D}\frac{W_{batt}}{W_T O}\eta_E SC\eta_m\eta_p \quad (1)$$

Where :

- c_b is the battery specific energy
- g is the gravitational constant
- $\frac{C_L}{C_D}$ is the lift to drag ratio
- W_{batt}, W_{TO} are the drone battery weight and the takeoff weight
- $\eta_E SC$, η_m, η_p are the Electronic Speed Controller (ESC), motor and propulsive efficiency

The MDAO is used to find a trade-off between all design variables of the problem that provides the maximum range of the drone, which is obtained as an output of the optimization.

In this paper, Bayesian optimization [Moc75] is performed and, the objective function is approximated by a Gaussian Process [Wil+20]. This approach suits to find a global optimum of an objective function expensive to evaluate [Fra18]. The optimizer used is named Super Efficient Global Optimization with Mixture Of Experts (SEGOMOE) [Pri+20]. SEGOMOE optimizer is a good candidate to solve global optimization problems subject to nonlinear constraints and involving numerous design variables [Bar+19].

The MDAO model is presented in Figure 4. The technical vocabulary defining a wing and used in the MDAO model is presented in Figure 5 and Figure 6.

The MDAO model is made up of seven disciplines.

- Atmos, that takes as inputs parameters the flight altitude of the drone and the Mach number. It uses these two parameters to compute several parameters such as temperature, speed of sound, and computes and returns Reynolds number, air density and True Air Speed (TAS) as outputs parameters. These values are available for the other disciplines and thus used in other computations.

- Geometry, this discipline creates the mesh of the wing and the tail of the drone. It takes as inputs data the root chord, the span and the twist of the wing, respectively the tail of the drone. This mesh is required for others disciplines such as Aerodynamics and Structure for instance.

- W0, this discipline adds the mass of the battery with the mass of the fuselage and of the payload. The output corresponding to the mass of the drone without wing and tail is required in the discipline Structure to determine the mass of the two entities wing and tail.

- AeroStructure: this fourth block corresponds to a second MDAO model presented in Figure 7. It is made up of three disciplines (Structure, Aerodynamics, and LoadsTransfer), and another block named DispTransferGroup.
  - Structure: discipline that takes as input the dihedral, the sweep, the twist, the taper ratio, the mesh and the thickness of the wing, respectively the tail of the drone, and W0. It outputs some parameters such as the displacement of the wing and the tail, the mass of the wing and the tail, and a new mesh taking into account the real shape of the wing and the tail (dihedral, twist, sweep...). It also verifies if the wing, respectively, the tail resist the forces exerted when the drone makes a turn.
  - DispTransferGroup: it is a third MDAO model shown in Figure 8 and made up of two disciplines: ComputeTransformationMatrix and DisplacementTransfer. ComputeTransformationMatrix takes as inputs the displacement of the wing and the tail from the discipline Structure. With this, it builds a matrix used by the discipline DisplacementTransfer to define the deformed mesh of wing and tail of the drone.
  - Aerodynamics: this discipline computes lift and drag cruise coefficients as well as forces applied on the wing and the tail. It uses variables previously computed by other disciplines like the Reynolds number, the air density, the TAS, the mesh and the deformed mesh of the wing and the tail. But also design variable found in the requirement diagram like the dihedral (wing and tail), the sweep (wing and tail), the taper ratio (wing and tail), the twist (wing and tail), the angle of attack (AOA) and the Mach.
  - LoadsTransfer: this discipline takes as input only variable defined by the other disciplines of the MDAO model. The deformed mesh of the wing and tail are transmitted by the discipline DisplacementTransfer, whereas forces applied on the tail and the wing come from the discipline Aerodynamics. The outputs of LoadsTransfer are the loads of the wing and the tail.

  Once the analysis of the AeroStructure MDAO model is completed, outputs values are sent to the first MDAO model Figure 4 and used for the computation of the state of charge of the drone battery.

- Masses, takes as inputs the output of W0 and the mass of the wing and tail computed by the discipline Structure to determine the Take-Off Gross Weight (TOGW) of the drone.

- BreguetElec, contains the equation (1) to optimize. It takes as input parameters such as the

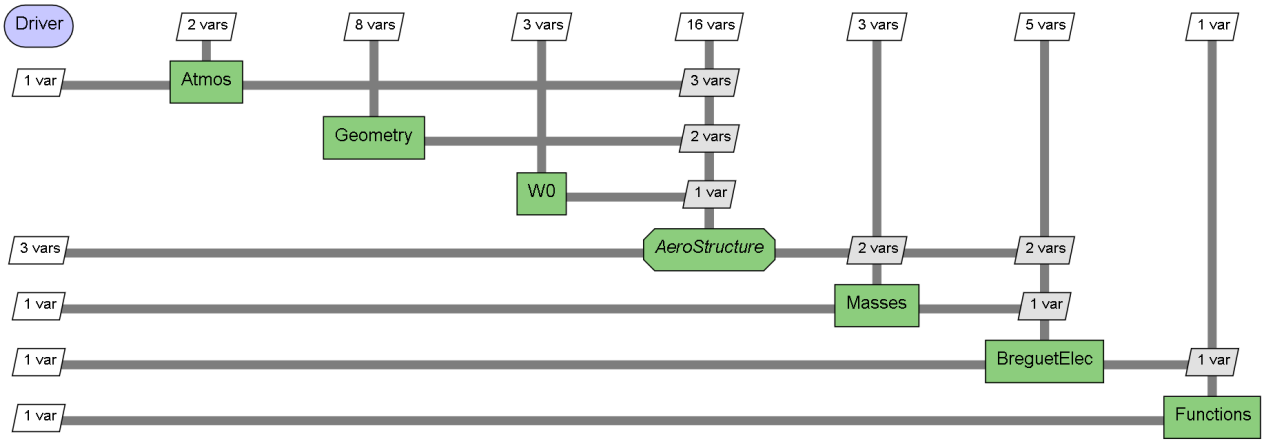


Figure 4: XDSM maximizing the drone range.

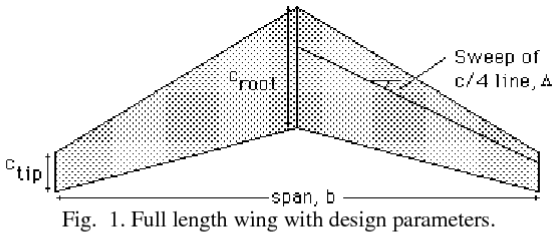


Fig. 1. Full length wing with design parameters.

- S =span
- AR =aspect ratio (b^2/s)
- sweep
- thickness
- Λ =tapper ratio (C_{tip}/C_{root})
- Θ= wing twist angle.

Figure 5: Technical vocabulary of a wing.

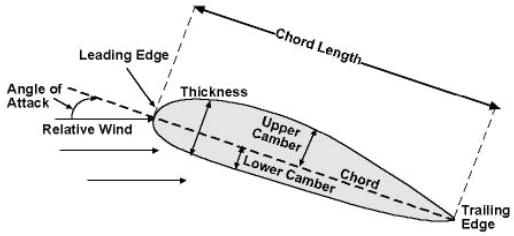


Figure 6: Technical vocabulary of a NACA profile.

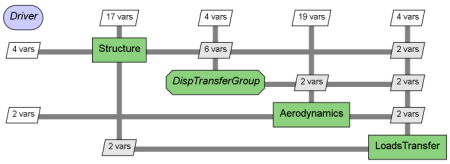


Figure 7: AeroStructure XDSM

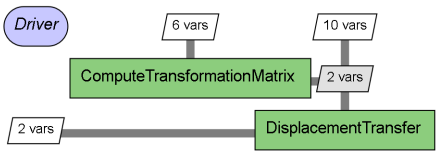


Figure 8: DispTransferGroup XDSM.

mass of the battery, the battery specific energy and the efficiency of the ESC, of the motor and of the propeller. It also uses lift and drag cruise coefficients obtained as output variables of the Aerodynamics discipline, and the TOGW computed by the discipline Masses. This discipline returns as output value the maximum range that the drone can travel.

- Functions, which is the objective function. It provides the optimized state of charge of the drone battery by taking as input variable the maximum range computed by BreguetElec and the target range (corresponding variable: target_range) to travel.

5 Case study

5.1 Mission Description

The mission considered in this paper is a high-voltage power line surveillance mission executed by a drone. In this mission, the drone should fly over a high-voltage power line recording data using embedded sensors.

Traditionally, inspection of high-voltage power lines is achieved by Humans or using an helicopter [Liu+19]. Dangers for human beings along with inspection costs

lead to investigate new inspection solutions. Drones have many advantages in terms of low cost, and easier access to areas that remain difficult for Humans or helicopters. For safety reasons, high-voltage power lines are often located far from housing and crowded areas. They can be located in mountains or in forests for example, which increases difficulty for technicians [Liu+19].

A high-voltage power line mission may present different objectives. The advantage of using a drone for executing such a mission is that a drone can be customized depending on the tasks it will have to perform. For instance, a drone can evaluate the state of the wires using cameras, make repairs on the wires with a robotic arm, and acquire electrical measurements from wires (for instance, the temperature and the value of the current). Drones need to be designed for the equipment they carry, and according to the parameters related to their mission. For instance, one needs to maintain a minimum distance from the power line, and to adapt to weather conditions and relief.

In the power line surveillance mission chosen in this paper, the main criterion to be taken into account is the distance to travel. The drone should take-off, inspect the high-voltage power line, return to its starting point and land. In this paper a fixed wing drone is preferred to a multirotor one because of the former's greater flight autonomy.

In addition to having an impact on the design of the entire drone, the mission objectives, the onboard and the operational environments have an impact on the drone's battery. At the moment, the mission achievement depends on the flight autonomy, and thus, on the battery performances. It should be noted that developing models for battery discharge is a challenge in many ongoing researches [Cha+16].

5.2 Drone battery

Lithium polymer (LiPo) batteries are commonly used to power drones since they present several advantages. For instance, they provide from ten to thirty times the theoretical energy density supplied by lead batteries. Further, they are lighter than Nickel-Cadmium batteries. Other interesting characteristics that justify their use in drones include the low cost, the durability, and the high charge and discharge rates [Cha+16].

In [Cha+16] the idea of categorizing drones and creating a large energy consumption model for each class created is introduced. The objective of these models is to estimate the energy consumption of the battery, once the drone is associated to the mission it has to execute. This way, the mission can be better planned before it is executed. The objective of this paper is similar, that is to say, to find a way to predict the battery discharge based on the mission being performed. The difference in this paper is that a method coupling the MBSE and MDAO approaches is proposed and implemented to predict the battery discharge. This coupling method is detailed in section 4.

5.3 Drone battery modeling

According to the requirement TargetRange (Figure 9) the length of the high voltage power line is 20000 m that corresponds to the value called target_range in the MDAO model. The target value expected as an input value in the MDAO model is provided by the excerpt of the SysML requirement diagram Figure 9. The remaining state of charge of the battery at the end of the mission is computed with this target_range value and other design parameters of the drone and the battery. In the MDAO model used in this paper, the problem is constrained by four parameters that are the lift coefficient of the drone, the wing and the tail failure, and the TOGW. Considering the lift coefficient as a constraint allows one to ensure that the drone is able to fly. Sizing the drone thinking about wing failure and tail failure parameters ensures that neither the wing nor the tail will break during the flight. Finally, the maximum weight of the drone authorized is 8 Kgs (requirement MaxWeight Figure 9). For this reason the parameter max_mass is also used to constrain the problem Figure 10.

The MDAO model built in this paper takes 19 inputs parameters and design variable, as described in section 4. After 250 optimization iterations it returns 31 variables as output, as shown on Figure 11, Figure 13, and Figure 12.

The best result obtained after 250 iterations is the one presented in Figure 12 that corresponds to a drone of 7.87 Kgs (TOGW).

The discipline breguet_elec shows that this drone is able to travel 109,442 m thanks to its battery, Figure 12. Knowing that the high voltage power line surveillance mission required to fly 20,000 m, it is possible to ensure that the drone is able to complete the mission described at the beginning of section 5. After executing this surveillance mission, the discipline functions computes the remaining SOC of the drone that is of 81,7% Figure 12.

In view of the results obtained, particularly in terms of remaining autonomy, it is legitimate to think that the drone is oversized. However this gives us the possibility to rethink and reorganize the mission. For example, it is possible to embed more energy-consuming sensors into the drone that would allow to better inspect the high voltage power line. Another option is to extend the distance of high voltage power line to monitor during the surveillance mission. For instance, it could be interesting to inspect a nearby high voltage power line the same day. However, this drone won't be sufficient to inspect the longest high voltage power line in the world which measures 1700 Km.

In addition, the objective of this example was a proof of concept showing the feasibility of the coupling mechanism proposed in this paper. For this reason, not all variables have been optimized. In particular, the battery mass has been fixed and was not part of the optimized variables. In future works, more variables will be optimized, including the mass of the battery which is a major point in the dimensioning of many systems.

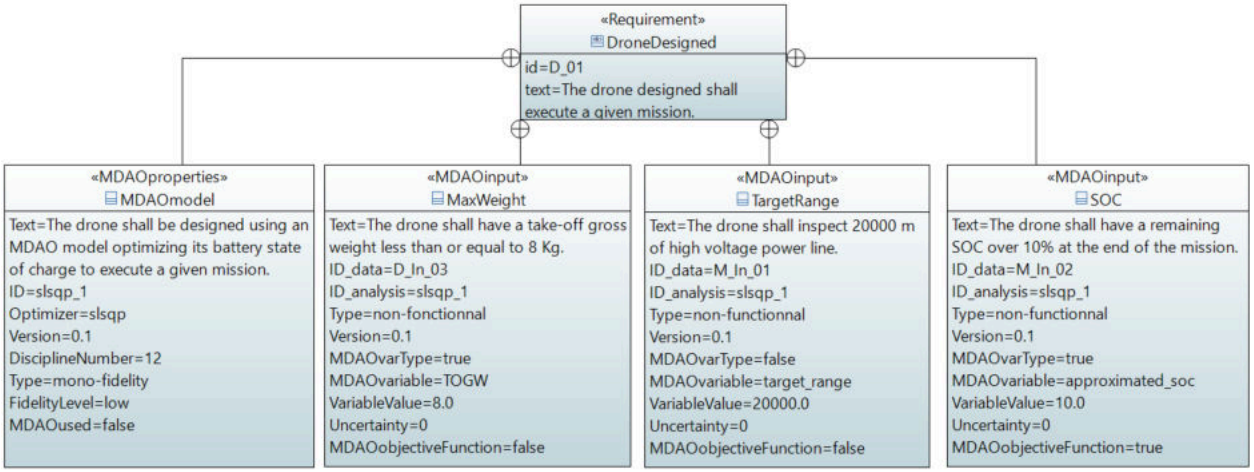


Figure 9: Excerpt of the initial requirements diagram.

```
40  [SEGO] Constraints:
41  CL_cruise (0)
42  wing_failure_cruise (1)
43  tail_failure_cruise (2)
44  max_mass (3)
```

Figure 10: Constraints of the MDAO problem.

```
70044  31 Explicit Output(s) in 'model'
70045
70046  varname                          val
70047  -------------------------------  --------------------
70048  atmos
70049    Reynolds                       [1477110.14567701]
70050    rho_air                        [1.28288361]
70051    TAS                            [19.94991158]
70052  geometry
70053    mesh_tail                      |33.441179704071445|
70054    mesh_wing                      |32.98522988588984|
70055  w0
70056    W0                             [6.]
```

Figure 11: Output variables of the disciplines atmos, geometry, and w0.

```
70086  masses
70087    TOGW                           [7.87133643]
70088  breguet_elec
70089    Range_max                      [109442.71278247]
70090  functions
70091    approximated_SOC               [0.81725599]
```

Figure 12: Output variables of the disciplines masses, breguet_elec, and functions.

The variables that characterize the drone are set in the block AeroStructure Figure 4 by the disciplines Structure, ComputeTransformationMatrix, DisplacementTransfer, Aerodynamics, and LoadsTransfer. The results are presented in Figure 13.

5.4 Results: requirement improvement

Finally, the analytical values obtained with the MDAO analysis (Figures 11, 13, 12) are used to update the initial requirements diagram Figure 9. The new stereotypes proposed in section 4 as well as the relations between requirements are applied in Figure 14, the requirement diagram obtained after inserting MDAO results inside.

```
70057  aero_structure
70058    structure
70059      disp_tail                    |15.099668870632314|
70060      disp_wing                    |14.696939311846727|
70061      fem_origin_tail              [0.35]
70062      fem_origin_wing              [0.35]
70063      mesh_tail_twisted            |33.450117129830765|
70064      mesh_wing_twisted            |32.99824747681123|
70065      nodes_tail                   |11.333583713997037|
70066      nodes_wing                   |10.810484330466467|
70067      struct_mass_tail_cruise      [0.24886297]
70068      struct_mass_wing_cruise      [1.62247346]
70069      tail_failure_cruise          [-0.98193365]
70070      wing_failure_cruise          [-0.84420774]
70071    disp_transfer_group
70072      compute_transformation_matrix
70073        transformation_matrix_tail |18.493242009045186|
70074        transformation_matrix_wing |18.000000685650097|
70075      displacement_transfer
70076        def_mesh_tail              |33.45011746224209|
70077        def_mesh_wing              |32.99820984283152|
70078    aerodynamics
70079      CD_cruise                    [0.03471341]
70080      CL_cruise                    [0.49999198]
70081      tail_sec_forces              |22.535731112734233|
70082      wing_sec_forces              |42.976823546451236|
70083    loads_transfer
70084      aero_loads_tail              |4.632074287590359|
70085      aero_loads_wing              |15.684387141358123|
```

Figure 13: Output variables of the blocks AeroStructure and DispTransferGroup.

The updated requirement diagram, more accurate than Figure 9, allows one to go to the next step of the drone battery design with a stronger background than if the design method had not used MDAO model.

6 Conclusions

From some years now, drones are even more present in the sky and execute a wide range of missions to ease Humans' work. There exists different kinds of drones. Some of them are employed to travel a long distance whereas others are smaller than a human hand

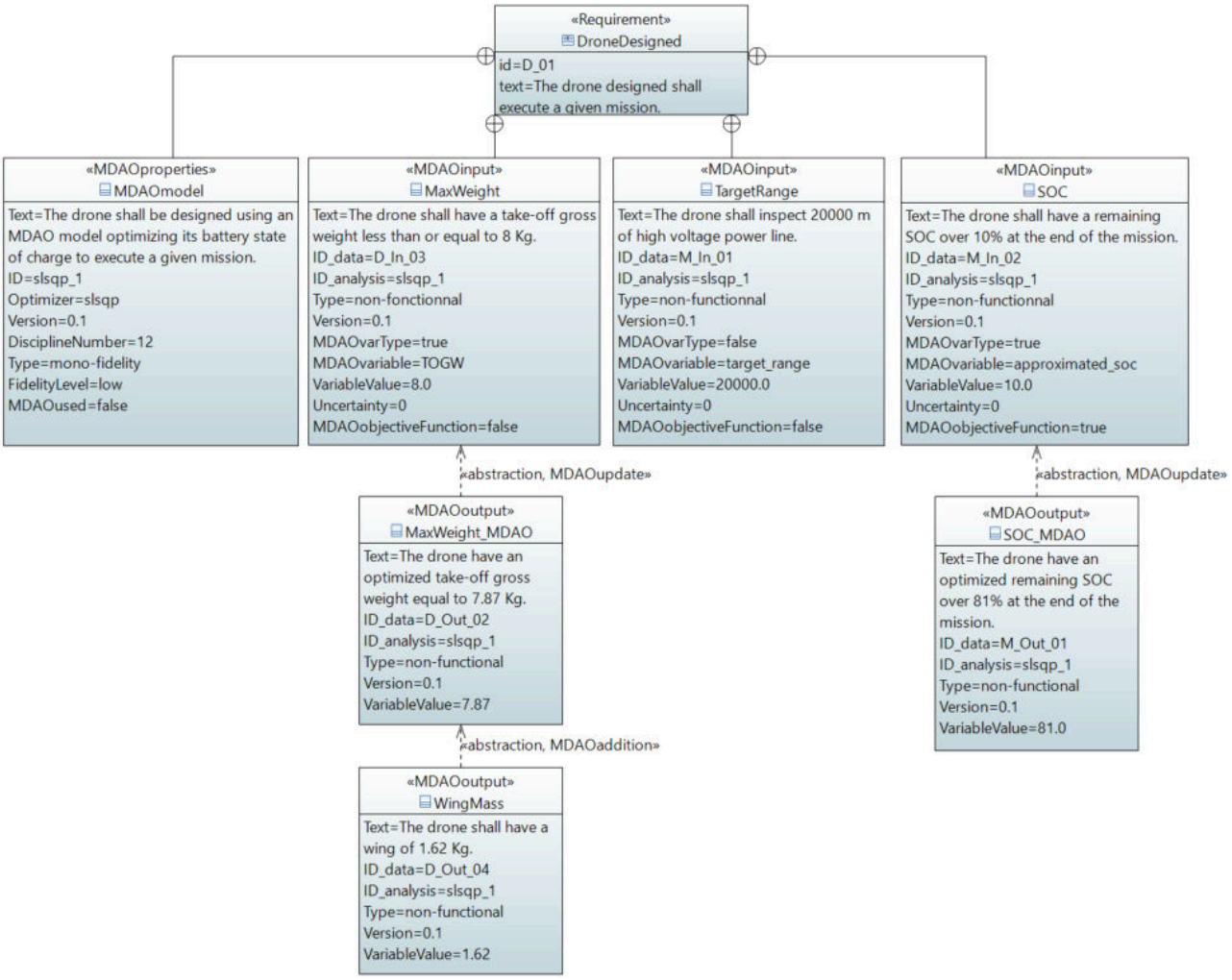


Figure 14: Updated excerpt of the requirement diagram.

[Gro+21]. The design of each drone depends on the mission it will have to execute. MBSE allows one to describe the system from at least a structural and behavioral point of view [CS21] and to manage the requirements that the system should satisfy. However some limitations in the MBSE approach can be pointed out [CS21].

This paper proposes to strengthen the MBSE approach by coupling it with the MDAO approach. The latter advantageously provides precise analytical values after analyzing and optimizing a model composed of several inter-related disciplines. In such a way, MBSE takes advantage of the mathematical computations achieved by the MDAO. On the other hand, the MDAO can use the MBSE as a data base to fill in its models. For instance, requirement diagrams may contain a lot of information expected as inputs by the MDAO. In this paper, different stereotypes are proposed to initiate the communication and exchange of data between MBSE and MDAO approaches. The consistency between all models (MBSE and MDAO) is preserved because the ones are modified according to the others. In other words, they evolve in a synchronous way.

Keeping in mind that the main objective of the MBSE-MDAO coupling is to accelerate the development process of the drone, future work will consist in automating the exchange of data between the requirement diagram and the MDAO model. A common language understandable by MBSE and MDAO will be proposed.

Formalizing the drone mission will be an important milestone of our project. More precise parameters required by the drone will be taken into account to satisfy the mission. This will contribute to an improved design of the drone.

Creating MDAO model patterns from the SysML requirement diagram is also an avenue to explore.

Finally, the coupling of MBSE and MDAO approaches will be extended beyond the requirement step in order to cover the entire lifecycle of the drone.

Acknowledgements

This work was supported by the Defense Innovation Agency (AID) of the French Ministry of Defense (research project CONCORDE N° 2019 65 0090004707501).

We would like to thank M. Rémy Charayron for his contribution to the realization of the MDAO models.

References

[Aïe+21] Ombeline Aïello et al. "Populating MBSE Models from MDAO Analysis". In: *7th IEEE International Symposium on Systems Engineering*. virtual, Viena, Austria, Sept. 2021.

[ASV20] Ludovic Apvrille, Pierre de Saqui-Sannes, and Rob A. Vingerhoeds. "An Educational Case Study of Using SysML and TTool for Unmanned Aerial Vehicles Design". In: *IEEE Journal on Miniaturization for Air and Space Systems* 1.2 (2020), p. 117..129.

[Bar+19] N. Bartoli et al. "Adaptive modeling strategy for constrained global optimization with application to aerodynamic wing design". en. In: *Aerospace Science and Technology* 90 (July 2019), pp. 85–102. ISSN: 12709638. DOI: 10.1016/j.ast.2019.03.041. URL: https://linkinghub.elsevier.com/retrieve/pii/S1270963818306011 (visited on 12/22/2021).

[BCN21] Luca Boggero, Pier Davide Ciampa, and Björn Nagel. "An MBSE Architectural Framework for the Agile Definition of System Stakeholders, Needs and Requirements". en. In: *AIAA AVIATION 2021 FORUM*. VIRTUAL EVENT: American Institute of Aeronautics and Astronautics, Aug. 2021. ISBN: 978-1-62410-610-1. DOI: 10.2514/6.2021-3076. URL: https://arc.aiaa.org/doi/10.2514/6.2021-3076 (visited on 11/02/2021).

[Cha+16] K. Chang et al. "LiPo battery energy studies for improved flight performance of unmanned aerial systems". In: *Volume 9837, Unmanned Systems Technology XVIII*. Ed. by Robert E. Karlsen et al. Baltimore, Maryland, United States, May 2016, 98370W. DOI: 10.1117/12.2223352. (Visited on 09/01/2021).

[CS21] Jean-Charles Chaudemar and Pierre de Saqui-Sannes. "MBSE and MDAO for Early Validation of Design Decisions: a Bibliography Survey". In: *5th annual IEEE International Systems Conference (SysCon 2021)*. 2021, pp. 1–8. DOI: 10.1109/SysCon48628.2021.9447140.

[CN21] Pier Davide Ciampa and Björn Nagel. "Accelerating the Development of Complex Systems in Aeronautics via MBSE and MDAO: a Roadmap to Agility". en. In: *AIAA AVIATION 2021 FORUM*. VIRTUAL EVENT: American Institute of Aeronautics and Astronautics, Aug. 2021. ISBN: 978-1-62410-610-1. DOI: 10.2514/6.2021-3056. URL: https://arc.aiaa.org/doi/10.2514/6.2021-3056 (visited on 11/02/2021).

[DLG09] Hubert Dubois, Fadoi Lakhal, and Sébastien Gérard. "The Papyrus Tool as an Eclipse UML2-modeling Environment for Requirements". In: *2009 Second International Workshop on Managing Requirements Knowledge*. Sept. 2009, pp. 85–88. DOI: 10.1109/MARK.2009.11.

[Fra18] Peter I. Frazier. "A Tutorial on Bayesian Optimization". In: *arXiv:1807.02811 [cs, math, stat]* (July 2018). arXiv: 1807.02811. URL: http://arxiv.org/abs/1807.02811 (visited on 12/11/2021).

[Gra+19] Justin S. Gray et al. "OpenMDAO: an open-source framework for multidisciplinary design, analysis, and optimization". en. In: *Structural and Multidisciplinary Optimization* 59.4 (Apr. 2019), pp. 1075–1104. ISSN: 1615-1488. DOI: 10.1007/s00158-019-02211-z. URL: https://doi.org/10.1007/s00158-019-02211-z (visited on 02/24/2021).

[Gro+21] Sébastien Grondel et al. "Towards the Use of Flapping Wing Nano Aerial Vehicles". In: *Modern Technologies Enabling Safe and Secure UAV Operation in Urban Airspace* (2021). Publisher: IOS Press, pp. 52–63. DOI: 10.3233/NICSP210006. URL: https://ebooks.iospress.nl/doi/10.3233/NICSP210006 (visited on 01/03/2022).

[Joa12] Joaquim R. R. A. Martins. *AE588 Multidisciplinary Design Optimization*. en. Mar. 2012.

[Ker+13] K. Kernschmidt et al. "Possibilities and challenges of an integrated development using a combined SysML-model and corresponding domain specific models". en. In: *IFAC Proceedings Volumes* 46.9 (2013), pp. 1465–1470. ISSN: 14746670. DOI: 10.3182/20130619-3-RU-3018.00391. URL: https://linkinghub.elsevier.com/retrieve/pii/S1474667016344998 (visited on 12/03/2021).

[LDL19] Rémi Lafage, Sebastien Defoort, and Thierry Lefebvre. "WhatsOpt: a web application for multidisciplinary design analysis and optimization". In: *AIAA Aviation 2019 Forum*. 2019, p. 2990. DOI: 10.2514/6.2019-2990. URL: https://arc.aiaa.org/doi/10.2514/6.2019-2990.

[LM12] Andrew Lambe and Joaquim Martins. "Extensions to the Design Structure Matrix for the Description of Multidisciplinary Design, Analysis, and Optimization Processes". In: *Structural and Multidisciplinary Optimization* (Aug. 2012). DOI: 10.1007/s00158-012-0763-y.

[Liu+19] Yao Liu et al. “Two-Layer Routing for High-Voltage Powerline Inspection by Cooperated Ground Vehicle and Drone”. en. In: *Energies* 12.7 (Apr. 2019), p. 1385. ISSN: 1996-1073. DOI: 10.3390/en12071385. URL: https://www.mdpi.com/1996-1073/12/7/1385 (visited on 09/01/2021).

[Moc75] J. Mockus. “On the Bayes Methods for Seeking the Extremal Point”. en. In: *IFAC Proceedings Volumes* 8.1 (Aug. 1975), pp. 428–431. ISSN: 14746670. DOI: 10.1016/S1474-6670(17)67769-3. URL: https://linkinghub.elsevier.com/retrieve/pii/S1474667017677693 (visited on 12/22/2021).

[Nik+15] Mara Nikolaidou et al. “Simulating SysML models: Overview and challenges”. In: *10th System of Systems Engineering Conference (SoSE)*. May 2015, pp. 328–333. DOI: 10.1109/SYSOSE.2015.7151961.

[Nik+16] Mara Nikolaidou et al. “Challenges in SysML Model Simulation”. In: *Advances in Computer Science: an International Journal* 5 (July 2016).

[Par+21] Gregory S. Parnell et al. “MBSE Enabled Trade-Off Analyses”. en. In: *INCOSE International Symposium* 31.1 (July 2021), pp. 1406–1416. ISSN: 2334-5837, 2334-5837. DOI: 10.1002/j.2334-5837.2021.00909.x. URL: https://onlinelibrary.wiley.com/doi/10.1002/j.2334-5837.2021.00909.x (visited on 11/24/2021).

[PF13] Paul Pearce and Sanford Friedenthal. “A Practical Approach for Modelling Submarine Subsystem Architecture in SysML”. en. In: *Engineering Conference* (2013), p. 14.

[Pri+20] Remy Priem et al. “An efficient application of Bayesian optimization to an industrial MDO framework for aircraft design.” en. In: *AIAA AVIATION 2020 FORUM*. VIRTUAL EVENT: American Institute of Aeronautics and Astronautics, June 2020. ISBN: 978-1-62410-598-2. DOI: 10.2514/6.2020-3152. URL: https://arc.aiaa.org/doi/10.2514/6.2020-3152 (visited on 12/22/2021).

[Rim+] Jasmine Rimani et al. “MBSE APPROACH APPLIED TO LUNAR SURFACE EXPLORATION ELEMENTS”. en. In: (), p. 4.

[Sha+21] Nicholas J. Shallcross et al. “A value of information methodology for multiobjective decisions in quantitative set-based design”. en. In: *Systems Engineering* 24.6 (2021), pp. 409–424. ISSN: 1520-6858. DOI: 10.1002/sys.21593. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/sys.21593 (visited on 12/21/2021).

[Tor+21] Francesco Torrigiani et al. “An MBSE Certification-Driven Design of a UAV MALE Configuration in the AGILE 4.0 Design Environment”. en. In: *AIAA AVIATION 2021 FORUM*. VIRTUAL EVENT: American Institute of Aeronautics and Astronautics, Aug. 2021. ISBN: 978-1-62410-610-1. DOI: 10.2514/6.2021-3080. URL: https://arc.aiaa.org/doi/10.2514/6.2021-3080 (visited on 11/02/2021).

[Wil+20] James Wilson et al. “Efficiently sampling functions from Gaussian process posteriors”. en. In: *Proceedings of the 37th International Conference on Machine Learning*. ISSN: 2640-3498. PMLR, Nov. 2020, pp. 10292–10302. URL: https://proceedings.mlr.press/v119/wilson20a.html (visited on 12/22/2021).

[ZMT18] Bernard Phillip Zeigler, Sarah Mittal, and Mamadou Kaba Traore. “MBSE with/out Simulation: State of the Art and Way Forward”. In: *Systems* 6.4 (2018).

[Zha+21] Yizhe Zhang et al. “Towards Holistic System Models Including Domain-Specific Simulation Models Based on SysML”. en. In: *Systems* 9.4 (Dec. 2021). Number: 4 Publisher: Multidisciplinary Digital Publishing Institute, p. 76. DOI: 10.3390/systems9040076. URL: https://www.mdpi.com/2079-8954/9/4/76 (visited on 12/05/2021).

Adaptation of an auto-generated code using a model-based approach to verify functional safety in real scenarios

Joelle Abou Faysal*†, Nour Zalmai†, Ankica Barišić*, Frederic Mallet *
*Universite Cote d'Azur, Cnrs, Inria, I3S, Sophia Antipolis, France
†Renault Software Factory, Sophia Antipolis, France
Email: joelle.abou-faysal@etu.univ-cotedazur.fr, nour.zalmai@renault.com,
Ankica.Barisic@univ-cotedazur.fr, Frederic.Mallet@univ-cotedazur.fr

***Abstract*—The level of autonomy of our vehicles is rapidly increasing. However, the acceptance of fully Autonomous Vehicles (AVs) depends on the confidence in their ability to operate safely in an uncontrolled environment. Hence, experts and non-experts must have a rigorous method along with adequate tools that can support their exigencies and safety specifications. This paper presents a Domain-Specific Modeling Language (DSML) for defining formal rules and generating flaw-less artefacts, which enables the application of a Safety Analysis of Violations and Inconsistencies (SAVI). The validity of the approach is illustrated on a Renault use case implementation with formal safety goals for autonomous vehicles. Our approach allows designers to detect violation ambiguities and rule inconsistencies on real or simulated scenarios.**

***Index Terms*—Autonomous vehicles, safety rules, model-based system engineering, formal methods, requirement engineering, model development and verification, test and simulation.**

Almost every mode of transportation is becoming autonomous. The main difficult hurdle in the autonomous domain is to guarantee that systems and software components are safe. The automotive industry is investing a lot in deploying self-driving systems in transportation technologies. It is necessary to overcome these challenges before having big fleets of cars on our roads. To avoid the rejection from the public opinion, we need to get them involved in the adoption of safety decisions. Operational safety defined in the ISO 26262 standard [1] implies having a design for the safety component that deals with all unsafe situations. The safety of intended functionality (SOTIF) approach that extends ISO 26262, is defined in ISOPAR 21448.1. SOTIF is concerned with failure causes related to system performance limitations and predictable misuse of the system. Performance limitations or insufficiency of the implemented functions are due to technical limitations such as sensor performance and noise. They can also be due to limitations of the algorithm such as object detection failures and limitations of actuator technology. Safety experts started to use traditional manual approaches to enforce safety decisions [2]. There is a threat of using these approaches, as safety experts' analyses depend on their experiences. Sometimes safety rules are not well formalized and usually are not reusable in the future. In addition, classical exhaustive verification techniques cannot guarantee that the system is safe because of the high degree of uncertainty in the environment. These test-based approaches also lead to a system complexity where time and cost are not under control [3]. Using the Model-Based System Engineering (MBSE) approach to assess software safety, enables formalization, improves reuse of the software, and helps to address safety analysis [4]. Model-driven approaches also address the complexity with a model-centric methodology that exploits domain models rather than documents. The use of Domain-Specific Modeling Language (DSML) enables fast prototyping of those behaviors by using a metamodeling structure. [5]. Endowed with formal semantics it brings a possibility to verify before generating a conforming code.

In this paper, we propose a DSML to verify the safety of Autonomous Vehicles (AVs). Monitors are generated to ensure the functional safety. We apply the study to real scenarios where we visualize many types of breaches.

This paper is organized as follows. Section I presents relevant background information. Section II presents the DSML proposed. We begin by detailing the language development and presenting the user process where he is informed of the procedure to deploy this language. The last part of this section describes our Safety Analysis of Violations and Inconsistencies (SAVI) on Renault use case. Finally, section III concludes and discusses future directions.

I. Background

The design of safety-critical systems involves people with different expertise. All DSML users, whatever their domain of expertise, must correctly evaluate the architecture and understand the impact of their decisions on safety. On the other hand, safety experts also need solutions to ensure a good coverage in the considered safety scenarios and use cases. Fortunately, DSML bridge the gap by translating artefacts from one domain to another, and maintaining a full synchronization with safety models [6]. The certification of the generation process from models to deployed artefacts ensures that designers can safely focus on models, less on implementation details, and therefore reduce development time. Thus, it improves the efficiency of the testing process, almost reduced to integration,

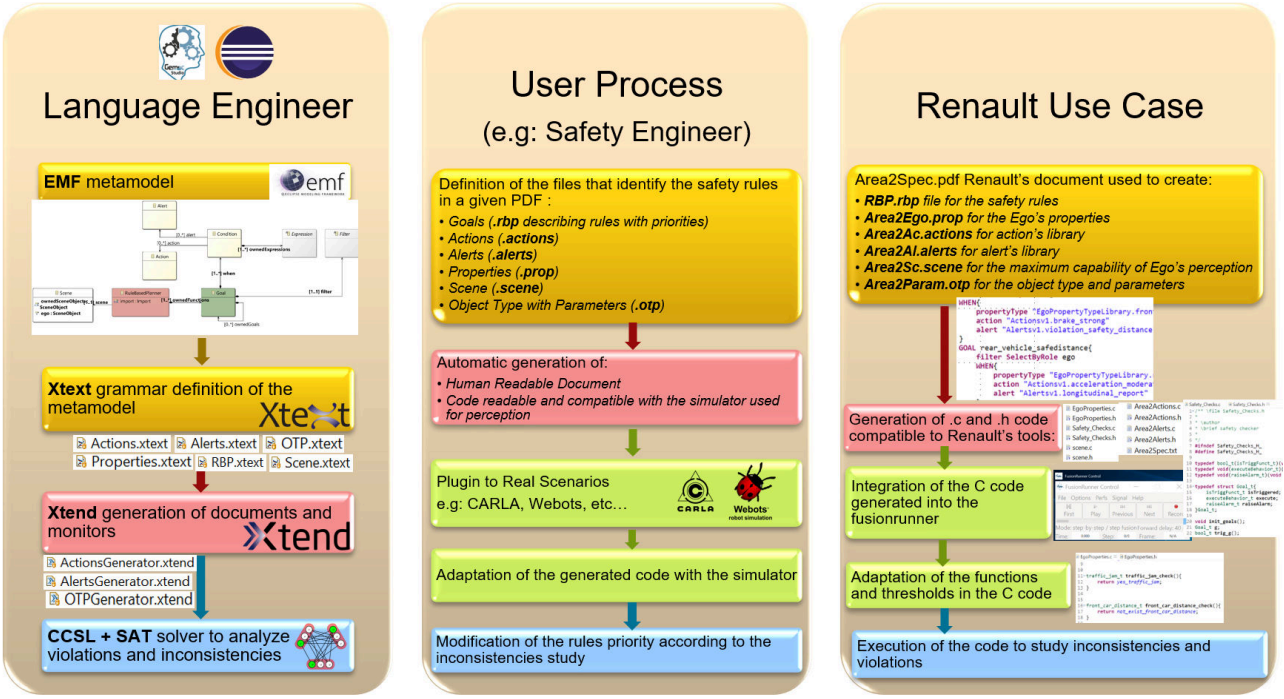


Fig. 1: Three-way views of the approach

as only safe artefacts are produced. Generated artefacts will then be used by engineers to provide a verification failure assessment.

Many approaches use the Model-Based System Engineering (MBSE) approach to provide a verification failure assessment. A safe autonomous vehicle trajectory DSML consists of driving a vehicle through a set of known waypoints by connecting motions in a sequence [6]. This work provides always a safe trajectory and creates a motion navigator for the autonomous car. Unfortunately, their domain is not applicable if the trajectory is already planned. It also does not provide to the user the alerts and the measures to take during an unsafe scenario, and does not know the original cause of the violation. An intelligible model is done to guarantee safety [7], but it is parametric and depends on environmental conditions. Sensor uncertainty dilemma has been handled using probabilistic models with formal specifications [8]. A rule-based strategy was also designed to evaluate sensors' dependability [9]. Yet, the main problem of autonomous driving systems perception is still not explored. In this paper, we give the user the capability to add parameters and specifications relying on the sensors and the environment, to examine ambiguities. Furthermore, a SceML study was carried out [10] to create a graphical model to generate new scenarios or test existing ones using Machine Learning. They facilitate the scenario creation process, but not the formal description of safety rules. This is why we apply formal semantics to this approach that allows specifying, designing, and analyzing the system. Mathematical reasoning can improve software reliability and dependability and is essential when developing complex software systems.

We develop an abstract model expressive enough to support safety analyses. We rely on the GeMoC framework [11] as it integrates a set of languages and tools based on the Eclipse Modeling Framework (EMF) to ease the definition of new DSMLs with inherent concurrency [12]. Additional components may be included to achieve artefact generation, such as correct syntactical code generation [13].

II. A Domain-Specific Modeling Language for the Safety of Autonomous Vehicles

We have developed a Domain-Specific Modeling Language (DSML) to verify the safety of Autonomous Vehicles (AVs). To ensure the acceptability of this language, we need to support the specification of the formal safety rules and the environment. An automatic generation of a monitoring system assists the interpreter by triggering alarms and possible recover maneuvers. Those monitors are expected to enable the user to detect inconsistencies and violations between rules and priorities. After performing a Safety Analysis of Violations and Inconsistencies (SAVI), the user can modify the rules and repeat the same cycle to ensure the correctness of the autonomous behavior.

We present three perspectives of our approach as seen in Fig. 1. First, from a language engineer perspective in section II-A, we describe necessary steps to implement the Extensible Platform for Safety Analysis of Autonomous Vehicles (EPSAAV) in [14]. EPSAAV is our DSML specified in our previous study where we detailed our metamodel and concrete syntax to auto-generate monitors. Second, we introduce gen-

eral steps for user perspective in section II-B. In this section, we detail how the user should use our approach and what he needs to do to perform a safety assessment. The third point of view describes Renault's use case in section II-C to test the approach and validate a SAVI in section II-C4.

A. Language Development

One of the main objectives of our language is to empower designers and experts with formal and yet practical solutions to describe the environment, the expected behavior, and the safety rules for the car under design. This is described in sections II-A1 and II-A2.
The second objective is to support the automatic generation of : (a) a human-readable document describing the rules and libraries used, and (b) monitors that allow the user to adapt output data of the simulator used with our safety rules. The violation of safety rules triggers alarms through these monitors. Section II-A3 details this part.
The third objective described in section II-A4, is to test, verify and identify inconsistencies. Sometimes a rule can lead to similar or contradictory behavior to another one. To avoid this, an inconsistency study must be conducted.
We use the open-source Gemoc tool [11] as it covers all aspects of a DSML development; from abstract and concrete syntax definition to semantics and operations. Gemoc is easy to integrate with all those technologies. It integrates solutions to ease the code generation and includes solutions to describe concurrent behaviors [12].

1) Abstract syntax: contains a graphical description of the metamodel. We use Eclipse Modeling Framework (EMF) technology. EMF is a framework and code generation facility that defines the model and generates implementation classes. EMF unifies the three important technologies: Java, XML, and UML. EMF model is the common high-level representation that glues them all together. The metamodel describes the classes and the relationships of the environment. For example, Fig.2 describes the abstract part of the rule-based planner, where *goals* are specified and composed of *conditions* referred to *alerts* and *actions*. The *goals* could be filtered either by role or by expression. The *RuleBasedPlanner* refers to a *Scene* that captures the perception capability of the vehicle. Rules are combined with logical operators. It is also important to prioritize rules in the case where several of them can be simultaneously triggered with contradictory behavior (e.g. break hard vs. maintain speed). The notion of *SelectByGoal* allows executing goals either in parallel or sequentially. Once the abstract syntax is specified, we build a concrete syntax.

2) Concrete syntax: defines the concrete terms that should be used by designers and the grammatical rules to bind them. We use Xtext technology to provide a concrete textual syntax to our language. In our language, we create Xtext files for some of the classes as defined in Fig.3. We get a separate description for the *scene*, *actions* and *alerts*, *parameters* and *properties*, and *goals* and priorities. The extensions defined for each class serve to create new files and libraries.

3) Auto-generation of a human-readable document and monitors: which enables safety engineers to easily integrate them with the chosen simulator and adapt the auto-generated code to check violations and consistencies described in section II-A4. We use Xtend technology to give the operational semantics and assign a behavior to each of the declarations of our DSL. We generate a readable document that gave the engineer the possibility to validate and communicate his choice. Usually, this document is the main artefact used by safety engineers. Here, the document can be generated when the model is updated. We also generate code that eases interfacing with the output of the simulator and checking the violations. The code enables the user to analyze and identify rule consistencies.

4) Establishing satisfiability to avoid inconsistencies: using SAT Solver. SAT Solvers have been used in many practical applications. We expect to enable safety engineers to create a resilient and safe driver monitoring system that checks safety rules defined previously and investigates inconsistencies, possibly assigning priorities to sort them out. The SAT problem is a decision problem, which, given a propositional logic formula, determines whether there is an assignment of the variables that makes the formula true [15]. This will test rule inconsistencies, and verify solutions for all the rules. All logic operations for rules specification are translated to specific forms of coding. The task comprises of testing the rules with specific formulas by auto-generating specific checks for each rule.

B. User Process

The user has three tasks to analyze and guarantee safety in real or simulated scenarios, as seen in Fig. 4. First, he must describe the environment (scene, parameters, and properties), the behaviors (alarms and actions), and the security rules with priorities. Fig.5 is an example of rules (also called goals) introduced on the user interface. We define a ***RuleBasedPlanner*** named ***rbp***, referring to a scene defined in another file where we introduce the capacity of ego's perception using another syntax. Two properties (***prop1*** and ***prop2***) were previously defined having different states in Fig.6. Libraries of actions and alerts were also created. Conditions are put together through logical expressions.

Safety engineers assign every goal to a type that is either a priority or a constraint. If a goal has a priority on another, the second one should not be executed if the first one is true. If a goal has a goal type as a constraint, both are executed in parallel. This priority-constraint categorization helps the engineer choose which rule should be accomplished before or at the same time as another one. It implies a hierarchy of priorities between all the rules. In Fig.7, we show an example of two contradictory actions triggered at the same time. Goal 1 consists of having three conditions : (1) following the PV, (2) respecting speed threshold, and (3) respecting safety distance of 2 s. The first goal leads to a light acceleration, contrary to the second goal that generates emergency braking. It consists

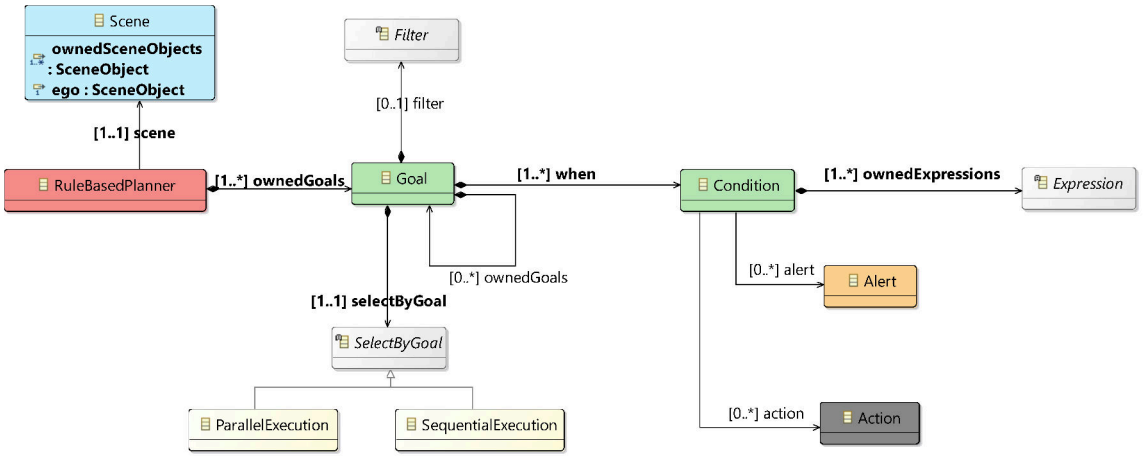


Fig. 2: RuleBasedPlanner Abstract metamodel

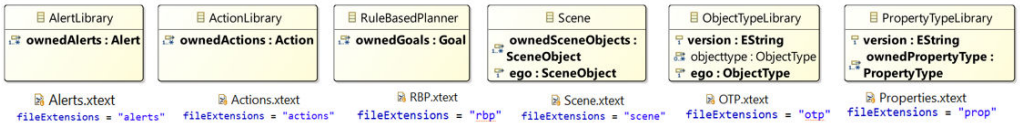


Fig. 3: Concrete Xtext Files in our metamodel

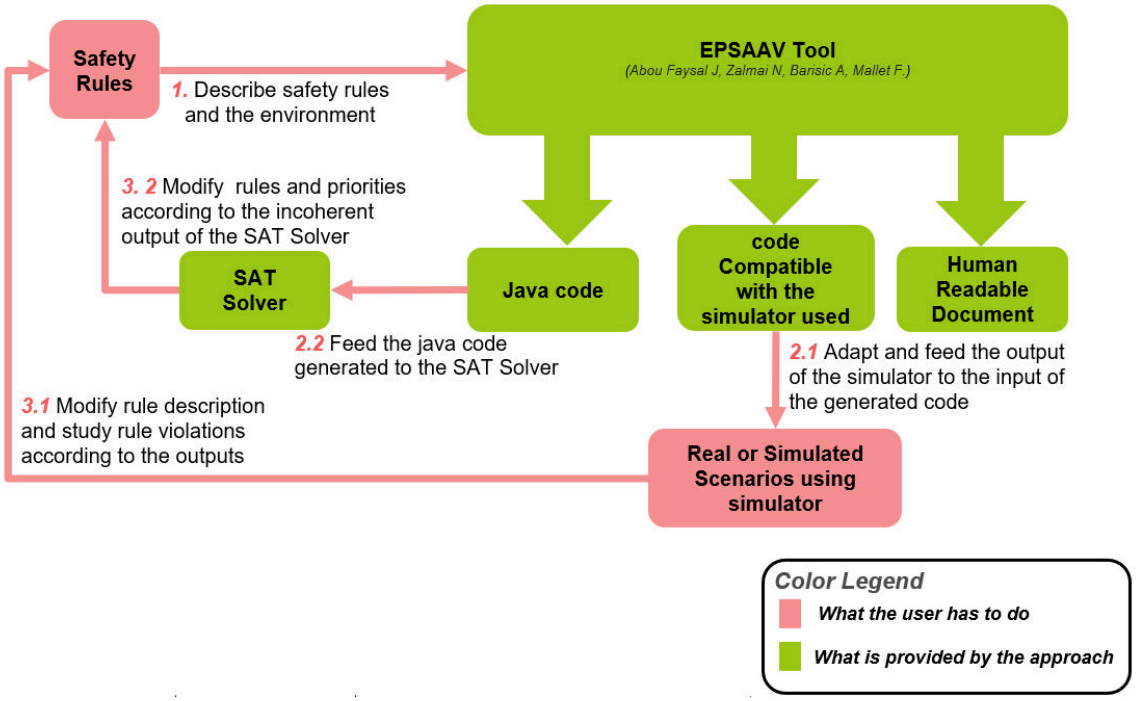


Fig. 4: User process to use the approach

of respecting a Time To Collision (TTC) for every Vehicle Road User (VRU), e.g. pedestrians. In this case, emergency braking has a strong priority over light acceleration, and this priority needs to be carefully defined within the priority-constraint sorting.

When the environment definition is finalized, we execute

```
RuleBasedPlanner rbp{
    scene ^Scene
    GOAL goal1{
        GoalType Constraint
        WHEN {
            AND (
                propertyType "Properties.prop1" is "Properties.prop1.state1",
                OR (
                    propertyType "Properties.prop2" is "Properties.prop2.state2",
                    propertyType "Properties.prop2" is "Properties.prop2.state3"
                )
            )
            action  "Actions.action1"
            alert "Alerts.alert1"
        }
    }
}
```

Fig. 5: Formal Rules using logical expressions described by the user.

```
//this is the definition of the library for the Ego properties

PropertyTypeLibrary Properties
{
    version '1'
    state "prop1" CanBe
    "state1"
    "state2"
    state "prop2" CanBe
    "state1"
    "state2" operator">=" value 2.0 unit "s" //[operator] [float] [valuetypes e.g m/s]
    "state3" operator">=" value 1.2 unit "s"
}
```

Fig. 6: Properties and states using formal syntax defined by the user.

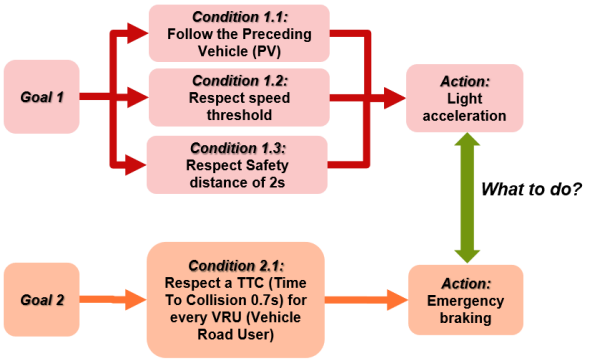


Fig. 7: Case where emergency braking should have a stronger priority over light acceleration.

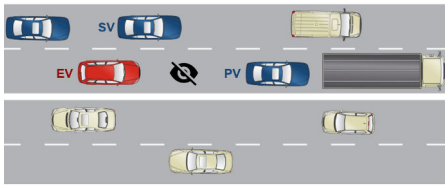
an automatic generation of monitors and a human-readable document. The main interest of this framework is to give the engineer the ability to define all the rules that he thinks should be triggered. It also gives him the ability to track rule modifications. In case of changes, the tool has the potential to generate a new code according to the specified rules. It is a generic tool for flexible rules that can be used not only for safety domains but also for security and failure domains. Depending on those rules, a specific code is then generated related to those formal rules. The document legibly describes the rules and libraries, so it helps to communicate the accuracy and completeness of those rules. The monitors allow the user to adapt perception data with safety rules and investigate rule inconsistencies. For the generated Java code, we feed it to the SAT Solver to test inconsistencies and modify safety rules. For the code compatible with the simulator used, the user has to take the generated monitors and plug them into real or simulated scenarios such as Webots [16], Carla [17]...
Then, an adaptation of our input data with the output data of the simulator is necessary to test, verify and identify rule violations using formal languages. This part is further detailed in our Renault use case in section II-C where we detect ambiguities in rules that may lead to violations.

C. Renault Use Case

To apply operational safety on the trajectories of self-driving cars, we will need the elementary data necessary for the system, such as rules and libraries. We translate scenarios, their risks and the measures to be taken, to formal rules describing lateral and longitudinal control. The EPSAAV tool facilitates this formalization process and improves communication between the engineers. We proceed to an auto-generation of a human-readable document and monitors, that will be helpful to acheive a SAVI.

1) Formal Goals Description: we introduced five scenarios in our use case as shown in Fig.10 that deal with risks of no/insufficient or unexpected braking, no lateral correction, and unexpected lane change. We take a scenario that describes the case where the Ego vehicle (EV) is having a problem detecting lines and/or Preceding Vehicle (PV) in Fig.8a. The risk we have is a lateral collision with the Side Vehicle, or Straddling Vehicle (SC). In case of a line or PV loss, the system shall maintain the EV in the lane using the remaining information. If the line disappears more than t6 seconds, the system must trigger an ***Emergency Operation*** (***EOP1***). Fig.8b formalizes this scenario by creating a goal that contains one condition to avoid the risks and to trigger behaviors. Goals can have multiple conditions. For each condition (*WHEN*), there is logical expressivity that constitutes the syntax.

2) Generation of documents and monitors: using the EPSAAV tool, we generate a readable document that gave the engineer the possibility to validate and communicate his choice. We also generate (a) C code which eases interfacing with the output of the simulator and checks the violations, and (b) Java code which enables the user to analyze and study rules incoherences. The C code is compatible with Renault's simulator called *"FusionRunner"*. We chose *"FusionRunner"* among all simulators for many reasons: (1) it executes perception's algorithm and sensors fusion data of Renault, (2) it runs driving data on open roads and many other real-life or simulated scenarios, and (3) it provides practical information (such as data sensor, data fusion, TTC, PT, Autonomous Emergency Braking (AEB), Adaptive Cruise Control (ACC), ...). We integrate this code into the perception algorithm and then adapt the safety functions by feeding the simulator's output to the input of the generated code.
The auto-generation consists of: (1) translating all the environment defined to functions compatible with Renault's language, (2) filling the functions needed to test rules according to defined thresholds, properties, and parameters



(a) Lateral control scenario: no lane/PV detection case.

```
when
    S_stable_control is stable AND
    (
        S_front_car_tracking is disappeared_more_than_t1 OR
        S_line_detection is no_detection_in_more_than_t6 OR
    )
then goal:
    executing A_emergency_operation_1
```

(b) rule specification formalized for this use-case scenario.

Fig. 8: Formalizing scenario to goal containing one condition and using logical expressiveness in case of a no lane/PV detection scenario.

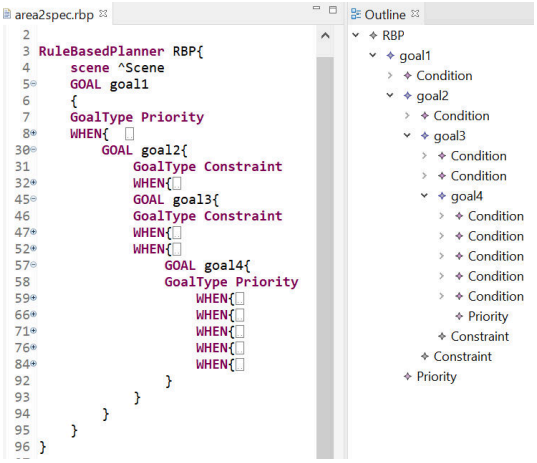


Fig. 9: Five formal rules introduced in our Renault Use Case with parallel and sequential executions.

Longitudinal Control	*Risk of no/insufficient braking*	Deceleration
	Risk of unexpected braking	False recognition
Lateral Control	*Risk of no lateral correction*	No lane detection
	Risk of unexpected lane change	Wrong lane detection
Lateral & Longitudinal	*Risk of no/insufficient braking /no lateral correction*	Swerving

Fig. 10: Five scenarios violations introduced in Renault's use case.

given in the beginning, and (3) creating a function that treats all goals and conditions in priority or parallel, depending on the goal type. We took the C auto-generated code and implemented it into the *"FusionRunner"*.

3) Interfacing the auto-generated C code with Renault's simulator: Fig.11, Fig.12, and Fig.13 constitute the windows output to better visualize violations defined in the rule-based planner. In Fig.11, we can visualize a real-time video in the Fusion Context View and the sketch of this video in the Fusion Display. In Fig.12, and Fig.13, we can see all the binary states' properties, five goals with their conditions, actions, and alerts triggered in the SAFETYCHECKER window output. We store, at each slot, the parameters' values from fusion data. The windows output helps us inspect

Fig. 11: Fusion Control View and Fusion Display windows at step 4217.

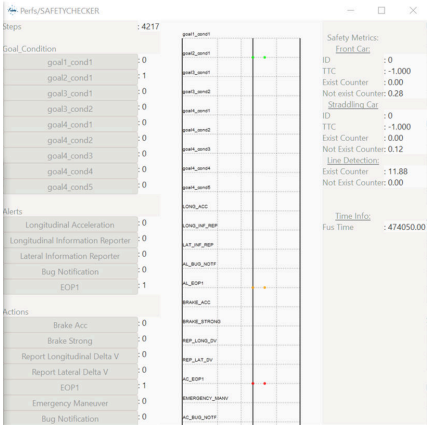


Fig. 12: SAFETYCHECKER Window for safety analysis of violations that shows the parameters, the goals with the alerts and actions triggered at step 4217.

violations and ambiguities in the goals declaration.
The results in the following section indicate that it is possible to reuse the model defined to verify safety in the automotive industry. It also shows the benefits and efficiency of using our DSML. In addition, it makes safety exploration easier for engineers, therefore improving the quality of their surveys.

4) Safety Analysis of Violations and Inconsistencies (SAVI): an ambiguity is presented in a violation that occurred in Fig.12

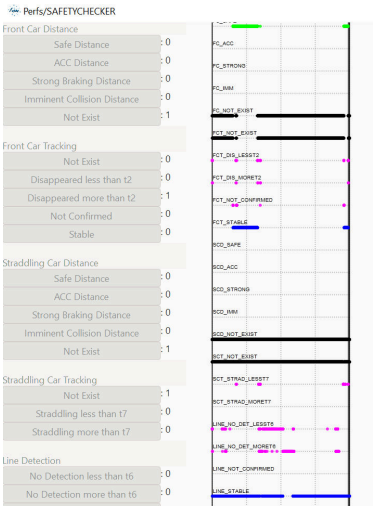


Fig. 13: SAFETYCHECKER Window for safety analysis of violations that shows the properties binary states at step 4217.

at step 4217. This violation is assigned to the *goal2_cond1* presented in 8b. The ***EOP1*** is triggered according to this goal when the PV disappeared more than a threshold. This is ambiguity because if we look at the Context View in Fig.11 at that step, we can see that there is no PV, and it is not logical to trigger an ***EOP1***.

We propose a modification for the safety rule to erase this ambiguity by adding a condition on the stability for the line detection as shown in Fig.14. To study rule inconsistencies,

```
when
    S_stable_control is stable AND
    (
    NOT ( S_line_detection is stable) AND
        (
            // 2.4.1.1
            S_front_car_tracking is disappeared_more_than_t1 OR
            // 2.4.1.1, 2.4.1.3.1
            S_line_detection is no_detection_in_more_than_
        )
    )
then goal:
    executing A_emergency_operation_1
```

Fig. 14: Modification in the rule expression to erase ambiguity.

the generated Java code fed to the SAT Solver will help us verify all solutions for all the conditions and goals. This work is still in process. By that, we will be achieving a SAVI.

III. Conclusion and Perspectives

This paper introduces a methodological proposal for using the MBSE approach in the automotive safety field. We describe the language development viewpoint where we talked about the abstract and concrete parts, the auto-generation of monitors and documents, and the SAT solver to study the inconsistency. We detail the user process tasks. The user has to specify requirements formally using EPSAAV to help him generate what he needs for safety analysis. We also show code generation that the user needs to link with the simulator's perception data. In our use case, we generated C code and tested and visualized goals to demonstrate that this approach is feasible. We show a goal's ambiguity, and the notifications triggered to the user. We propose a modification to delete an uncertain violation. If we could find all violation ambiguities and analyze inconsistencies, we can assure that all specifications are realizable and complete. We can apply this generic tool to test rules other than safety domains, such as security or failure domains. Ethic people can apply this framework to two different sets of rules. They can then select what is the best set of generated monitors. For future work, we will auto-generate java code for the SAT Solver. The SAT problems will help check rule inconsistencies and achieve SAVI analysis. We will also test rules on more real-life scenarios and analyze their output on more use-cases.

References

[1] M. A. Gosavi, B. B. Rhoades, and J. M. Conrad, "Application of functional safety in autonomous vehicles using ISO 26262 standard: A survey," in *SoutheastCon 2018*. IEEE, 2018, pp. 1–6.

[2] C. Ackermann, J. Bechtloff, and R. Isermann, "Collision avoidance with combined braking and steering," in *6th International Munich Chassis Symposium 2015*. Springer, 2015, pp. 199–213.

[3] Y. Sirgabsou, C. Baron, C. Bonnard, L. PAHUN, L. Grenier, and P. Esteban, "Investigating the use of a model-based approach to assess automotive embedded software safety," in *13th International Conference on Modeling, Optimization and Simulation (MOSIM20)*, AGADIR, Morocco, Nov. 2020. [Online]. Available: https://hal.laas.fr/hal-02942695

[4] J. Kapinski, J. V. Deshmukh, X. Jin, H. Ito, and K. Butts, "Simulation-based approaches for verification of embedded control systems: An overview of traditional and advanced modeling, testing, and verification techniques," *IEEE Control Systems Magazine*, vol. 36, no. 6, pp. 45–64, 2016.

[5] K. Falkner, V. Chiprianov, N. Falkner, C. Szabo, and G. Puddy, "A model-driven engineering method for dre defense systems performance analysis and prediction," in *Handbook of research on embedded systems design*. IGI Global, 2014, pp. 301–326.

[6] M. Bunting, Y. Zeleke, K. McKeever, and J. Sprinkle, "A safe autonomous vehicle trajectory domain specific modeling language for non-expert development," in *Proceedings of the International Workshop on Domain-Specific Modeling*, 2016, pp. 42–48.

[7] S. Shalev-Shwartz, S. Shammah, and A. Shashua, "On a formal model of safe and scalable self-driving cars," *arXiv preprint arXiv:1708.06374*, 2017.

[8] C. Tessier, C. Cariou, C. Debain, F. Chausse, R. Chapuis, and C. Rousset, "A real-time, multi-sensor architecture for fusion of delayed observations: application to vehicle localization," in *2006 IEEE Intelligent Transportation Systems Conference*. IEEE, 2006, pp. 1316–1321.

[9] M. Gao and M. Zhou, "Control strategy selection for autonomous vehicles in a dynamic environment," in *2005 IEEE International Conference on Systems, Man and Cybernetics*, vol. 2. IEEE, 2005, pp. 1651–1656.

[10] B. Schütt, T. Braun, S. Otten, and E. Sax, "Sceml: a graphical modeling framework for scenario-based testing of autonomous vehicles," in *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, 2020, pp. 114–120.

[11] B. Combemale, O. Barais, and A. Wortmann, "Language engineering with the gemoc studio," in *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*. IEEE, 2017, pp. 189–191.

[12] B. Combemale, J. DeAntoni, M. V. Larsen, F. Mallet, O. Barais, B. Baudry, and R. B. France, "Reifying concurrency for executable metamodeling," in *Int. Conf. on Software Language Engineering, SLE*, ser. Lecture Notes in Computer Science, M. Erwig, R. F. Paige, and E. V. Wyk, Eds., vol. 8225. Springer, 2013, pp. 365–384.

[13] S. Kelly and J.-P. Tolvanen, *Domain-specific modeling: enabling full code generation*. John Wiley & Sons, 2008.

[14] J. Abou Faysal, N. Zalmai, A. Barisic, and F. Mallet, "Epsaav: An extensible platform for safety analysis of autonomous vehicles," *Advances in Model and Data Engineering in the Digitalization Era (MEDI 2021 Workshops)*, 2021.

[15] Borealis AI. Tutorial 9: Sat solvers i: Introduction and applications. Accessed: 2021-12-19. [Online]. Available: https://www.borealisai.com/en/blog/tutorial-9-sat-solvers-i-introduction-and-applications/

[16] Webots for automobiles. Webots user guide and reference manual. Accessed: 2020-06-05. [Online]. Available: https://cyberbotics.com/doc/automobile/introduction

[17] A. Dosovitskiy, G. Ros, F. Codevilla, A. M. López, and V. Koltun, "CARLA: an open urban driving simulator," *CoRR*, vol. abs/1711.03938, 2017. [Online]. Available: http://arxiv.org/abs/1711.03938

Session We.1.C

HW Formal Verification

Wednesday 1st June

11:30

–

Room Pastel

Formal Processor Modeling for Analyzing Safety and Security Properties

Benjamin Binder, Samira Ait Bensaid, Simon Tollec, Farhat Thabet, Mihail Asavoae and Mathieu Jan
Université Paris-Saclay, CEA, List, F-91120, Palaiseau, France

***Abstract*—The emergence of open hardware initiatives exposes the exact behavior of hardware designs, which thus can be analyzed and combined with application-level semantics to formally verify complex safety and security properties. Formal verification relies on appropriate abstract models to cope with the omnipresent state-space explosion. In this paper, we compare the different needs when designing abstract processor models for the evaluation of timing predictability, a safety property, and for security assessments when considering fault injections. We also report how the process of building these abstract processor models could be automated.**

***Index Terms*—Formal Methods, Hardware Models, Safety and Security Properties**

I. Introduction

The design of complex computational systems, such as Cyber-Physical Systems (CPSs) or the Internet of Things (IoT), is facilitated by the emergence of open hardware initiatives [2]. Such initiatives propose software-like development workflows, from complex high-level Hardware Description Languages (HDLs) [6] down to circuits, while using sophisticated compilation chains. These approaches favor the availability of hardware designs, which can thus be used as (detailed) golden models, replacing the standard manual reference where only certain design details are provided.

These CPSs and the IoT are often subjected to safety and/or security requirements. Ensuring those requirements can be done with various degrees of confidence, from informal argumentation to formal verification of properties. When using the latter approach, the formal verification of software and hardware parts of a system is generally performed as separate activities and focuses mostly on functional correctness. For example, a binary representation of an application can be executed with formal Instruction Set Architecture (ISA) semantics [1], while hardware designs can be composed from formally proved modules [14]. However, in the former there is no hardware model while the latter ignores the software level. The availability of hardware designs, combined with ISA application representations, enables new possibilities in the formal verification of safety and security properties at system level, in particular by integrating non-functional characteristics such as timing. We restrict the notion of hardware design to that of processor design.

In this paper, we study how to design formal processor models so as to prove safety and security properties, such as timing predictability and assessments of Fault-Injection (FI). We present several instances of a general workflow, shown in Fig. 1, to address various needs as designing specialized abstractions. Due to the availability of a processor design, different abstract processor models can be derived while analyzing its code, both data and control paths (i.e., model generation in Fig. 1). In the case of security assessments under FI, such an abstraction requires a high temporal and spatial accuracy on certain parts of the design while non-important aspects of the system can be removed. Both control and data path are relevant, since security properties should follow how instructions (i.e., control) are correctly (or not) executed on the processor (i.e., data). In the case of timing predictability evaluations, the datapath is abstracted to black-box, but cycle-accurate, pipeline stages imposing timing constraints to the instruction flow, except for the execute stage where in addition

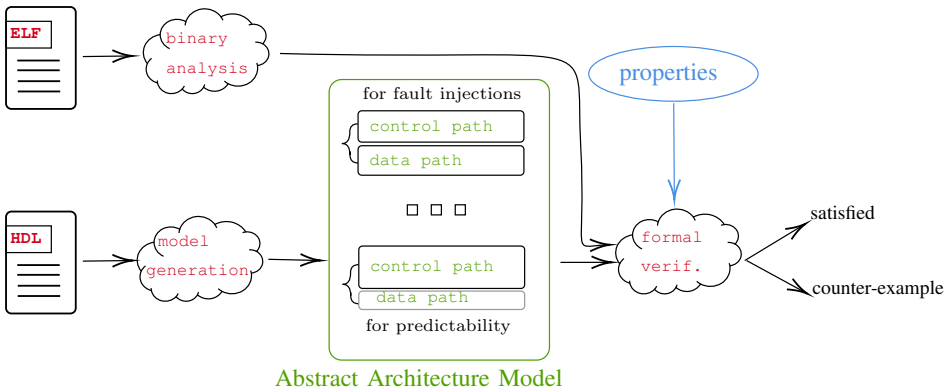


Figure 1: General workflow for the verification of system-level safety and security properties.

the scheduling algorithm mapping instructions to functional units is fully captured. The actual system executions capture how a given binary code is executed on the processor design and hence, we extract a binary representation (e.g., a control-flow graph or explicit traces) to represent these sequences of instructions (i.e., binary analysis in Fig. 1) in the formal verification step. For timing predictability, the captured instruction semantics is limited to dependencies between instructions.

The remainder of this paper is organized as follows. Section II describes related work on formal modeling of micro-architectures. In Section III, we show how to design abstract processor models to assess timing predictability and we apply this approach over an out-of-order pipeline to detect timing anomalies. In Section IV, we present how to design abstract processor models to identify exploitable fault-injection points and we illustrate this process over a RISC-V processor. In Section V, we look how to automate the construction of formal hardware models, using underlying hardware compilation chains. Finally, Section VI concludes the paper.

II. Related Work

The literature around modeling real-time systems often elaborates results using a pen and paper approach, without relying on formal and executable models—necessary in the automatic verification of properties. There are several exceptions, for example, the framework PROSA [13], which focuses on the correctness of schedulability analyses (and not microarchitectural modeling). The SIC pipeline is accompanied with a formal semantics based on a detailed transition system but that is not intended to execute in a verification methodology [21]. Even when formal models are assumed for the automatic verification, few modeling details are provided [18]. Model checking, a popular formal verification technique, has been used to estimate the worst-case execution time (WCET), however on in-order architectures [17], [35]. It has been used to co-validate hardware-software designs wrt. timing properties on a simple processor [3]. It has also permitted verifying the absence of particular timing anomalies in predictable pipelines [26] and studying their occurrences in an industrial processor [9]. A similar approach to ours for verifying a specific safety property by model checking has already been proposed [5], [4]. In the present work, we nevertheless adopt a more generic viewpoint, e.g., letting the number of resources be provided as parameters. A similar architecture template (cf. Fig. 2) is used in [33], however together with an analytical approach.

Formally verifying FI-related security properties has been the subject of several works, in general at the ISA level. The modeling of the program execution has allowed to explore all the possible effects that certain FIs, such as register corruption, may have on program execution [10], [41], [19]. Nevertheless, recent results [20], [30] have shown the interest of considering the microarchitectural behavior to find FI points ignored by a strictly ISA-based study. This also helps to limit spurious vulnerabilities (i.e., false positives) that are not feasible in practice. However, to the best of our knowledge, no work has combined formal modeling of hardware and the verification of FI-related security properties. In the scope of general security properties, formally verifying hardware/software co-designs has been the subject of numerous works, such as [38]. However, they mainly focus on functional properties, such as the correctness of updating ISA-level states of processors [22], viewed mainly through the register file within the pipeline, or detecting bugs in specific hardware components synthesized on FPGA [37]. Note that safety properties can also target functional failures [16], which share similar modeling requirements with our security-assessment approach.

Generating formal models from hardware designs is an extensively studied problem [28], [25], [36], [32], [23]. While these works address Verilog or VHDL languages, we consider high-level design languages that pose different challenges when generating abstract processor models specific to the targeted properties. This is possible due to hardware compilation frameworks that are currently developed (see [6], [43], [40] to cite a few). Note that our contribution related to the automatic generation of abstract processor models is agnostic to the choice of a hardware compilation tool chain.

III. Abstract Processor Models for Timing Predictability

In this section, we target the verification of non-functional, safety properties, namely relative to the processor timing behavior. We first address the general modeling needs for timing predictability, then we exemplify this modeling approach through an Out-of-Order (OoO) template, and finally we verify a particular safety property on the OoO model.

A. Abstract Modeling for Timing Predictability

A suitable model necessarily integrates both hardware and software features, whose combination characterizes the system and in particular its non-functional timing characteristics. The properties are not correlated to the functional complexity of architectures, which materializes into the datapath. More precisely, we do not need to consider the functional aspects beyond their impact on the pipeline-level timing behavior. Instead, we need to develop an abstract formal model of the processor focusing on the instruction progress—the software characteristics—through the successive pipeline stages of the processor—the hardware characteristics. Finer models are unnecessary since they describe changes in internal, hidden states of the datapath, typically a matter of functional correctness. On the contrary, pipeline-level models are required since pipeline stages are essential to the cycle-accurate timing behavior, which enables to observe external events, e.g., the full completion of an instruction.

Our abstraction thus needs to precisely delimit the pipeline stages from the datapath of the hardware architecture, and to extract the control signals that impact the timing behavior of the control path according to the pipeline stalling logic. We also need to map at any time instructions onto the identified

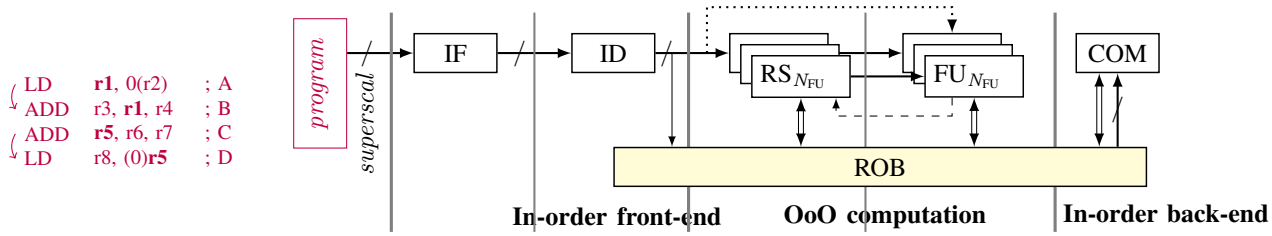


Figure 2: Representative hardware template of an OoO pipeline based on Tomasulo's algorithm. The pipeline has N_{FU} functional units and is able to fetch, decode and commit *superscal* instructions per cycle from a software specification (*program*) [8].

stages that process them, from the input program—this is the combination of the hardware and software specifications. Finally, our model also requires explicitly capturing the execution time, in view of verifying properties.

B. Application to an Out-of-Order Pipeline Template

Hereafter, we exemplify the main features of a suitable model for the verification of timing-predictability properties, from a simplified standard OoO-pipeline template, shown in Fig. 2. We present an overview of this template designed for the detection of timing anomalies [8]. We intend to provide formal-modeling details below. This template consists of stages that process instructions in the same order as specified in the program (i.e., IF, ID and COM), namely the in-order front-end and the in-order back-end, as well as of Functional Units (FU) associated with Reservation Stations (RS) and a Reorder-Buffer (ROB) in order to implement Tomasulo's algorithm [44] for OoO computation. Instructions are dispatched to the ROB that keeps track of their status (pending/completed/committed) (⟷), which is used to schedule instructions to the FUs taking into account data dependencies and to later commit instructions in-order. Results from the FUs are bypassed to all RSs (⇢), which essentially avoids waiting for the ROB update and allows back-to-back executions.

We encode the template introduced above into a formal specification (written in the TLA+ modeling and verification language [29]). In our abstraction, pipeline stages do not have side effects, such as a write to the memory or the register file. We consider multi-cycle instructions that thus may cause stalling. The pipeline timing behavior depends on the number of units for each stage (cf. *superscal* and N_{FU} in Fig. 2), on the program dependencies that clearly restrict OoO computations, and, when needed, on the mere information of the required computation clock cycles.

1) Abstract Datapath and Computations: The specification relies on hardware parameters, i.e., *superscal* and N_{FU}, which allow to represent a particular version of the pipeline template by fixing its abstract datapath. We define a state variable for each pipeline stage ($\_IF$, $\_ID$, $\_RS$, $\_FU$ and $\_COM$), which notably contains the instructions that are currently processed. The specification also relies on a software-execution parameter, i.e., *program*, which specifies the input instruction sequence with increasing addresses[1] associated with information about the mapping onto the hardware architecture. This information originates from the analysis of the concrete program: each instruction embeds the set of admissible functional units (as an abstraction of the functional instruction class), as well as the set of possible latencies related to timing-variable stages (as an abstraction of the intended computations). The variables related to a timing-variable stage also contain the current elapsed latency (i.e., a counter) and the total required latency in the stage (assigned from the *program* input parameter). The memory is not explicitly modeled, but the IF stage and the FUs instead feature a variable timing behavior, since for instance they both may perform memory accesses resulting either in an instruction/data-cache hit or miss. The register file is not modeled either, but only the (Read-After-Write) data dependencies (⇂), which are explicitly encoded in *program*. The resulting abstract specification allows for all the *behaviors*, i.e., series of states, that are concretely made possible by different initial hardware states (e.g., the initial cache content), considering the execution of the input instruction sequence on the target architecture. It remains to actually make instruction classes progress through the pipeline, i.e., to encode the control path from the established datapath and the execution information.

2) Timing-Oriented Modeling of the Control Path: The template specification is a *transition system* (TS) characterized by an *initial-state predicate* and a *next-state relation* built from actions, namely transition predicates relating the values of variables in the current state (e.g., x) to their values in the next state (x'). We consider that the pipeline is initially empty of any instruction (cf. the datapath state variables). Transitions model the control path, which entails changes in the datapath state. In order to get a cycle-accurate abstraction of the control path, a transition models one clock cycle, where each stage processing is modeled by an action involving a datapath variable. The additional state variable $currCycle$ is a counter modeling the absolute time ($currCycle' = currCycle + 1$ as long as the input sequence has not fully executed). Finally, the $prog$ state variable is a record monitoring the execution, with a field ($rest$) containing the remaining instructions (not fetched yet) from *program* and a field ($exec$) modeling the ROB. Each of these fields are sequences of instructions, where the

[1]We exclude branch instructions, thus focusing on one program path.

instructions are nested records, e.g., with Booleans *completed*, *committed* for the instruction status in the ROB.

We now illustrate how the abstract datapath state is used in order to accurately model the control path, by focusing on one of the most critical, OoO-specific elements of the control path. Modeling the scheduling of instructions to FUs requires selecting the next pending instruction. It relies on the instruction status in the ROB.

a) Prerequisite Operator: Let us define an operator *Exec* used in order to specify the next-state relation. It returns the set of the ROB indexes of the instructions that will have already completed in the next cycle.

$$NxtFUBusy(i) \triangleq \_FU[i].currLat < \_FU[i].baseLat$$
$$Exec \triangleq \{i \in 1 \,..\, Len(prog.exec) :$$
$$\lor\ prog.exec[i].completed$$
$$\lor \exists j \in 1 \,..\, N_{FU} : prog.exec[i].PC = \_FU[j].PC$$
$$\land \neg NxtFUBusy(j)\}$$

The operator *Exec* returns the set of indexes in the range of the current ROB (first line of the definition) s.t. the relevant Boolean field (*completed*) of the corresponding instructions (*exec*[i]) is set (first disjunct) or a back-to-back execution is possible (second disjunct). In the latter case, the instruction itself (*PC* field) is currently handled by one of the FUs, i.e., it is the instruction of the j-th element of the $\_FU$ state variable (penultimate line), and the instruction in this FU is to leave the FU in the next cycle (last line). Indeed, the $NxtFUBusy(i)$ operator uses the information about latencies contained in the $\_FU$ variable to determine whether the instruction currently handled by a given FU should remain in the FU in the next cycle and, hence, cause a pipeline stalling. This operator compares the current latency *currLat* of the i-th FU with the total required latency *baseLat*. The operator *Exec* is used to update the ROB field *exec* of *prog* in each cycle.

b) Scheduling on the FU*s:* Based on the operator *Exec*, we can now specify the very scheduling of instructions to the FUs. The operator $NxtFU(i)$ returns the instruction that is to be scheduled to the i-th FU in the next cycle, or a special instruction *empty* that models the absence of an instruction:

$$NxtFU(i) \triangleq \text{IF } NxtFUBusy(i) \text{ THEN } empty$$
$$\text{ELSE LET } minReady \triangleq Min(\{x.pc : x \in$$
$$\{y \in \_RS[i] \cup FURout(i) :$$
$$\forall z \in y.dep : z \in Exec\}\}) \text{IN}$$
$$\text{IF } minReady = 0 \text{ THEN } empty$$
$$\text{ELSE CHOOSE } x \in \_RS[i] \cup FURout(i) : x.pc = minReady$$

In the case that the i-th FU does not suffer stalling and thus may accept a new instruction in the next cycle (i.e., $NxtFUBusy(i)$ evaluates false line 1), we define a local operator *minReady* (lines 2-4) that determines the *address* (*pc* field from the *program* input parameter) of the relevant instruction among the candidate instructions. If this instruction exists (0 is the conventional address of the *empty* instruction; penultimate line), we select (TLA+ CHOOSE operator) the *instruction* itself whose address has been determined by the local operator *minReady* (last line). *minReady* implements an age-ordered policy that selects the oldest instruction whose all dependencies are satisfied (or will be in the next cycle). It is based on the assumption that older, preceding instructions in the program order, have smaller addresses (see above). It is also based on the operator $FURout(i)$ (not detailed) providing the set of the currently decoded instructions (in the ID stage) that have *actually* been assigned the FU under consideration. This is trivial when the decoded instructions have only one admissible FU and it lies on an arbitrary choice otherwise. Consequently, *minReady* selects the smallest address (line 2), from the instructions waiting in the associated RS or directly from the ID stage[2] (line 3), more precisely only those (line 4) whose all dependencies (*dep* field assigned from the *program* input parameter) will have been computed.

The issued instructions are removed from the RSs in accordance, while the non-issued decoded instructions are added for a later selection, the whole through simple set-theory operators. Each entry of the $\_RS$ variable (one per RS/FU) is updated under this consideration:

$$\_RS' = [i \in (1 \,..\, N\_FU) \mapsto (\_RS[i] \cup FURouting(i))$$
$$\setminus \{NxtFU(i)\}]$$

Similarly, the $\_FU$ variable is updated using the $NxtFU(i)$ operator for each FU i.

C. Formal Verification of Timing Anomalies

As a case study, we introduce timing anomalies and show how to detect them in the context of a program execution on our OoO-pipeline template.

1) Timing Anomalies: Evaluating the timing behavior or predictability of CPSs is often based on the estimation of the worst-case execution time (WCET). Several methods allow to compute such a bound, e.g., testing-based methods [31], probabilistic methods [12] or static analysis [47]. In any case, these methods rely on specific assumptions addressing the absence of exhaustivity, which is impractical for complex designs. Certain execution phenomena, called Timing Anomalies (TAs), question those assumptions and may thus skew WCET analyses. As a consequence, it is essential to accurately detect the occurrences of TAs in the execution of an application on the target hardware architecture. Common OoO processors are known to present TAs [46]. We are to illustrate the detection of counter-intuitive TAs in our model, according to one family of formal definitions of TAs from the literature [18], [27], [11]. These definitions essentially consider that a TA occurs iff the (local) commit of a given instruction (A) is performed earlier in one execution behavior, say α (see Fig. 3), than in another behavior (β), whereas the commit of a later instruction in the program (D) is performed earlier in β than in α—in other words, a timing reversal in *commit* events.

[2]A decoded instruction is immediately issued to the FU if it is ready to execute and the related RS is empty (see ⋯► in Fig. 2).

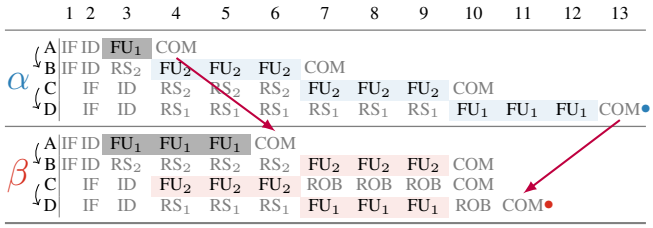


Figure 3: Execution traces showing the assignment to functional units (■/■/■) and a reversal of the order of commits (→) representing a counter-intuitive TA, obtained from the OoO template and the *program* of Fig. 2 (with $superscal = 2$ and $N_{\text{FU}} = 2$) [8].

2) Formalization of a TA Property: In order to address the verification/detection of TAs, we expand the template specification with a $comTime$ field in the ROB entries that keeps track of the instant ($currCycle$) of each commit event occurring during the execution. Besides, TAs are defined as a relation between *two* different executions of the same program. One behavior of the template specification is related to a *single* (arbitrary) execution of the program. We thus need to consider two behaviors at a time. To do so, we define a self-composition through two instances of the template specification in a main, higher-order specification module [7]. All state variables are duplicated and each instance manipulates its own set. Both instances share the input parameters, which guarantees that we consider the same program and the same version of the architecture datapath. However, they may progress at their own pace, according to their actual choices of FUs and latencies.

We endow the main specification module with a safety property related to the *absence* of TAs. We use the simplest form of safety properties, i.e., an invariant, in order to stipulate that (counter-intuitive) TAs may never occur while executing the input program on the considered version of the architecture. We may observe a TA only when *both* executions have completed, at least up to a certain instruction. The operator $ProgDone(n)$ (not detailed here) returns a Boolean indicating whether both executions have completed (at least) up to the n-th instruction of the input sequence. The operator $ComTime(ex, n)$ returns the value of the $comTime$ field for the n-th instruction of the (first or second) execution ex. Based on these operators, we now specify the property expressing the absence of TAs:

$$
\begin{aligned}
NoTA \triangleq\ & \forall\, k \in 1\,..\,Len(Program) - 1 : \\
& \quad \forall\, n \in k+1\,..\,Len(Program) : \\
& \quad \wedge\ ProgDone(n) \\
& \quad \wedge\ ComTime(1,\, k) < ComTime(2,\, k) \\
& \quad \Longrightarrow\ ComTime(1,\, n) \leq ComTime(2,\, n)
\end{aligned}
$$

The execution of the input sequence on the underlying architecture exhibits no TA iff, for any instruction k (line 1) and for any *subsequent* instruction n (line 2), it holds that if:
a) the execution is completed up to the considered instructions in both instances under consideration (line 3) and
b) the (local) commit ordering for instruction k is s.t. the first instance (e.g., α) precedes the second one (e.g., β) (line 4), then the commit ordering is the same for the subsequent instruction n (line 5). Note that both instances are totally interchangeable. That justifies the fact that we fix a priori the roles of each in the property, namely their commit ordering.

3) Verification Example: We verify the property $NoTA$ by model checking. Consequently, the violation of the property indicates the *existence* of a TA, and the provided counter-example is a TA scenario. Let us consider the verification with $superscal = 2$, $N_{\text{FU}} = 2$ and $program$ describing the input sequence of Fig. 2 annotated with the data dependencies (↓), the set of admissible FUs for the instructions (respectively, the singletons $\{1\}$, $\{2\}$, $\{2\}$ and $\{1\}$) and the possible latencies ($\{1\}$, i.e., no instruction cache miss,[3] for each instruction in the IF stage, and, respectively, $\{1, 3\}$, $\{3\}$, $\{3\}$ and $\{3\}$ in the FUs, where A experiences a variable latency). The model checker signals that the property is violated, thus indicating the *detection of a TA* while executing the input program on the OoO architecture. The reported counter-example is graphically represented here in Fig. 3.

IV. Abstract Processor Models for Fault Injections

In this section, we target the formal verification of security properties under Fault-Injection (FI) attacks.

A. Modeling Requirements for FI Security Assessment

FIs consist in applying an abnormal physical stress to the hardware to modify the behavior of the microelectronics. This leads to the appearance of incorrect values called *faults* in the micro-architecture as detailed by Yuce et al. [49]. These faulty values can be recovered or propagated through the processor circuit and lead to observable effects at the ISA or software level. For instance, this can result in the execution of a casual instruction, reading or writing to a wrong address in memory [34]. The observable effects of the faults can then be exploited by an adversary. To comprehensively locate and characterize FI-based vulnerabilities, developing a formal model of the processor helps to identify which FI attacks, targeting the hardware, defeat a given security requirement often expressed at the software level. Such a suitable processor model requires four elements that we now describe.
1) The *hardware model* of the processor provides a complete representation of both the combinatorial and sequential logics constituting a processor design.
2) The *software model* describes the sequence of instructions, represented in a binary form and performed on the hardware.
3) The *fault model* indicates how the physical attack interferes with the hardware model. A fault model has three dimensions: *temporal*, *spatial* and *effect*. The *temporal* dimension specifies the targeted clock cycles by the attack. The *spatial* dimension describes which signals can be modified by the fault. Finally, the *effect* dimension defines which values can be applied to

[3]Consider an instruction scratchpad for instance.

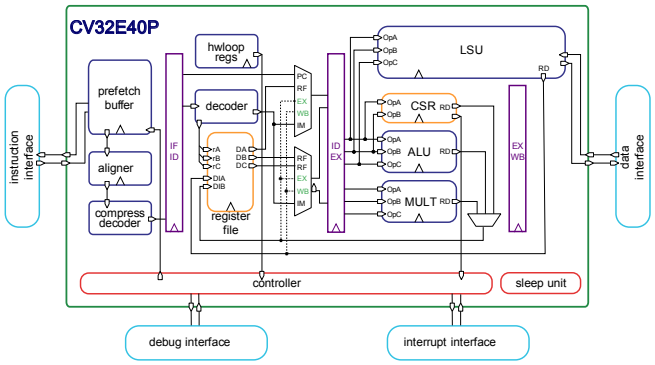


Figure 4: CV32E40P block diagram.

the faulty signals.

4) The *security property* is necessary to identify if a FI attack can lead to new vulnerabilities by encoding the expected software behavior.

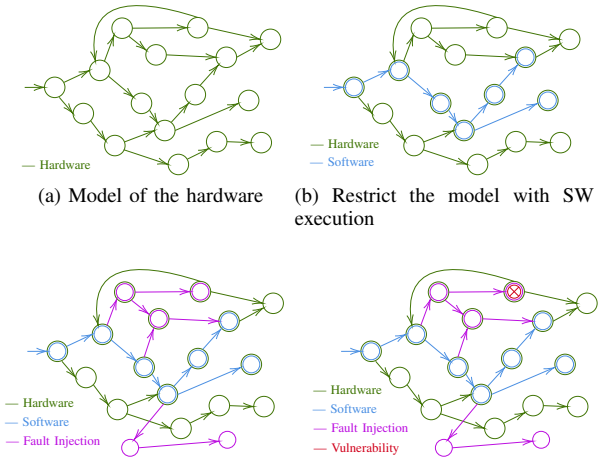
Compared to the modeling requirements described in Sec. III, the assessment of FI attacks requires not only a cycle-accurate but also a bit-accurate hardware model. This is necessary to accurately propagate the effects of the faulty signals into the hardware model. Note that various fault models can be assessed on formal models involving hardware, software and security properties. Hereafter, only transient faults that do not permanently damage the processor are considered—i.e., when the clock cycle leaves the temporal window specified in the fault model, the targeted signals are no longer under the control of the fault model.

B. FI-Assessment Process of an In-Order RISC-V Processor

We now illustrate how to implement these four elements over an in-order RISC-V processor and simplify them to formally verify FI-based vulnerability.

1) Hardware Design: The CV32E40P is a 32-bit processor intended for light and embedded use. It has a 4-stage pipeline (IF, ID, EX, WB) and the RTL implementation in the SystemVerilog language is provided by the OpenHW Group [39]. Fig. 4 shows the block diagram containing the main modules and their interconnections. From the hardware-design description, a formal Satisfiability Modulo Theories (SMT) model is produced by relying on the open-source synthesis tool Yosys [48]. This model is represented as a classical transition system and is illustrated (in green) in Fig. 5a. Each state of this transition system contains the values of the memory elements included in the micro-architecture while the transitions are governed by the combinatorial logic of the design.

2) Specified Software: Fig. 5b shows (in blue) how the specified program is used to restrict the model to some execution paths. Since the program behavior may depend on the input data, multiple execution paths can be explored in the model through the use of symbolic variables. Note that these unreachable states, when executing a non-faulty program, should not be removed as faults may have the effect to turn them into reachable states.



(a) Model of the hardware (b) Restrict the model with SW execution

(c) Add new states and transitions due to FI (d) Specify the vulnerability state

Figure 5: Modeling steps for FI vulnerability evaluation.

Listing 1: Example of a (bit-reset) fault model at RTL level.

```
1 state 10:90
2 assume [fw_mux] bit-reset
3 count 1
```

3) Fault Model: By injecting faults into the system, the program behaves differently over the micro-architecture. Such a fault injection is specified into the formal model by defining its temporal and spatial location and its effect. As an example, Listing 1 describes a bit-reset fault model, where the `fw_mux` bits are set to 0 (line 2) between cycles 10 and 90 (line 1). `count 1` (line 3) specifies that only one fault injection is allowed on this time interval. Fig. 5c illustrates the faulty behaviors by adding (purple) transitions into the model towards the (purple) states, which were previously not reachable by the program or not allowed by the hardware. This creates many additional execution possibilities that can be explored by a model checker.

4) Security Property: Security requirements can be defined by expressing a property on the states of the model. Fig. 5d illustrates this additional information by considering a (red) state to be faulty. A model checker is then used to verify if the property holds or is violated, and, in this latter case, counter-example(s) are provided. Thereafter, the security property is intended to control the instructions passing through the pipeline. This analysis ensures that secure data are never accessed or that a given instruction, whose semantics indicate successful authentication at the program level for instance, is never executed.

Next, we present several simplifications to efficiently handle the model that we have previously described.

a) Hardware Reduction: The security property only checks the values taken by a restricted number of state variables. The hardware model can thus be reduced by keeping only the necessary ones [15]. Whether variables are useful is determined iteratively by observing if they have an influence

on the property to be verified. When performed manually, this technique can also remove entire modules from a hardware design if they are not involved in the faulty behaviors of the program. Regarding the CV32E40P use case, the Control and Status Register (CSR) module—which manages privileged execution modes [45], performance statistics, and interruption and debugging mechanisms—can safely be removed as its behavior does not interfere with the specified security property. Table I shows the proportion of variables saved in the formal model of the CV32E40P processor when applying this optimization. Even if the CSR does not represent a large part of the netlist, its deletion considerably reduces the model size because of the memory features that it contains.

Variables	Netlist			Formal Model
	Wire bits	Logic cells	of which Flip-Flops	SMT-functions
Quantity	11903	374	18	110033
Proportion	6.6%	4.5%	10%	32%

Table I: Number of variables saved both at the RTL (Netlist) and formal model levels when removing the CSR module.

The model can be further simplified by adding constraints on the environment. The CV32E40P micro-architecture has memory elements (e.g., flip-flops providing a large range of initial states) that we can constrain. External modules can also interact with the processor via interrupts or debugging signals. Since these demands are not part of the model but can influence the security property, they must be controlled.

b) Software Reduction: At the software level, some instructions are not relevant regarding the security property and unused functions are embedded within the program due to the library linking process. We manually eliminated such dead code which allows us to considerably limit the size of the input program used for the software model. We restricted our models to execute an assembly program with 200 instructions—possibly with loops and unconstrained data input. This helps to reduce the number of states to be explored and their size in the formal model.

C. Formal Verification of FI-based Vulnerability

We now illustrate the capability of our abstract formal model to detect hardware vulnerabilities when verifying a specific security property. Listing 2 shows a security property that forbids the execution of the instruction, `0xfd5ff06f` in hexadecimal, allowing a secure authentication. The program does not normally allow this behavior, but it can be enabled by FIs.

Listing 2: An example of a security property.

```
assert (distinct [instr_id] 0xfd5ff06f)
```

Our fault model consists of a single bit-flip (bit-inversion) attack during one clock cycle of the program execution. We arbitrarily restrict the exploration of FI locations to the signals contained in the ID stage.

Model checking identifies several injection points on the micro-architecture, all due to the forwarding mechanism. Laurent et al. already point out [30] that it is possible to modify the forwarding control signal in order to recover values from EX and WB stages. These values can then be used as new operands, allowing, for example, to fool the comparison instructions. By modifying the forwarding behavior of the CV32E40P (depicted in Fig. 4 by a multiplexer in the ID stage) with a simple bit flip, the program execution reaches the vulnerable instruction indicated in Listing 2.

V. Towards Automated Extraction of Abstract Processor Models

The formal models described in the previous sections are all based on the accurate representation of specific details of the processor microarchitecture. Ideally, these models should be automatically derived from HDL processor designs. In this section, we present how we can take advantage of hardware compilation frameworks to ease the generation of abstract pipeline-stage-level models.

A. Hardware Compilation Framework: Chisel/FIRRTL

High-level design languages and the associated compilation chains enable the use of highly parameterized generators, domain-specific language constructs and advance module systems to facilitate the hardware design [6], [43], [40]. These compilation chains rely on several transformation passes to optimize the HDL (Verilog or VHDL) code that they generate for a later use as input in classical commercial (FPGA or ASIC) hardware-design flows. As in software compilers, hardware designers can also insert specific transformation passes in those compilation chains to manipulate the designs. This enables to deploy, within these chains, analyses to automatically construct abstract processor models. For instance, when targeting timing predictability, a transformation would mainly focus on the sequential logic to generate pipeline-level models, while for FI the whole design would initially be extracted, before abstractions are applied over the generated models.

We select the Chisel/FIRRTL hardware-compilation toolchain [6], [24] to illustrate this idea. Chisel (Constructing Hardware In a Scala Embedded Language) [6] is an open-source hardware-construction Domain-Specific Language (DSL) embedded in the Scala programming language. Adding hardware construction primitives to the Scala language allows the designers to write parameterized circuit generators, while using object-oriented and functional programming features to design circuits. Chisel emits synthesizable Verilog through an intermediate representation called FIRRTL [24], which stands for Flexible Intermediate Representation for RTL. Chisel thus constitutes the frontend part of the toolchain, FIRRTL, the middle-end from which all Scala-related hardware generators have been executed, and finally Verilog as a backend.

FIRRTL comes with different Intermediate Representations (IR), called forms. Each form uses a smaller, stricter and simpler subset of the Chisel language features and defines different transformations to generate the next (lower) form. Compiling FIRRTL to Verilog is implemented as a set of

passes that implement optimizations, such as constant folding or dead-code elimination. A so-called *high form* supports the Chisel high-level constructs such as vector types, bundle types, and conditional statements. These constructs are replaced by a set of low-level features, resembling a structured netlist that simplifies its translation to Verilog, in the lowest form named *low form*. Which one of these forms is the most appropriate one to generate abstract processor models? A lower form ensures an easier equivalence to the actual pipeline circuits and is thus mandatory for fault-injection assessment (Sec. IV), while a higher-level form facilitates the integration of complex properties such as timing predictability (Sec. III).

We now illustrate these differences in the high and low forms, through a very simple example presented by Listing 3. This example defines two registers, `reg1` and `reg2`, at lines 1 and 2. Both registers are initialized on reset using the `RegInit` construct. Note that the reset and clock signals are implicit in Chisel (but can be explicit in the FIRRTL forms). Finally, both registers are updated at lines 3 and 4, but for register `reg2` this update depends on the value of the `cond` variable (actual value not defined to simplify) and is thus performed within a Chisel `when` construct (line 4).

Listing 3: A simple Chisel code.

```
1 val reg1 = RegInit(0.U(4.W))
2 val reg2 = RegInit(0.U(4.W))
3 reg1 := value
4 when (cond) { reg2 := reg1 }
```

Listings 4 and 5 describe, respectively, the FIRRTL high form and low form obtained from the Chisel Listing 3.

Listing 4: FIRRTL high form from Listing 3.

```
1 reg reg1 : UInt<4>, clock with :
2    reset => (reset, UInt<4>("h0"))
3 reg reg2 : UInt<4>, clock with :
4    reset => (reset, UInt<4>("h0"))
5 reg1 <=value
6 when cond :
7    reg2 <=reg1
```

Listing 5: FIRRTL low form from Listing 3.

```
1 reg reg1 : UInt<4>, clock with :
2    reset => (UInt<1>("h0"), reg1)
3 reg reg2 : UInt<4>, clock with :
4    reset => (UInt<1>("h0"), reg2)
5 node _GEN_0 = mux(cond, reg1, reg2)
6 reg1 <=mux(reset, UInt<4>("h0"), value)
7 reg2 <=mux(reset, UInt<4>("h0"), _GEN_0)
```

It can be noticed that the `when` statement remains in the high form (line 6, Listing 4), while it is translated into a multiplexer in the low form (line 5, Listing 5). Note that in Chisel, a condition can be translated into a set of multiplexers to implement a multi-variable condition. Directly translating a high form into a formal-specification language may thus discard these multiplexers, which can be source points for a fault-injection attack as presented in Sec. IV. Such a statement can however be safely translated into a formal statement when targeting a pipeline-stage abstract model, such as the one shown for the detection of TAs (Sec. III). This demonstrates the need to develop custom FIRRTL passes aimed at automatically generating abstract models, against the targeted properties.

B. Extracting Abstract Pipeline Models

The Chisel/FIRRTL toolchain allows such an easy integration of a custom pass to take advantage of the different FIRRTL forms generated from a Chisel-based design. The FIRRTL design is internally represented with an Abstract Syntax Tree (AST) structure, where passes recursively visit nested elements to manipulate the AST. The FIRRTL AST consists of IR nodes represented by objects, each of which is a subclass of the following IR abstract classes: circuit, module, port, statement, expression or type. The registers and `when`-condition nodes, shown on the previous listings, are in the statement class. Each update of a register is represented by a connect node that is also part of the statement class.

We now present a custom pass that currently targets the automatic generation of abstract pipeline models specific to safety properties, as presented in Sec. III. This pass analyzes both the combinatorial and sequential logic of the pipeline datapaths in their high-form FIRRTL representations. It produces the pipeline stages, their order (thus forward edges between stages) and the backward edges between stages. Note that backward edges do not only correspond to the processor data-forwarding mechanisms between stages, e.g., classically from the write-back/memory to the execute stages, but also for instance simply the update of the program counter.

To achieve this, Chisel registers must first be identified at the FIRRTL level and their dependencies analyzed, by exploring their combinatorial input and outputs. Then, assigning a pipeline stage to the identified registers starts from an arbitrary specified[4] register, to be placed in the first pipeline stage. Two successive explorations of the set of identified registers are performed. The first exploration aims to assign pipeline stages based on two rules relying on register dependencies: 1) when only a single forward link between a source and a destination register exists, assign to the destination register the immediately following stage of the source register, and 2) when a destination register has several (already assigned) source registers, assign to the destination register the immediately following stage of the register having the minimal depth in the pipeline. This latter rule detects any forwarding mechanism within a pipeline. The second rule relies on a heuristic based on the idea that a designer simultaneously updates the registers of the same pipeline stage within a same conditional block.

C. Case Study: a RISC-V-based Processor

We now illustrate how our pass analyzes a RISC-V processor, called KyogenRV [42], so as to extract an abstract pipeline

[4]By the hardware designer for instance.

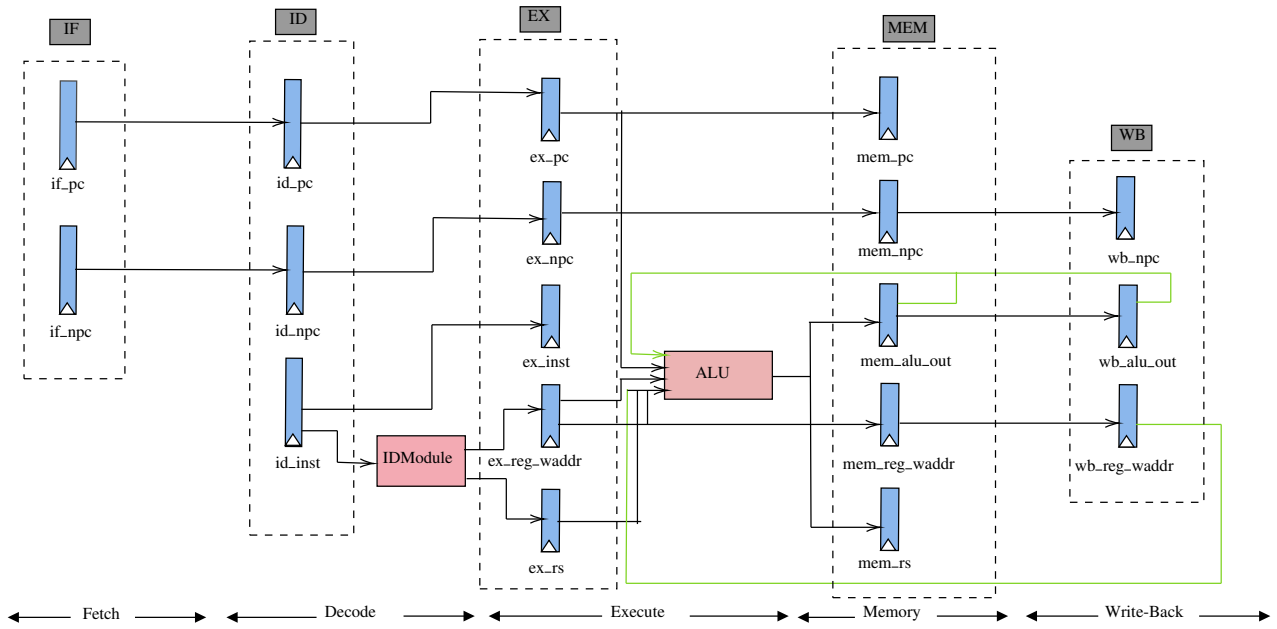


Figure 6: Representation of the extracted registers and (part) of the abstract pipeline model of the KyogenRV processor.

model. KyogenRV is an open-source 5-stage pipeline processor (IF, ID, EX, MEM, WB) targeting Intel FPGAs and developed for academic purposes. We focus on the top module of the pipeline, which has 60 registers. We specify the `if_pc` register to our pass as being located in the first stage of this pipeline. Our pass then automatically computes the dependencies between these registers, which are made of 46 edges, assigns a pipeline stage to each register, and outputs the abstract pipeline model. Fig. 6 shows a subset of the identified registers and their dependencies (omitting the combinatorial circuitry). Registers are represented by blue boxes, while red boxes represents Chisel instances of modules embedded within the currently analyzed Chisel module. Our pass correctly identifies the 5 stages, with forwarding mechanisms from the WB and the MEM to the EX stage. On the Fig. 6, the abstract pipeline model corresponds to the dashed boxes only, one for each stage, and associated edges are not shown for readability reason. It is thus similar to the one used for the detection of TAs (Sec. III) with: 1) a single forward edge kept between each stage, and 2) the green edges (potentially merged) implementing the forwarding mechanism corresponding to the dashed edge (- -▸) of Fig. 2, which enables a back-to-back execution of instructions over the FUs.

Listing 6 presents a subset of the data forwarding implemented from the WB and MEM towards the EX stage. The Chisel wire `ex_reg_rs1_bypass` is updated from the output of the `mem_alu_out` register (line 5), located in the MEM stage, or from the `wb_alu_out` register (line 6), located in the WB stage, through the Chisel MuxCase construction. In the FIRRTL low form the MuxCase is translated into a cascade of multiplexers. This forwarding corresponds to the green arrows, shown in Fig. 6, from the registers `wb_alu_out` and `mem_alu_out` to the input of the ALU red box. The other green arrow, between the register `wb_reg_waddr` and the input of the ALU, is part of the check to detect the need for forwarding a value (not shown in Listing 6).

Listing 6: Forwarding in KyogenRV 5-stage.

```
1  val ex_reg_rs1_bypass = Wire(UInt(32.W))
2
3  /* Ci, i = 1..2 - conjuncts of write enable
       ↪ and selection signals */
4  ex_reg_rs1_bypass := MuxCase(ex_rs(0), Seq(
5    (ex_reg_raddr(0) === mem_reg_waddr && C1)
       ↪ → mem_alu_out
6    (ex_reg_raddr(0) === wb_reg_waddr && C2)
       ↪ → wb_alu_out))
7  ...
8  when (C4 /* no stalling condition */) {
9    mem_rs(0) := ex_reg_rs1_bypass
10 }
```

VI. Conclusion and Future Work

We have presented the various needs in the formal modeling of pipeline processors to verify both safety and security properties of CPS or IoT systems. The safety property that we consider (detection of timing anomalies) requires an abstract pipeline model where only stages are visible and whose timing behavior is accurate, while the security property (identification of fault-injection points) requires a cycle- and bit-accurate model where unnecessary either software or hardware parts of the considered system have been removed. Finally, we have shown how a high-level hardware compilation toolchain can ease the adaptation of automatically generated formal abstract models to such safety and security properties. We

have reported on a custom pass to generate an abstract pipeline model (and to be used for the detection of timing anomalies).

As future work, we are currently investigating how to expand the definition of timing anomalies to other hardware resources such as caches or speculation mechanisms and verify the needs in formal modeling of these elements. We also plan to implement various abstraction strategies of the formal models to speedup the fault-injection assessment. Finally, we plan to expand our pass to be able to generate the different formal models needed for both the safety and security properties.

Acknowledgments. The authors would like to thank Claire Pagetti from ONERA and the anonymous reviewers for providing valuable feedback that helped us improve this work.

REFERENCES

[1] Armstrong, A., Bauereiss, T., Campbell, B., Reid, A., Gray, K.E., Norton, R.M., Mundkur, P., Wassell, M., French, J., Pulte, C., Flur, S., Stark, I., Krishnaswami, N., Sewell, P.: ISA semantics for armv8-a, risc-v, and CHERI-MIPS. ACM Program. Lang. **3**, 71:1–71:31 (2019)

[2] Asanović, K., Patterson, D.A.: Instruction sets should be free: The case for risc-v. Tech. Rep. UCB/EECS-2014-146 (Aug 2014)

[3] Asavoae, M., Haur, I., Jan, M., Hedia, B.B., Schoeberl, M.: Towards formal co-validation of hardware and software timing models of cpss. In: CyPhy/WESE. LNCS, vol. 11971, pp. 203–227 (2019)

[4] Asavoae, M., Hedia, B.B., Jan, M.: Formal Executable Models for Automatic Detection of Timing Anomalies. In: WCET (2018)

[5] Asavoae, M., Jan, M., Ben Hedia, B.: Formal modeling and verification for timing predictability. In: ERTS (2020)

[6] Bachrach, J., Vo, H., Richards, B.C., Lee, Y., Waterman, A., Avizienis, R., Wawrzynek, J., Asanovic, K.: Chisel: constructing hardware in a scala embedded language. In: DAC'12. pp. 1216–1225 (2012)

[7] Barthe, G., D'Argenio, P., Rezk, T.: Secure information flow by self-composition. In: CSF. pp. 100–114 (2004)

[8] Binder, B., Asavoae, M., Ben Hedia, B., Brandner, F., Jan, M.: Is this still normal? Putting definitions of timing anomalies to the test. In: RTCSA (2021)

[9] Binder, B., Asavoae, M., Brandner, F., Hedia, B.B., Jan, M.: Scalable detection of amplification timing anomalies for the superscalar tricore architecture. In: FMICS (2020)

[10] Bréjon, J.B., Heydemann, K., Encrenaz, E., Meunier, Q., Vu, S.T.: Fault attack vulnerability assessment of binary code. In: CS2 Workshop. pp. 13–18 (2019)

[11] Cassez, F., Hansen, R.R., Olesen, M.C.: What is a Timing Anomaly? In: WCET. vol. 23, pp. 1–12 (2012)

[12] Cazorla, F.J., Kosmidis, L., Mezzetti, E., Hernandez, C., Abella, J., Vardanega, T.: Probabilistic worst-case timing analysis: Taxonomy and comprehensive survey. ACM Comput. Surv. **52**(1) (Feb 2019)

[13] Cerqueira, F., Stutz, F., Brandenburg, B.B.: PROSA: A case for readable mechanized schedulability analysis. In: ECRTS. pp. 273–284 (2016)

[14] Choi, J., Vijayaraghavan, M., Sherman, B., Chlipala, A., Arvind: Kami: a platform for high-level parametric hardware specification and its modular verification. ACM Program. Lang. **1**, 24:1–24:30 (2017)

[15] Clarke, E.M., Kurshan, R.P., Veith, H.: The localization reduction and counterexample-guided abstraction refinement. In: Time for verification, pp. 61–71. Springer (2010)

[16] Cuenot, P., Delmas, K., Pagetti, C.: Multi-core processor: Stepping inside the box. In: ESREL 2021. Angers, France (2021)

[17] Dalsgaard, A., Olesen, M., Toft, M., Hansen, R., Larsen, K.: Metamoc: Modular execution time analysis using model checking. In: WCET. vol. 15, pp. 113–123 (2010)

[18] Eisinger, J., Polian, I., Becker, B., Thesing, S., Wilhelm, R., Metzner, A.: Automatic identification of timing anomalies for cycle-accurate worst-case execution time analysis. In: DDECS. pp. 13–18 (2006)

[19] Given-Wilson, T., Jafri, N., Lanet, J.L., Legay, A.: An Automated Formal Process for Detecting Fault Injection Vulnerabilities in Binaries and Case Study on PRESENT. In: Trustcom. pp. 293–300 (2017)

[20] Given-Wilson, T., Jafri, N., Legay, A.: Combined software and hardware fault injection vulnerability detection. Innovations in Systems and Software Engineering **16**(2), 101–120 (Jun 2020)

[21] Hahn, S., Reineke, J.: Design and analysis of sic: A provably timing-predictable pipelined processor core. In: RTSS. pp. 469–481 (2018)

[22] Hicks, M., Sturton, C., King, S.T., Smith, J.M.: Specs: A lightweight runtime mechanism for protecting software from security-critical processor bugs. SIGPLAN Not. **50**(4), 517–529

[23] Irfan, A., Cimatti, A., Griggio, A., Roveri, M., Sebastiani, R.: Verilog2SMV: A tool for word-level verification. In: DATE. pp. 1156–1159 (2016)

[24] Izraelevitz, A., Koenig, J., Li, P., Lin, R., Wang, A., Magyar, A., Kim, D., Schmidt, C., Markley, C., Lawson, J., Bachrach, J.: Reusability is firrtl ground: Hardware construction languages, compiler frameworks, and transformations. In: ICCAD. p. 209–216 (2017)

[25] Jain, H., Kroening, D., Sharygina, N., Clarke, E.: VCEGAR: Verilog CounterExample Guided Abstraction Refinement. In: TACAS. pp. 583–586 (2007)

[26] Jan, M., Asavoae, M., Schoeberl, M., Lee, E.A.: Formal semantics of predictable pipelines: a comparative study. In: ASP-DAC (2020)

[27] Kirner, R., Kadlec, A., Puschner, P.: Worst-case execution time analysis for processors showing timing anomalies. Tech. rep., Technische Universität Wien (01 2009)

[28] Kroening, D., Purandare, M.: Ebmc. http://www.cprover.org/ebmc/

[29] Lamport, L.: Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers. Addison-Wesley Longman Publishing Co., Inc., USA (2002)

[30] Laurent, J., Beroulle, V., Deleuze, C., Pebay-Peyroula, F.: Fault Injection on Hidden Registers in a RISC-V Rocket Processor and Software Countermeasures. In: DATE. pp. 252–255. Florence, Italy (2019)

[31] Law, S., Bate, I.: Achieving appropriate test coverage for reliable measurement-based timing analysis. In: ECRTS. pp. 189–199 (2016)

[32] Lee, S., Sakallah, K.A.: Unbounded scalable verification based on approximate property-directed reachability and datapath abstraction. In: CAV. pp. 849–865 (2014)

[33] Li, X., Roychoudhury, A., Mitra, T.: Modeling out-of-order processors for wcet analysis. Real-Time Systems **34**, 195–227 (11 2006)

[34] Library, J.I.: Application of Attack Potential to Smartcards and Similar Devices. Tech. rep. (2013)

[35] Mangean, A., Béchennec, J.L., Briday, M., Faucou, S.: Wcet analysis by model checking for a processor with dynamic branch prediction. In: VECoS. pp. 64–78 (2017)

[36] McMillan, K.: Cadence smv. Cadence Berkeley Labs, CA, 2000

[37] Mukherjee, R., Purandare, M., Polig, R., Kroening, D.: Formal techniques for effective co-verification of hardware/software co-designs. In: Proc. of the 54th Annual Design Automation Conference 2017. pp. 1–6

[38] Nienhuis, K., Joannou, A., Bauereiss, T., Fox, A., Roe, M., Campbell, B., Naylor, M., Norton, R.M., Moore, S.W., Neumann, P.G., Stark, I., Watson, R.N.M., Sewell, P.: Rigorous engineering for hardware security: Formal modelling and proof in the CHERI design and implementation process. In: 2020 IEEE Symposium on Security and Privacy

[39] OpenHW Group: Core-v cv32e40p risc-v ip. https://github.com/openhwgroup/cv32e40p

[40] Papon, C.: SpinalHDL. https://github.com/SpinalHDL

[41] Pattabiraman, K., Nakka, N., Kalbarczyk, Z., Iyer, R.: SymPLFIED: Symbolic program-level fault injection and error detection framework. In: Intl. Conf. on DSN. pp. 472–481 (2008)

[42] Saitoh, A.: Kyogenrv: simple 5-staged pipeline RISC-V. https://github.com/panda5mt/KyogenRV

[43] Schuiki, F., Kurth, A., Grosser, T., Benini, L.: LLHD: a multi-level intermediate representation for hardware description languages. In: Donaldson, A.F., Torlak, E. (eds.) PLDI. pp. 258–271 (2020)

[44] Tomasulo, R.M.: An efficient algorithm for exploiting multiple arithmetic units. IBM J. of Research and Development **11**(1), 25–33 (1967)

[45] Waterman, A., Asanovic, K., Hauser, J., Division, C.: Volume II: Privileged Architecture. Tech. rep. (2021)

[46] Wenzel, I., Kirner, R., Puschner, P., Rieder, B.: Principles of timing anomalies in superscalar processors. In: QSIC. pp. 295–303 (2005)

[47] Wilhelm, R., Engblom, J., Ermedahl, A., Holsti, N., Thesing, S., Whalley, D., Bernat, G., Ferdinand, C., Heckmann, R., Mitra, T., Mueller, F., Puaut, I., Puschner, P., Staschulat, J., Stenström, P.: The worst-case execution-time problem—overview of methods and survey of tools. ACM Trans. Embed. Comput. Syst. **7**(3) (May 2008)

[48] Wolf, C.: Yosys open synthesis suite. http://www.clifford.at/yosys/

[49] Yuce, B., Schaumont, P., Witteman, M.: Fault Attacks on Secure Embedded Software: Threats, Design, and Evaluation. Journal of Hardware and Systems Security **2**(2), 111–130 (2018)

An Automated Framework Towards Widespread Formal Verification of Complex Hardware Designs

Jonathan CERTES, Benoît MORGAN
IRIT-ENSEEIHT, University of Toulouse
email : *firstname.lastname*@irit.fr

***Abstract*—Verification is an essential step of critical systems design flow with regard to safety and security. It supports respectively fault and vulnerability removal. *Model-checking* ensures that a design meets its specifications by using an exhaustive state exploration approach. It has been adopted to design critical and/or secure by design embedded hardware systems. On the one hand, *model-checking* is fully automated; on the other hand, it does not scale and faces state-space explosion when working with large industrial circuits and complex specifications. Compositional *model-checking* and theorem proving enable to verify large designs at the cost of finding abstractions and proving an implication of the specifications. In this paper, we present a method along with a framework to reduce this cost and improve hardware verification performances.**

We formally verified a hardware security monitor involved in a remote attestation scheme for microprocessors. This verification could not be achieved using classical approaches as it faces state-space explosion: the monitor is complex enough to be unfit for *model-checking*. So, we applied our method to the verification and successfully proved the security of remote attestation using symbolic *model-checking* and automatic theorem proving. Our verified security monitor is described with an hardware description language and its specifications are written in property specification language. Our method makes extensive use of compositional *model-checking* techniques to leverage the modular and partitioned aspects of most automata involved in hardware modelling. This method is fully automated in a framework build on top of free model checkers and automatic theorem provers. Automation relies on synthesis and translation tools which exploit the modular structure of sequential circuits and avoid having to re-design.

***Index Terms*—Automation, formal verification, security, FPGA**

I. Introduction

Verification of hardware designs is traditionally performed through simulation and testing. Selected input vectors are fed into the system and outputs are checked for correctness. The drawback of these traditional verification techniques is that they aim at tracing the most potential defects and suffer from incompleteness.

Another approach, adopted to design critical and/or secure hardware systems [1], [2], relies on formal methods to ensure that the design meets its specifications. Formal verification is generally conducted in three steps. First, the system (hardware or software) is modeled, for example as an automaton. Then, properties to be satisfied by the system are formally expressed. Eventually, *model-checking* and/or proof is used against the model to demonstrate that it satisfies the required properties.

The major obstacle for widespread application of *model-checking* to real-world designs is called the state-space explosion problem [3]: when the number of states needed to model the system accurately turns out to exceed the physical limits of the computer memory. One of the most successful ways to cope with this problem is to use abstract models and check the property on them instead of the original model. However, over-approximation, as a consequence of abstraction, turns out to generate false negatives [4]. Finding satisfactory abstractions, which reduce the model enough and do not lead to false negatives, is non-trivial. Compositional *model-checking* techniques, such as *localization reduction* and *partitioned transition relations*, guarantee the absence of false negatives [5]. These techniques are particularly adapted to abstract hardware designs as partitioned structure of sequential circuits ensures no interdependence between processes.

In this paper we propose a mostly fully automated framework to improve hardware verification performances and efficiently remove the presence of faults (consequently vulnerabilities) in realistic hardware designs. Our method is based on a specific case of compositional *model-checking* and exploits the modular and partitioned structure of sequential circuits. Then, we extensively present how we applied this approach to formally verify a hardware security monitor for remote attestation of microprocessor software [6]. One of the objectives of this monitor is to preserve the confidentiality of a secret key used to compute a HMAC. It relies on a complex trace decompresser for ARM *CoreSight* debug interface [7] whose verification is the cornerstone of overall system security. Our main objective is to prove the security for remote attestation of microprocessor software from locally verified properties.

A lot of work has been dedicated to efficient and/or automatic abstraction for Register Transfer Level (RTL) *Verilog*, either with a smart predicate partitioning through slicing of bit vectors [8], [9], using uninterpreted functions abstraction [10] or by applying algebraic rewriting to extract arithmetic functions [11]. The authors automatically provide sound partitions for the system to increase compositional *model-checking* efficiency. Unfortunately, writing consistent properties to be verified regarding the content of the generated abstract model is left to the designer's expertise. This process is inconvenient since the main goal is to verify overall specifications (i.e. formal definition of security), potentially expressed with signals or memories that have been removed or split/renamed in the generated abstract models. Also, local properties are

then to be expressed with generated names while making sure that their conjunction still implies the overall specifications. A more convenient approach, that we extensively use in our method, is to write local properties with a proof deduction in mind and then find abstract models that are adapted to their verification.

For a wide adoption of formal methods to hardware verification, expressing specifications and dividing them into local properties must be convenient and theorem proving must be automated. As a consequence, automatic theorem proving must be conducted on highly expressive temporal logics. Moreover, abstraction that is adapted to the verification of these local properties must be automated too. To answer the aforementioned bottleneck and challenges, we introduce in this paper two main contributions:

1) an automated translation of Property Specification Language (PSL) [12], dedicated to automatic theorem proving. The tool also supports uninterpreted functions abstraction, if needed, in order to relieve deductive proof systems in solving the satisfiability problem;
2) an automated framework that computes, for complex hardware designs, adapted abstract models from temporal properties, verifies their soundness and feeds *model-checking* software.

Source code for algorithms that rewrite PSL and verifies the soundness of abstract models is publicly available at [13].

This article is organized as follows: state of the art is given in Section II. Section III details our automated verification framework. The case study is described in Section IV and Section V shows the application of our framework. Eventually, limitations are listed in Section VI.

II. State of the Art

In this section we provide background on modeling complex hardware designs and formal verification.

A. Modeling of complex hardware designs

Automatic translation tool *Verilog2SMV* [14] converts *Verilog* to *SMV* language, which supports similar high-level types of state variables as Hardware Description Languages (HDL) for wires and registers. In particular, *Verilog2SMV* aims at handling designs with memories efficiently [14]. *Verilog2SMV* is built on top of *Yosys* [15], a free *Verilog* synthesis suite. It first flattens the *Verilog* high-level design, synthesizes the RTL circuit while providing optimizations and then translates the output into a corresponding *SMV* model.

Tuerk et al. have formally validated the correctness of a translation from PSL to Linear Temporal Logic (LTL) [16], they produced a *model-checking* infrastructure that works by translating *model-checking* problems to equivalent checks for the existence of fair paths through a Kripke structure [17].

B. Formal verification

Deterministic Büchi automata are specific ω-automata that can accept infinite words. More especially, a word is accepted if and only if the automaton goes infinitely often through accepting states, called acceptance set. A deterministic Büchi automaton can be defined as a tuple $A = \langle \mathcal{Q}, \Sigma, \Delta, I, \mathcal{F} \rangle$ where:

- $\mathcal{Q}$ is a finite set of states
- Σ is an alphabet
- $\Delta : \mathcal{Q} \times \Sigma \to \mathcal{Q}$ is a transition function
- $I \subseteq \mathcal{Q}$ is a set of initial states
- $\mathcal{F} \subseteq \mathcal{Q}$ is an acceptance set

Deterministic Büchi automata are used to model finite state machines and formulae in temporal logics [18]. *Model-checking* algorithms manipulate them according to the automata theory approach.

NuSMV is an open-source model checker which implements *symbolic model-checking*, using a fixed point algorithm with *Binary Decision Diagrams* (BDD), where a set of states is represented by a BDD instead of an exhaustive list of individual states [19]. Models are described in SMV language and *NuSMV* supports the analysis of specifications expressed as invariants or in temporal logics, including LTL and PSL.

Duret-Lutz et al. presented *Spot*, a C++ library with Python bindings designed to manipulate LTL and ω-automata [20]. *Spot* contains algorithms to perform the usual tasks in *model-checking*, including filtering, conversion and transformation for LTL formulae and Büchi automata.

As a response to the state-space explosion problem in *model-checking*, R. Kurshan, E. Clarke and H. Veieth formalized the most commonly used abstraction techniques [4]: *localization reduction* abstracts models by hiding variables that are not referenced in the verified property; *predicate abstraction* is an over-approximation technique which may produce spurious counterexamples; *counterexample-guided abstraction refinement* is an iterative process to create new abstract models and checks spurious counterexamples on the concrete model until the checked property is either proved or disproved. Berezin et al. describe the rules to follow to ensure the soundness of compositional *model-checking* techniques, including with *partitioned transition relations* [5], where the global transition relation of a system is written as the conjunction or disjunction of transition relations for the individual components of this system. Also, R. Milner introduced the concept of simulation between automata [21]. Bensalem et al. demonstrated the preservation of properties in the case of simulations parameterized by a Galois connection [22]: establishing a simulation relation between systems, which is straightforward in the case of compositions, allows to share a proof that a property is verified.

III. Automated verification framework

In our framework, overall specifications are formally expressed with PSL and local properties for compositional *model-checking* are elaborated manually to be sufficient to imply the specifications. All PSL properties are translated into LTL by model checker *NuSMV*, which is considered correct. Nevertheless, it is possible to also rely on proven translation works [16], which preserve PSL semantics. Then, *Spot* is

leveraged to prove the implication of the overall specifications from the conjunction of the local properties.

The hardware design is described in *Verilog*, it is synthesized and translated using *Verilog2SMV* into a Büchi automaton, which model is described in *SMV* language. To avoid state-space explosion, we propose a method to automatically generate abstract models, with *localization reduction*, for each PSL property to verify. Property preserving simulation relations are established between the abstract models and the concrete model: this provides a certificate that the abstractions are sound. Then, PSL properties are verified locally with model checker *NuSMV*.

Our approach differs from previous works as abstract models are deduced from the properties to verify. We believe that writing local properties with a proof deduction in mind is key to proving the implication of complex specifications, even if it implies a non-optimal reduction for the abstractions. Furthermore, our framework is automated and relies on synthesis tools so that the generated models are close to the final implementation of the system.

Several Electronic Design Automation (EDA) suite are available in the industry, most of which providing solutions for synthesis and assertion based formal verification. The approach we propose is agnostic to the choice of an EDA toolchain: it can be applied to any of these suite as soon as the dedicated tools can be instrumented with a high level of granularity. For this reason, we instantiated it in a framework which relies on open-source software.

Now, we present our automated framework, depicted in Figure 1, which follows those three major steps:

- Step 1: as opposed to [8], [9], [10], [11], we start by rewriting the specifications in local properties. We take great care in ensuring that the conjunction those local properties is sufficient to imply the specifications.
- Step 2: once we have the local properties, we prove that their conjunction imply the specifications, i.e. when this implication is a tautology. To do so, we rely on *Spot* which provides filtering algorithms for LTL properties [20].
- Step 3: finally, we can move on to the automatic abstraction of the system using the local properties. Hopefully, those local properties are shrunk enough to generate *localization reductions* of the system that can be model checked in a reasonable amount of time. *Model-checking* with *NuSMV* ensures the verification of the local properties.

A. Step 1: Formal expression of specifications and local properties with PSL

In our framework, we take advantage of extended next operators, a subset of PSL that can be seen as syntactic sugar for LTL. This subset is supported by *NuSMV* for *model-checking* [23]. PSL can be described as follows:

In addition to propositional operators, such as conjunction ($\wedge$), disjunction ($\vee$), negation ($\neg$), and implication ($\rightarrow$), PSL features temporal operators along with replicators, which are

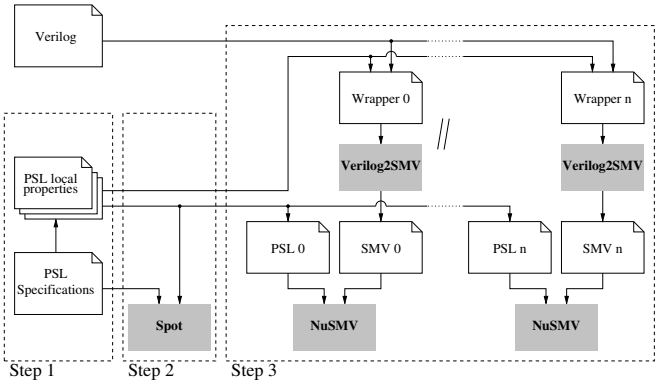


Fig. 1. Automated verification framework

quantification operators. The following operators find their equivalent in LTL, they are simple but of interest to express our specifications:

- $\mathbf{always}(\phi)$: holds if property ϕ is true for all future states;
- $\mathbf{next}(\phi)$: holds if ϕ is true at the next system state;
- (ψ) $\mathbf{until}$ (ϕ): holds if there is a future state where ϕ is true and ψ is true for all states before that.

In addition to these basic temporal operators, PSL also offers extended next operators, including:

- $\mathbf{next\_event\_a}(\psi)[range](\phi)$: holds if ϕ is true at all the next states where ψ is true in the range defined by *range* (where a range is a set of consecutive integer numbers);
- $\mathbf{next\_event}(\psi)(\phi)$: this is a shorthand for $\mathbf{next\_event\_a}(\psi)[1:1](\phi)$.

The last interesting operator is the **forall** replicator:

- $\mathbf{forall}\ i\ \mathbf{in}\ \{range\} : \phi(i)$: holds if the conjunction of parameterized sub-properties $\phi(i)$ is true for all possible values of identifier i in the range defined by *range*.

Specifications are formally expressed with PSL and local properties are elaborated while trying to keep their conjunction sufficient for the implication. These tasks rely on the designer's expertise in logics. This is the only part of our framework which is manual.

B. Step 2: Proof strategy

Implication of the specifications is obtained through automatic theorem proving using *Spot*. To achieve this, we have instrumented parsing and translation functions from *NuSMV* in order to convert PSL, which is not entirely supported by *Spot*, to LTL. As a consequence, extended next operators are only expressed with basic temporal operators; for instance, operator $\mathbf{next\_event}$ is expressed as:

$$\mathbf{next\_event}(a)(b) \equiv (\neg(a)\ \ \mathbf{until}\ \ (a \wedge b)) \vee \mathbf{always}\,(\neg(a))$$

After the translation, we generate the following formula from a conjunction of the local properties and an implication of the specifications:

$$(\mathit{property}_0) \wedge (\mathit{property}_1) \wedge \ldots \rightarrow (\mathit{specifications})$$

We rely on filtering functions from *Spot* [20] to automatically process this formula:

- if the filtered formula is a tautology, then specifications are implied;
- if the filtered formula is a temporal expression, then the conjunction of local properties is not sufficient to imply the specifications;
- if the physical limits of the computer memory are reached, we cannot conclude.

Despite being automatic, proving the implication consists in solving a satisfiability problem which suffers from state-space explosion just like *model-checking*. To tackle this problem, we provide uninterpreted functions abstractions to reduce the proof effort:

- logic operations between bit vectors are abstracted into booleans. For instance, equality between two vectors a and b is only considered as either true or false to prove the implication, regardless of the vectors size;
- **forall** replicators are not expanded into a conjunction of sub-properties: only replicated sub-properties (i.e. expressions using the identifier) are considered. This greatly reduces the complexity of the proof but comes at the cost of manually making sure that all possible values are verified through *model-checking*.

For example, the following expression gives a property that has been verified through *model-checking*:

$$\mathbf{forall}\ i\ \mathbf{in}\ \{0:255\}:\ \mathbf{always}\{(a=i)\rightarrow \mathbf{next}(b=i)\}$$

Our abstractions reduce it to boolean "$(a = i)$" always implying boolean "$(b = i)$" at the next state. Since sizes for vectors a and b do not appear in the expression, it is the responsibility of the designer to ensure that *model-checking* with range $\{0 : 255\}$ for identifier i covers all possible values for both a and b. In a case where the provided abstractions are still not sufficient to run the proof, rewriting local properties from *Step 1* or separating the proof into several steps is required.

Our translation and abstraction algorithm is available at [13]. It has been implemented using *pyNUSMV* [24], a Python framework for prototyping and experimenting with BDD-based *model-checking* algorithms from *NuSMV*.

C. Step 3: Automatic localization reduction and model-checking

A model of the hardware design is described in *Verilog* at RTL. To automatically generate abstract models dedicated to the verification of the properties, we rely on hypothesis **H0**, defined as follows:

H0: the concrete model is composed of multiple finite systems running in parallel. When outputs are unused, optimizations step of synthesis separates them, removes unused registers and leaves the useful parts of the system untouched.

We take advantage of synthesis optimizations step to create *localization reductions* of the model: outputs are left unconnected and all state variables from the model that are irrelevant in verifying the property are abstracted. With **H0**, we assume that the useful parts of the system are left untouched and that there is a simulation relation parameterized by a Galois connection between the concrete model and the generated abstract models.

Since **H0** is a hypothesis, we conduct an *a posteriori* verification and provide a certificate that the abstract model is simulated by the concrete model. Verified properties are then preserved. This task is embedded in our framework and is automatically applied to all generated abstract models. The strength of this approach is that it makes the verification process possible for any synthesis optimizations algorithm and configuration as soon as our certificate guarantees its soundness. The only restriction is to use the same algorithm and configuration for converting both the concrete model and its abstract model.

Regarding our framework, we proceed as follows:

- a *Verilog* wrapper is automatically created where all the module outputs which do not appear in the property are left unconnected. This is achieved using the Verilog Procedural Interface of *Icarus Verilog* [25].
- *Verilog2SMV* synthesizes and translates the *Verilog* model, extended with the previously generated wrapper, and converts it to SMV. Optimizations step from *Yosys* proceeds to a *localization reduction* of the model which is dedicated to verify the property.
- Verification of replicated properties is split into the verification of several non-replicated sub-properties, one for each value of the replicator. This reduces the cost of *model-checking* as size of the BDD grows exponentially with the complexity of the property.
- *NuSMV* is used to verify that the property holds on its dedicated abstract model. If the property does not hold, *NuSMV* generates a counterexample which is converted into Value Change Dump (VCD) format.

This operation is repeated for each property as depicted in sub-graph "Step 3" from Figure 1.

Certificate of soundness

We establish a simulation relation between the concrete model and the generated abstract model, this is performed by comparing transition functions of Büchi automata. To understand the verification of soundness, we first need to introduce some concepts regarding SMV models.

A SMV model is defined by input variables, state variables, initial values and transition functions [14], [19]:

- SMV input variables represent the inputs of the hardware circuit; all possible tuples for SMV input variables values represent the alphabet (Σ) of the automaton.
- SMV state variables represent *Verilog* registers as either single bits, bit vectors or multidimensional arrays; all possible tuples for SMV state variables values represent the finite set of states ($\mathcal{Q}$) of the automaton.
- Their possible initial values are set as a result of the translation from *Verilog initial* directive; initial states (I)

of the automaton are the product of initial values for all SMV state variables.
- Finally, SMV transition functions determine, for each SMV state variable, which SMV state the variable should be set to from its current state and SMV input variables; the global transition function (Δ) of the Büchi automaton is also the result of a product between SMV transition functions for all SMV state variables.

There is no acceptance set ($\mathcal{F}$) in the SMV model since it is a consequence of having a property to be verified: final states are, in *model-checking*, states that satisfy the negation of the property on the product of both the model and the automaton generated from the property.

SMV models are then represented by a conjunctive partitioned transition relation [5]: each partition (δ) of the transition function (Δ) for the model is then a transition function for one SMV state variable. To establish a simulation relation, we verify that all partitions of the transition function and initial states from the abstract model are identical in the concrete model.

Algorithm 1 Comparison of transition functions

Input: C: concrete model, A: abstract model
Output: $boolean$: soundness of the abstraction

```
1: [Δ_C, I_C] = parse(C)
2: [Δ_A, I_A] = parse(A)
3: normalize(Δ_C); normalize(I_C)
4: normalize(Δ_A); normalize(I_A)
5: for all δ_A in partition(Δ_A) do
6:    if not (∃δ_C ∈ partition(Δ_C)|δ_C ≡ δ_A) then
7:       return False
8:    end if
9: end for
10: for all q_A in partition(I_A) do
11:    if not (∃q_C ∈ partition(I_C)|q_C ≡ q_A) then
12:       return False
13:    end if
14: end for
15: return True
```

Algorithm 1 presents how the comparison of transition functions is performed. It requires two models generated from the RTL description of the circuit: the concrete model (C) and the abstract model (A), where both are generated using *Verilog2SMV* with the same synthesis optimization algorithm and configurations. It returns a boolean giving the soundness of the abstraction. This returned value is true if the expression of all transition functions and initial states from the abstract model exist with the same expression in the concrete model.

- First, it parses both the concrete model and the abstract model to extract their global transition function (Δ_C, Δ_A) and sets of initial states (I_C, I_A). As explained earlier, global transition functions of both automata are the results of a product between several transition functions (δ_C, δ_A) from all state variables. Respectively, global initial states of both automata are the results of a product between several initial states (q_C, q_A) from the same state variables.
- Then, a normalization is performed so that descriptions of initial states and transition functions are only expressed with members of the alphabet (Σ) and other states variables ($\mathcal{Q}$). This step is important since different circuits have been processed by the optimization algorithm and descriptions may not result from the same wiring (*Verilog2SMV* preserves the hierarchy of the circuits by expressing wires through the definition of symbols [14]; initial states and transition functions are expressed with these symbols). Normalization allows to abstract the wiring of the circuit as these are not memories and do not alter the BDD when *model-checking*.

An implementation of Algorithm 1 is available at [13]. Equivalence of partitioned transition functions is verified through a comparison at syntactic level. This is justified by the fact that many abstract models have to be checked for soundness (one per property) for a single circuit and it makes the algorithm more efficient. Also, this makes the algorithm easily adaptable to fit a particular purpose if enhanced with the use of semantic comparison instead of checking for an equivalence. It has been implemented using *pyNUSMV*. Thus, parsing and normalization functions follow an instrumentation of *NuSMV*: the same model checker is used to verify the soundness of the abstraction and the satisfiability of the properties.

IV. Case study

We successfully applied our framework to the verification of a hardware security monitor involved in a remote attestation scheme for ARM microprocessors [6]. This could not be achieved with classical automated approaches.

Modern Systems on Chip (SoC), such as Xilinx Zynq-7000, integrate ARM microprocessors along with programmable logic in a single device. This combines the flexibility and the parallelism of a Field-Programmable Gate Array (FPGA) with the performances of an Application-Specific Integrated Circuit. Spatial partitioning for sensitive memories and implementation of our hardware security monitor takes place in the FPGA.

ARM microprocessors come with a debug interface called *CoreSight* which enables real-time instruction flow tracing without slowing down execution. Traces contain information to reconstruct the execution of a program which, in our case study, is composed of cryptographic primitives.

During the computation of an integrity check, the activation of program flow tracing, combined with the addition of specific instructions, provides data that can be used for monitoring. These traces are accessed by the hardware security monitor in the FPGA and processed to achieve remote attestation security.

The proof strategy to achieve remote attestation security is described in [6] along with the architecture of our security monitor. Proving the security is an iterative process involving *model-checking* and proving lemmas for each of four hardware sub-modules as described in figure 2.

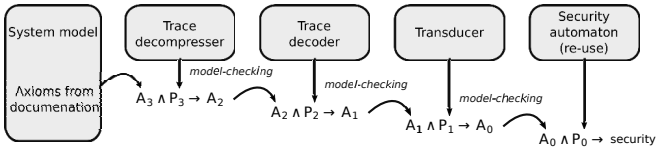


Fig. 2. Proof strategy

Overall security is based on axioms that are formally expressed from the documentation of the SoC (A_3). In particular, the format of ARM *CoreSight* traces and the events causing their output are described in PSL. The security monitor is a composition of four sub-modules dedicated to trace decompression, trace decoding, transduction and security enforcing. *Model-checking* ensures that each sub-module verifies local properties P_i (with $i \in [0:3]$) and proofs of lemmas provide new axioms A_i for the next sub-module.

Understanding the overall verification of the security monitor is not a prerequisite to appreciate the application of our method as it is an iterative process. To illustrate our approach, we focus on one iteration: the verification of temporal properties and proof of a lemma on the *CoreSight* trace decompresser. This is relevant because the trace decompresser is the sub-module of the security monitor which comes with most memories, hence the bottleneck to *model-checking* regarding the state-space explosion problem.

A. The decompresser

Traces are transmitted packet-wise by *CoreSight*. These packets are compressed so that unmodified data between two transmissions is not repeated. A packet header determines the type of packet being transmitted and compression bits inform about the presence of following bytes of data [26]. Figure 3 gives an overview of the decompresser's input/output.

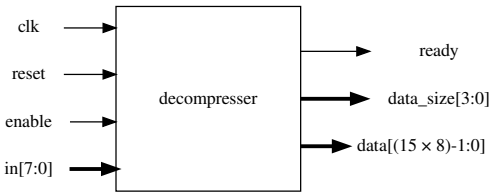


Fig. 3. Overview of the decompresser's inputs/outputs

The decompresser retrieves data from a 8-bit vector (in) and features a synchronization clock (clk) and an $enable$ bit: data is read from input in only when *CoreSight* asserts this $enable$ bit, which can be de-asserted in the middle of a reception for an undefined amount of time. Once all data is received, it is output to a longer vector which contains the content of the whole packet. Minimal size for the decompresser's memory is fixed by the length of the biggest packet, hence a 15 bytes memory to store the payload of *isync* packets [26].

The type of packet being decompressed is identified from the received header. Expected size for a packet depends on this identification and the content of the packet; it can vary between 1 and 15 bytes. Output $ready$ is set at the reception of the last byte, output $data\_size$ gives the number of bytes in the decompressed packet and output $data$ gives its content.

B. Specifications

To guaranty the overall security for the monitor, decompression for several types of packets must be correct. In particular, *CoreSight branch* packets, that provide destination address for an indirect branch and exception informations, are the longest packets which decompression must be verified [26], [6]. In this section, we describe the specifications for correct decompression of a *branch* packet.

- data is memorized from input in, at rising edge of clk, when $enable$ bit is asserted;
- packet type is deduced according to the received header, i.e. the first received byte;
- if input $reset$ is asserted at rising edge of clk, memorization and packet type deduction are discarded;
- output $ready$ rises when the decompresser receives the last byte of a *branch* packet and is set only during one clock cycle;
- output $data\_size$ gives the number of bytes in the decompressed packet when $ready$ rises;
- output $data$ takes the value of the decompressed packet when $ready$ rises.

Figure 4 shows an example of the reception of a 4 bytes packet, where w, x, y and z are received bytes (w contains the packet header) and signal $memory$ refers to an internal memory. The last received byte is output at its reception.

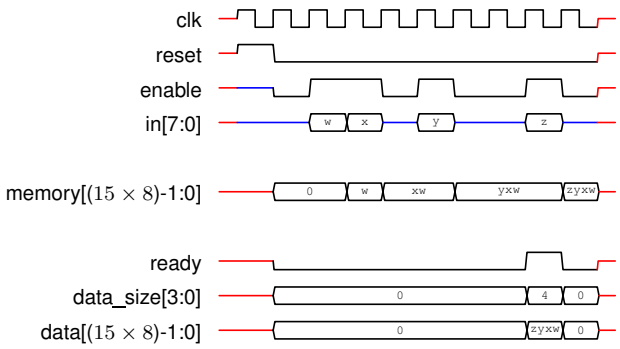


Fig. 4. Description of the data flow stream

V. Application to the Case Study

In this Section, we show how we applied our approach to verification of the decompression for ARM *CoreSight branch* packets.

The concrete model for the decompresser uses a total of 132 bits for SMV state variables and 11 bits for SMV input variables. Its description in *Verilog* HDL has approximately five hundred lines of code. *Verilog2SMV* is used to re-create the SMV model from the HDL, which has approximately five thousand lines of code. Both *Verilog* and SMV models are available at [13].

Verification is conducted using our framework on a computer cluster of 256 nodes for parallelization. Each node has $4GB$ of RAM and a single-core CPU running at $3GHz$. Computation times and memory usage for this application may differ if our approach is instantiated in an other toolchain.

A. Step 1: Formal expression of specifications and local properties

The first step consists in manually expressing the specifications and local properties in PSL. Formal expression of specifications is a difficult process and might give different results depending on the designer's writing choices: several PSL expressions may translate the same specifications in natural language.

Here, we provide an example for the formal expression of specifications from section IV-B. To ease understanding, we purposefully omit certain aspects of the specifications, such as the input filtering depending on the format of *CoreSight branch* packets. We also omit the duplication of **forall** operators for all replicators: this is not syntactically correct as a PSL replicator only accepts one identifier but it greatly reduces the representation of the expression. Complete PSL expressions with correct syntax and dependencies to *CoreSight* format are available at [13].

To guaranty the overall security for the monitor, decompression for the first 7 bytes of *branch* packets must be correct as they contain destination addresses and exception informations [26]. This is formally expressed by specification 1.

$$
\begin{aligned}
&\textbf{forall } i_0,i_1,i_2,i_3,i_4,i_5,i_6 \textbf{ in } \{0:255\}: \\
&\textbf{always}(\\
&\quad ((clk \wedge reset) \vee (clk \wedge ready)) \wedge \textbf{next}(\\
&\quad\quad \neg(clk \wedge reset) \textbf{ until } (clk \wedge ready) \\
&\quad\quad \wedge \textbf{next\_event}(clk \wedge ready)(data\_size \geq 7) \\
&\quad\quad \wedge \textbf{next\_event\_a}(clk \wedge enable)[1:1](in = i_0) \\
&\quad\quad \wedge \textbf{next\_event\_a}(clk \wedge enable)[2:2](in = i_1) \\
&\quad\quad \wedge \textbf{next\_event\_a}(clk \wedge enable)[3:3](in = i_2) \\
&\quad\quad \wedge \textbf{next\_event\_a}(clk \wedge enable)[4:4](in = i_3) \\
&\quad\quad \wedge \textbf{next\_event\_a}(clk \wedge enable)[5:5](in = i_4) \\
&\quad\quad \wedge \textbf{next\_event\_a}(clk \wedge enable)[6:6](in = i_5) \\
&\quad\quad \wedge \textbf{next\_event\_a}(clk \wedge enable)[7:7](in = i_6) \\
&\quad) \\
&\rightarrow \\
&\quad \textbf{next}(\\
&\quad\quad \textbf{next\_event}(clk \wedge ready)(data[7:0] = i_0) \\
&\quad\quad \wedge \textbf{next\_event}(clk \wedge ready)(data[15:8] = i_1) \\
&\quad\quad \wedge \textbf{next\_event}(clk \wedge ready)(data[23:16] = i_2) \\
&\quad\quad \wedge \textbf{next\_event}(clk \wedge ready)(data[31:24] = i_3) \\
&\quad\quad \wedge \textbf{next\_event}(clk \wedge ready)(data[39:32] = i_4) \\
&\quad\quad \wedge \textbf{next\_event}(clk \wedge ready)(data[47:40] = i_5) \\
&\quad\quad \wedge \textbf{next\_event}(clk \wedge ready)(data[55:48] = i_6) \\
&\quad) \\
&) \qquad (1)
\end{aligned}
$$

This PSL specification deals with traces starting when the decompresser is reset or ready. From the next state, a restriction for the traces is that input $reset$ cannot be asserted at a rising edge of clock until output $ready$ is set. Also, only traces where $data\_size$ is greater or equal to 7 are considered.

Left side of the implication expresses that data is memorized from input in, at rising edge of clk, when $enable$ bit is asserted. This is true for all possible values between 0 and 255 for each of the 7 bytes of data. Right side of the implication expresses that output $data$ takes the value of the decompressed packet when $ready$ rises.

Expression of local properties is a manual process. Conjunction for these local properties must imply PSL specification 1. The designer can keep in mind that this implication is proven by an automatic theorem prover and that abstracting a module's output in a property reduces both the complexity of the property and the size of the automaton for *model-checking*.

One possible solution for expressing local properties is to split the content of output $data$ into seven bytes, where each local property has one replicator of 256 possible values. An other solution is to split the content of output $data$ into 7×8 bits, where each local property has one replicator of 2 possible values. The first solution is straightforward as expressing a local property only consists in removing **next_event_a** operators from specification 1. The second solution requires more rewritings but reduces the complexity of the property and the size of the automaton even more, enabling *model-checking* for more complex specifications.

For our case study, following the first solution is sufficient to enable formal verification. An example for one local property is given by expression 2. A total of seven local properties, following the same approach, is needed to imply the specification.

It is possible to give more expressive power to a local property — in case the same property helps proving more than one specification. For this reason, the value of output $data\_size$ in property 2 is now greater or equal to one —

$$
\begin{aligned}
&\textbf{forall } i_0 \textbf{ in } \{0:255\}: \\
&\textbf{always}(\\
&\quad ((clk \wedge reset) \vee (clk \wedge ready)) \wedge \textbf{next}(\\
&\quad\quad \neg(clk \wedge reset) \textbf{ until } (clk \wedge ready) \\
&\quad\quad \wedge \textbf{next\_event}(clk \wedge ready)(data\_size \geq 1) \\
&\quad\quad \wedge \textbf{next\_event\_a}(clk \wedge enable)[1:1](in = i_0) \\
&\quad) \\
&\rightarrow \\
&\quad \textbf{next}(\\
&\quad\quad \textbf{next\_event}(clk \wedge ready)(data[7:0] = i_0) \\
&\quad) \\
&) \qquad (2)
\end{aligned}
$$

since we only verify the first byte of data. But, care must be taken, when modifying local properties, that the specification remains provable.

B. Step 2: Proof strategy

Once all seven local properties are available, a proof that the specifications are implied must be conducted. Since we wrote the local properties with proof deduction in mind, the proof strategy is straightforward:

- expressions that appear in both the specification and the local properties can be left uninterpreted and abstracted; for instance, we can replace the following PSL expressions with single booleans:
  - **next_event_a**$(clk \wedge enable)[1:1](in = i_0)$
  - **next_event**$(clk \wedge ready)(data[7:0] = i_0)$
- axioms are added if we gave more expressive power to a local property. For instance, one axiom can be as follows: if $data\_size$ is greater than 7, then it is greater than 1.
- we rely on our framework to translate PSL into LTL, provide abstractions to reduce the proof effort and generate the formula. *Spot* is leveraged to prove the implication.

Figure 5 shows how we prove for the decompression for ARM *CoreSight branch* packets. This can be separated in three steps:

1) uninterpreted PSL expressions are replaced with single booleans ;
2) a proof is conducted that the specification is implied by the conjunction of local properties. This step is where *Spot* is leveraged, it is represented in red in Figure 5.
3) uninterpreted PSL expressions are refined to re-create the specification.

Optimizations can be achieved by separating the proof into several steps. In this case, we create an intermediate property, which is larger than the local property but smaller than the specifications to prove, then we follow the same approach. For example, an intermediate property can describe the decompression of the first four bytes; then an other intermediate property describes the decompression of the last three bytes. Steps $2'$ and $2''$ in Figure 5 represent how we optimize the proof. Sources to reproduce the experiment are available at [13]. Table I summarizes memory usage and computation times.

Optimizations	%MEM	Computation time
No (step 2)	9.6	2 minutes
Yes (steps $2'$ and $2''$)	< 1	< 2 seconds

TABLE I
PROOF: MEMORY USAGE AND COMPUTATION TIMES

Note: separating the proof into several steps is also an elegant solution in case we opt for the second solution when expressing the local properties, i.e. a split of output $data$ into 7×8 bits. In this case, property from expression 2 serves as an intermediate property. It must be proven from a conjunction of the local properties and the rest of the proof remains the same.

Regarding the abstractions of uninterpreted expressions, i.e. steps 1 and 3 in Figure 5, we provide a *coq* proof at [13] that this is correct at semantic level for any temporal property. This proof relies on a LTL library written in *coq* [27]. We assume that LTL semantic is identical for both *Spot* and this *coq* library. So, for this particular proof, verifying the correctness in *Spot* for each abstraction is unnecessary. Nevertheless, to advocate in favor of abstraction of uninterpreted functions, we leveraged *Spot* for a verification in some cases. A verification consists in proving an implication between a local property and its abstracted form (step 1 in Figure 5). In the abstracted form, for each value of integer j, the following PSL expressions are replaced with single booleans:

- **next_event_a**$(clk \wedge enable)[j+1:j+1](in = i_j)$
- **next_event**$(clk \wedge ready)(data[8\times(j+1)-1 : 8\times j)] = i_j)$

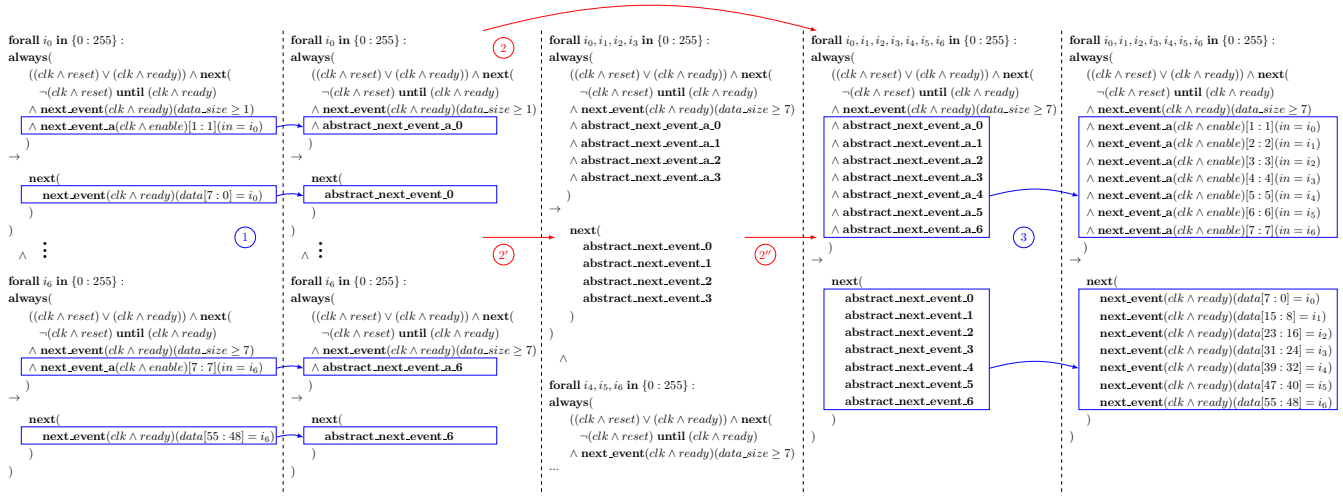


Fig. 5. Proving the decompression for ARM *CoreSight branch* packets

This verification is resource-intensive as complexity of the properties grows exponentially with the value of integer j. Sources to reproduce the experiment are available at [13]. Table II summarizes memory usage and computation times.

j	%MEM	Computation time
0	< 1	10 seconds
1	< 1	35 seconds
2	1.8	4 minutes
3	100	(runs out of memory)

TABLE II
UNNECESSARY VERIFICATION OF ABSTRACTIONS

This shows the limitations of automatic theorem proving when dealing with complex properties. Expression of extended next operators with basic temporal operators, which is not required if they appear in both the local properties and the specification, prevents the proof to complete. This is because the number of basic temporal operators grows exponentially with the value of integer j for our PSL expressions.

C. Step 3: Automatic localization reduction and model-checking

The conjunction of our local properties is sufficient to imply the specifications. Now, *model-checking* must guaranty that the decompresser verifies these local properties.

Since we opted for local properties where the content of output $data$ is split into bytes, our framework automatically removes 14 bytes of memory from the model, out of 15, at each abstraction. Also, since our local properties have one replicator of 256 possible values, our framework automatically splits the verification into 256 steps where properties are not replicated. As a consequence, a verification of a local property on our concrete model results in 256 verifications of sub-properties that are 2^8 times smaller, on an abstract model that is $2^{(14\times 8)}$ times smaller, hence an exponentially reduced BDD.

Note: even if the specification is not split into local properties, our framework would also automatically remove 8 bytes of memory out of 15 in the abstraction. This is because these 8 bytes of memory do not appear in the specification either. The verification process would also be split into 7×256 steps since the specification has 7 replicators with a range of 256 values each.

To verify the soundness, our framework automatically creates one concrete SMV model — in addition to an abstract SMV model for each sub-property. For each abstract SMV model, it tries to establish a simulation relation with the concrete SMV model:

- in case of success, *NuSMV* is used to verify that the property holds;
- in case of failure, it shows in a log file the differences between transition functions and initial states for the state variables that differ.

NuSMV computes *symbolic model-checking* of the sub-properties on their dedicated abstract model. It shows in a log file the sub-properties and the results of the computation:

- in case a sub-property holds, *NuSMV* provides the mention "is true";
- in case a sub-property does not hold, *NuSMV* provides a counterexample. In addition to the log file, our framework converts the counterexample into a VCD file which can be visualized with a waveform viewer.

Memory usage and computation times depend on both the size of the abstract model and the complexity of the property. Size of the abstract model is identical for each of our seven local properties. Complexity of the property mainly results of having a high integer number in the range of PSL operator **next_event_a**. Table III and figure 6 summarize memory usage and computation times on one node of our computer cluster, i.e. for the verification of one sub-property. Row indexes represent the integer number in the range of PSL operator **next_event_a**. Indexes 1 to 7 are of interest as they represent our seven local properties.

Note: for the sake of the experiment, we verified two unnecessary properties to evaluate our strategy of decomposition for more complex specifications. This shows that we could rely on the same solution if we specify the decompression of packets with 8 bytes of data, but we nearly reach the limits of our computer's memory at the 9th byte.

#	%MEM	Computation time
1	21.9	7 seconds
2	28.4	8 seconds
3	29.9	10 seconds
4	39.5	15 seconds
5	60.2	35 seconds
6	28.7	7 minutes
7	41.6	25 minutes
8	47.9	39 minutes
9	95.0	6 hours 55 minutes

TABLE III
MODEL-CHECKING: MEMORY USAGE AND COMPUTATION TIMES

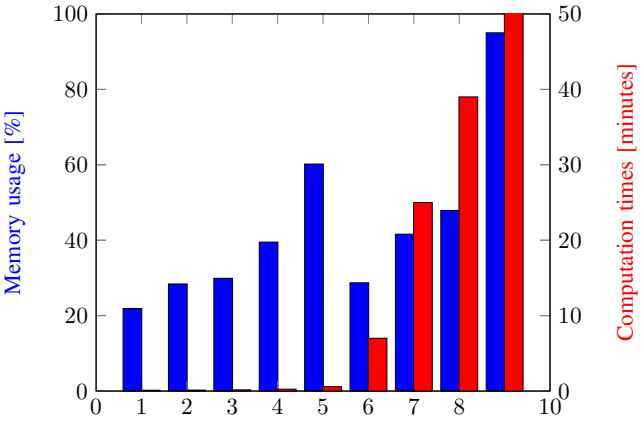


Fig. 6. Model-checking: memory usage and computation times

Sources to reproduce the experiment are available at [13]. We recommend to proceed to *model-checking* on a computer cluster of 256 nodes. Otherwise, computation times to verify one local property would be approximately 256 times higher.

D. Summary

To sum up, our framework automatically provided sound abstractions for our ARM *CoreSight* trace decompresser and increased compositional *model-checking* efficiency. Synthesis optimizations algorithms helped deducing these abstractions from the local properties to verify.

We had several solutions to express our local properties: we opted for a decomposition of an output vector into bytes as it eases proof deduction and allows *model-checking* in a reasonable amount of time. Fortunately, expression is conducted with proof deduction in mind. So, finding a strategy to prove the specification is straightforward: we opted to abstract functions depending on the decomposed bytes as they appear in both the specifications and the local properties.

In the case where our specification would have been more complex, we could also have opted for a decomposition of the output vector into bits. On the one hand, this solution would have forced us to add one step in the proof strategy since local properties would be smaller. On the other hand, it would have greatly reduced *model-checking* computation times as the abstract model would automatically be reduced as well.

We also applied our automated framework to the verification of several hardware modules: we verified the correctness for trace decompression, trace decoding, transduction and security enforcing. In the end, it allowed to verify the design of our whole security monitor and to prove the security for remote attestation of microprocessor software [6].

VI. Limitations

This approach can be generalized to the verification of other sequential circuits with complex specifications. Although, automatic theorem proving may show some limitations. Despite many abstractions, implication of the specifications may be too large to be processed in an acceptable amount of time. A solution is then to create an intermediate property, which is larger than the ones that are verified through *model-checking* but smaller than the specifications to prove. Then the proof must be separated into several steps following the same approach. An example of a proof in more than one step is publicly available at [13].

Other limitations may occur when it comes to verifying systems depending on a high number of inputs or when the memory of an atomic automaton already exceeds the tolerated size. For such systems, a workaround consists in altering their architecture so that they can be turned into partitioned systems. Such a strategy must be anticipated as freezing the architecture of a system is part of the early stages of the design flow for integrated circuits.

In a case where the specification contains PSL operators $\mathbf{next\_event\_a}$ with a too high integer number in the range, a different proof strategy must be considered. This might imply using *model-checking* to verify simpler properties — for instance, about the internal memory of the hardware — and rely on a more complex proof to imply the specifications. A drawback is that it increases the level of expertise needed to conduct the proof. In such case, relying on a proof assistant might be a more elegant solution than decomposing the proof in a high number of steps.

VII. Conclusion

The major obstacles for widespread application of formal verification to real-world designs are the complexity of their specifications and the state-space explosion problem. A lot of work has been dedicated to automatically provide smart abstractions for RTL *Verilog* and increase compositional *model-checking* efficiency [8], [9], [10], [11]. However, dealing with complex specifications requires to prove their implication from local properties, and writing properties function of automatically generated abstract models is inconvenient for automatic theorem proving.

In this paper we propose a framework to overcome this challenge and help the community going towards the adoption of formal methods. Our framework leverages the modular and partitioned aspects of sequential circuits and relies on synthesis and translation tools [14] to avoid a profound redesign. It computes convenient and sound abstract models from RTL *Verilog* and the properties to verify, allowing to prove complex specifications for hardware designs. Translation of PSL properties, to be fed to automatic theorem provers, is also automated and supports uninterpreted functions abstraction to relieve deductive proof systems in solving the satisfiability problem.

Further work can be conducted such as using a proven translation from PSL to LTL [16] to ensure the correctness of the LTL properties to manipulate. Nevertheless, our framework has been successfully applied as is to verify the design of a hardware security monitor for remote attestation of microprocessor software [6]. Such hardware design contains many memories and has similar complexity to the ones we find in industrial designs. This brings hope to see our approach adopted in the industry, especially since it is agnostic to the choice of an EDA toolchain.

References

[1] W. Khan, M. Kamran, S. R. Naqvi, F. A. Khan, A. S. Alghamdi, and E. Alsolami, "Formal verification of hardware components in critical systems," *Wireless Communications and Mobile Computing*, 2020.

[2] I. D. O. Nunes, K. Eldefrawy, N. Rattanavipanon, M. Steiner, and G. Tsudik, "VRASED: A verified hardware/software co-design for remote attestation," in *28th USENIX Security Symposium*. Santa Clara, CA: USENIX Association, Aug. 2019.

[3] C. Wang, G. D. Hachtel, and F. Somenzi, *Abstraction Refinement for Large Scale Model Checking*. Springer, 2006.

[4] E. M. Clarke, R. P. Kurshan, and H. Veith, "The localization reduction and counterexample-guided abstraction refinement," in *Time for Verification, Essays in Memory of Amir Pnueli*, Z. Manna and D. A. Peled, Eds. Springer, 2010.

[5] S. Berezin, S. V. A. Campos, and E. M. Clarke, "Compositional reasoning in model checking," in *Compositionality: The Significant Difference, International Symposium, COMPOS'97, Bad Malente, Germany. Revised Lectures*, W. P. de Roever, H. Langmaack, and A. Pnueli, Eds. Springer, 1997.

[6] J. Certes and B. Morgan, "Remote attestation of bare-metal microprocessor software: a formally verified security monitor," *The 5th International Workshop on Cyber-Security and Functional Safety in Cyber-Physical Systems (IWCFS)*, 2021.

[7] *Coresight Technology System Desgin Guide*, no. ARM DGI 0012D ID062610, 2006-2013.

[8] Z. S. Andraus and K. A. Sakallah, "Automatic abstraction and verification of verilog models," in *Proceedings of the 41th Design Automation Conference, DAC 2004, San Diego, CA, USA*, S. Malik, L. Fix, and A. B. Kahng, Eds. ACM, 2004.

[9] H. Jain, D. Kroening, N. Sharygina, and E. M. Clarke, "Word-level predicate-abstraction and refinement techniques for verifying RTL verilog," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 2008.

[10] Y. Ho, P. Chauhan, P. Roy, A. Mishchenko, and R. K. Brayton, "Efficient uninterpreted function abstraction and refinement for word-level model checking," in *2016 Formal Methods in Computer-Aided Design, FMCAD, Mountain View, CA, USA*, R. Piskac and M. Talupur, Eds. IEEE, 2016.

[11] C. Yu and M. J. Ciesielski, "Automatic word-level abstraction of datapath," in *IEEE International Symposium on Circuits and Systems, ISCAS 2016, Montréal, QC, Canada*. IEEE, 2016.

[12] *1850-2010 IEEE Standard for Property Specification Language (PSL)*. IEEE, 2010.

[13] J. Certes and B. Morgan, "Verification materials: source code and examples," 2021. [Online]. Available: https://gitlab.irit.fr/these-jonathan-certes-public/erts-2022

[14] A. Irfan, A. Cimatti, A. Griggio, M. Roveri, and R. Sebastiani, "Verilog2smv: A tool for word-level verification," in *2016 Design, Automation & Test in Europe Conference & Exhibition, DATE 2016, Dresden, Germany*, 2016.

[15] C. Wolf, "Yosys open synthesis suite," http://www.clifford.at/yosys/.

[16] T. Tuerk and K. Schneider, "From PSL to LTL: A formal validation in HOL," in *Theorem Proving in Higher Order Logics, 18th International Conference, TPHOLs 2005, Oxford, UK, Proceedings*, J. Hurd and T. F. Melham, Eds. Springer, 2005.

[17] T. Tuerk, K. Schneider, and M. Gordon, "Model checking PSL using HOL and SMV," in *Hardware and Software, Verification and Testing, Second International Haifa Verification Conference, HVC 2006, Haifa, Israel, Revised Selected Papers*, E. Bin, A. Ziv, and S. Ur, Eds., 2005.

[18] M. Y. Vardi, "An automata-theoretic approach to linear temporal logic," in *Logics for Concurrency - Structure versus Automata (8th Banff Higher Order Workshop, Banff, Canada, Proceedings)*, 1995.

[19] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella, "Nusmv version 2: An opensource tool for symbolic model checking," in *Proc. International Conference on Computer-Aided Verification (CAV)*. Copenhagen, Denmark: Springer, July 2002.

[20] A. Duret-Lutz, A. Lewkowicz, A. Fauchille, T. Michaud, E. Renault, and L. Xu, "Spot 2.0 — a framework for LTL and ω-automata manipulation," in *Proceedings of the 14th International Symposium on Automated Technology for Verification and Analysis (ATVA)*. Springer, Oct. 2016.

[21] R. Milner, "An algebraic definition of simulation between programs," in *Proceedings of the 2nd International Joint Conference on Artificial Intelligence. London, UK*, 1971.

[22] S. Bensalem, A. Bouajjani, C. Loiseaux, and J. Sifakis, "Property preserving simulations," in *Computer Aided Verification, Fourth International Workshop, CAV '92, Montreal, Canada, Proceedings*, G. von Bochmann and D. K. Probst, Eds. Springer, 1992.

[23] *NuSMV 2.6 User Manual*. FBK-irst - Via Sommarive 18, 38055 Povo (Trento) – Italy, 2015.

[24] S. Busard and C. Pecheur, "Pynusmv: Nusmv as a python library," ser. LNCS, G. Brat, N. Rungta, , and A. Venet, Eds., vol. 7871. Springer-Verlag, 2013, pp. 453–458.

[25] S. Williams, "Icarus verilog," http://iverilog.icarus.com/.

[26] *CoreSight Program Flow Trace Architecture Specification*, no. ARM IHI 0035B ID060811, 1999-2011.

[27] C. M. V. Reyes, "Ltl (linear temporal logic) in coq," https://github.com/spidermoy/LTL_Coq.

Session We.2.PO

Poster overview

Wednesday 1st June

14:00

–

Amphithéâtre

Short paper - Structural consistency of MBSE and MBSA models using Consistency Links

Romaric DEMACHY
Sébastien GUILMEAU
romaric.demachy@irt-saintexupery.com
sebastien.guilmeau@irt-saintexupery.com
IRT Saint Exupéry
Toulouse, France, France

ABSTRACT

The systems designed in industrial fields such as aeronautics or aerospace are more and more complex. In order to handle this complexity as well as the increasing need of digital continuity, model-based solutions are more and more introduced for system design (*MBSE*). In this picture, in order to ensure the consistency between the system design and the safety analysis, and thus increase the confidence in safety analyses results, our proposal is to ensure the consistency between *MBSE* and *MBSA* (Model Based Safety Analysis), which represent different views of the same system. To do so, we define Consistency Links (*CL*) that make a bridge between the structural items of each model. Associated with dedicated rules, that can be systematically checked, the *CL* can be used to drive the *cross-review* of models done by system engineers (*SE*) and safety specialists (*SA*) to increases detection of inconsistencies between models.

The work presented here is part of the S2C project and involves industrial partners from the space and the aeronautical industry. It is led jointly by IRT Saint Exupéry and IRT SystemX.

Keywords : Model-Based System Engineering, Model-Based Safety Assessment, digital continuity

1 PROBLEM STATEMENT

When designing a system which must comply with safety requirements, the process shall ensure, as early as possible and all along the design phase, that the system architecture is compatible with them.

The safety assessment is performed by *SA* teams and imply the usage of safety methods and tools. Methods such as Fault Tree Analysis have been used for decades, but *MBSA* approach has emerged and is now recognized as an acceptable mean of compliance by aeronautic regulation authorities. Indeed, this approach is identified in ARP4761A, which gives guidelines and methods of performing the safety assessment for certification of civil aircraft, and will be soon published by SAE International. Ensuring the correctness of *SA* models with regards to the system design is mandatory for the relevance of the safety assessment. In current practices, this relies on exchanges and *cross-review* between the *SE* team and the *SA* team.

As the system design process also progressively relies on *MBSE* approaches, there is an opportunity to ease this review process by taking advantage of the provided model's formalism. The problem becomes : how to ensure a better consistency between the *MBSE* model and the *MBSA* model ? As these analyses are currently performed on specific tools dedicated to either *MBSE* or *MBSA* analysis, we choose here to focus on the case where two different (i.e. each model has its own objectives and modeling choices) and heterogenous (i.e. each model uses a specific modeling language) models are used. In this paper, the problem is narrowed taking into account the following constraints :

- Constraint A (CA): For *MBSE* models: only architecture description models (Capella, SysML, ...) and exclude specialised models (digital mock-up, electrical wiring models,etc...)
- Constraint B (CB): For *MBSA* models: we consider failure propagation models (Altarica,...)
- Constraint C (CC): Only the structural consistency is covered in this paper, the consistency of the behavior being a more difficult issue

The benefits of using such models to ease the review between system and safety experts in the aeronautical context has been discussed in [1]. The problem of synchronisation of architecture models and safety models has been adressed in [2]. This thesis is based on the analysis that the information exchanged between these assets are informal. In [3] and [4] a processus for the synchronisation of *MBSE* and *MBSA* models has been proposed, consisting in the projection onto a dedicated language called S2ML (System Structure Modeling Language). [5] proposes an approach consisting in a digital collaborative space based on the federation of modeling languages into a common ontology.

The work presented here consists in a projection into a pivot meta-model in order to evaluate the structural consistency. The proposed method is implemented it in a tool and experimented on a representative case study.

Section 2 describes the principles of the method and details the consistency link concept. Section 3 gives details on how the method has been implemented in a tool for a Proof of Concept. Section 4 shows how the method and its implementation have been experimented on a representative case study, and what are the qualitative gains that have been identified. Section 5 lists the perspectives for future activities.

2 CONSISTENCY LINKS METHOD

To ease the consistency review, the proposed method consists in defining consistency links (*CL*) between groups of artifacts of each model with the following semantic : "The *MBSE* model element(s) and the *MBSA* model element(s) linked together represent the same object". Then, we can decompose the review to address only small and well defined perimeters at a time.

A *CL* carries, also, the consistency validation by reviewers. This is made concrete by the elements associated to the *CL* : a rationale that captures justification and assumptions, a validation status, meta-data such as the the review date and authors. The meta-model of the *CL* object and associated concept is represented on the figure 1 below.

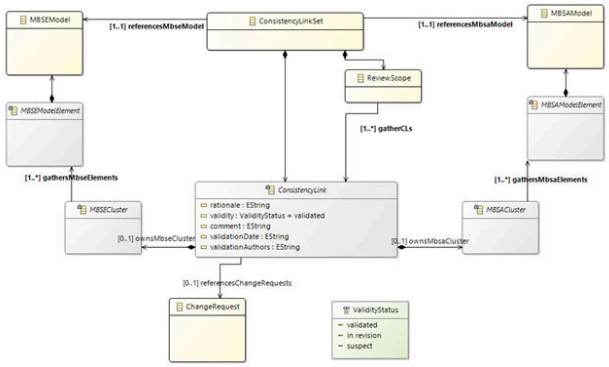


Figure 1: Metamodel of consistency link.

In this paper, the *CL* has been particularized for the functional architecture, although it could be adapted to other viewpoints, such as the logical or the physical architecture. Two types of CL are defined : *CL* for Functions (*CLF*), and *CL* for functional flows (*CLfl*). These concepts are illustrated on figures 2 and 3 below.

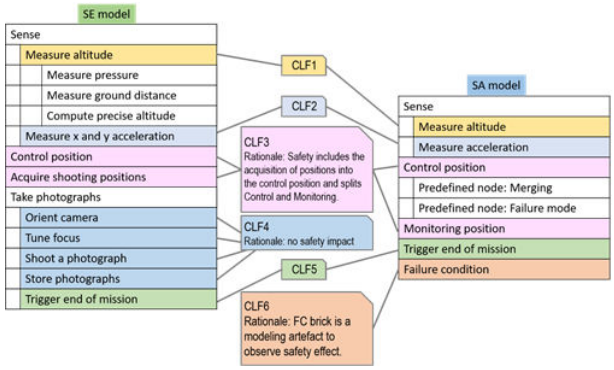


Figure 2: Consistency links for functions.

Coverage and consistency rules for these CL have also been defined.

The coverage rules ensure that all leaves of the functional breakdown and all functional flows are covered by one and only one *CL*

- Rule 1 : Each leaf function (i.e. lowest function in functional breakdown structure) of each model shall either be linked by one *CLF*, or have one hierarchical function (at any level of breakdown) that is linked by one *CLF*.
- Rule 2 : Each flow whose source and destination functions are linked to different CLF shall be linked by a *CLfl*.

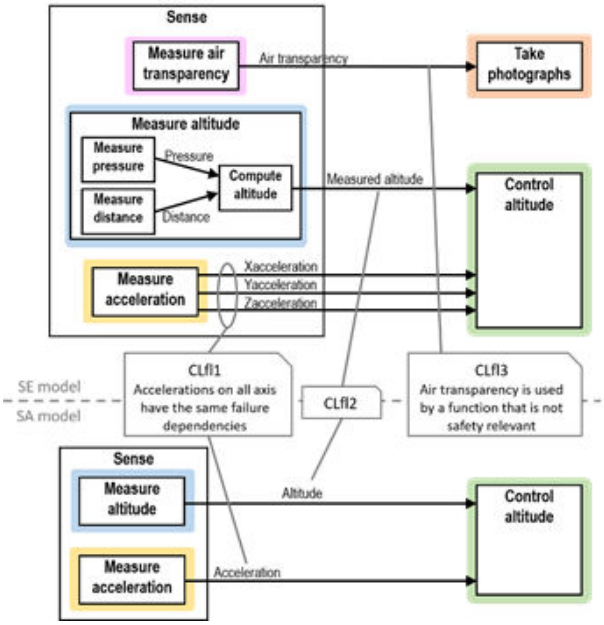


Figure 3: Consistency links for functional flows.

The consistency rules ensure that the defined *CLF* and *CLfl* are globally consistent.

- Rule 3 : In each model, two flows that are linked to a same *CLfl* shall have source functions that are linked to a same *CLF*. Symmetrically, they shall have destination functions that are linked to a same *CLF*.
- Rule 4 : Given a *CLfl*, the source *CLF* from MBSE model shall be the same as the source *CLF* from MBSA model. Symmetrically, the destination *CLF* from MBSE model shall be the same as the destination *CLF* from MBSA model.

Checking that the *CL* set is compliant to these rules can be easily automated.

3 IMPLEMENTATION

The implemented process is illustrated by the figure 4, and starts with the *SE* and *SA* domain's models that are to be *CL*-linked. Constraints CA and CB induce an horizontal "language gap" because methods and tools (*M&T*) differs between domains. To work around this, models are translated automatically to abstracted ones (considering their relative *M&T*). Pragmatically, those new models are compound data flow graphs where edges are only between the more nested nodes. That means hierarchical functions of functional decomposition are the compound nodes with no flows between them. *SEIM* (Systems Engineering Information Model) meta-model rules both new models. It is limited to a subset of concepts required by structural consistency needs due to constraint CC. So, *SEIM* introduces a vertical "language gap", filled by the definition of a transformation logic producing *SEIM* concepts from domain's tools ones. The automation (i.e. concepts' extraction from domain's model then transformation to *SEIM* ones) is the last step of abstraction activity. *SEIM* policy is not to merge both domain's languages

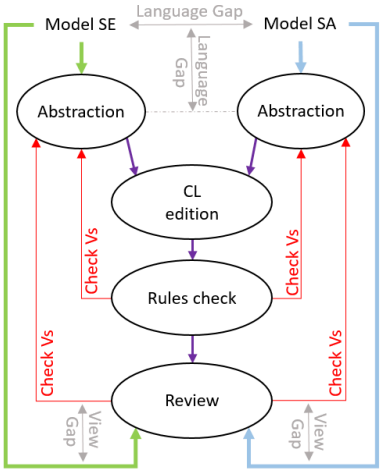


Figure 4: Implementation synopsys.

in an universal one, but rather to be the minimal intersection between domain's meta-models to fulfill targeted needs. This policy reduce the analysis workload on tools' meta-models.

Next process' step is the edition of *CL* via a graphical user interface (*GUI*), see figure 5. Through it, zero or more *SE* abstracted

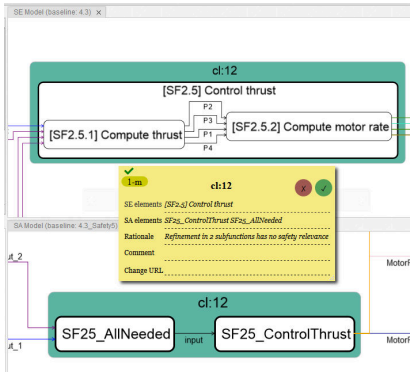


Figure 5: Partial view of the Graphical User Interface.

artefacts are linked with zero or more *SA* abstracted ones. As the new models may have not the same abstraction's depth (inherent to the initial models), abstracted hierarchical compound nodes can be linked too. This allows a customized alignment between abstracted models limited to structure in this paper. This alignment is dependent of the granularity of domain's models, that means a coarsest model will drive the alignment of models. Complementary data (like rationale) are added, also, to get grips with realized grouping for the future *cross-review*. *GUI* has graphical capabilities that auto-layouts part of both abstracted models simultaneously, so that editor can navigate freely through them or redisplay artefacts he grouped previously via *CL*. At each saving action, the defined rules are automatically checked using *CL* and models. Linking may reveal inconsistencies originated by erroneous *CL* edition or by inconsistencies between models. In the first case, *CL* are updated while correction and publication of model has to be done for the second one. At end of the step, both disjunctive partition of the respective intermediate and abstracted models is reached.

When all rules are passed, *cross-review* between *SE* and *SA* experts starts from *CL* editor's proposed-partitioning. *GUI* graphical capabilities are used again so that both contributors share the same common representation. As the *GUI* represents an abstraction of original models, it exists a "view gap" (see 4). For Structural concerns, this discrepancy is limited. Round trips between authoring tools and *GUI* remain easy. After experts agree on a *CL*-scoped consistency they change its status and update possibly its rationale too. When all *CL*-scoped *cross-review* are done, a global and justified consistency status can be acted regarding the models.

Implementation considered also the iteration problem of models and *CL*. During design phase, models are updated (corrected, enlarged, etc,...) so the consistency status becomes suspicious at each evolution. But for the versioned models and *CL*, the rules of method and the automated step are reused to identify the flaws and/or corrupted *CL*. So incremental *cross-review* can be achieved by *SE* and *SA* experts to foster inspection only on impacted part of their models.

4 VALIDATION

In order to assess the feasibility and evaluate the gains of the method presented in 2 and implemented in the consistency management tool as described in 3, it has been experimented with the AIDA case study [1]. This case study is a drone system which aims at assisting the pre-flight check of a commercial aircraft. It consists mainly in an architecture description model in Capella [2] which is used here as the MBSE model. For the purpose of the S2C project, an *MBSA* model has been developed in the SimfiaNeo tool, edited by Apsys[3]. This case study is representative of a medium size aeronautical system : obviously not as complex as an aircraft, but with enough depth and complexity to assess the feasibility of the method in an industrial case.

The validation activity has explored several phases of the lifecycle of such models : the initial creation of the *CL* set, the update of the *CL* set following changes in one of the models, the *cross-review* led jointly by *SE* and *SA* specialist to validate each *CL*. The *cross-review* exercise has been done as closest as possible to real conditions, with a system engineer and a safety specialist.

The tables 1 and 2 show some metrics to illustrate the size of the case study. Table 1 shows the number of model items, before and after the abstraction step. This illustrates the benefits of the abstraction step, which "flattens" the flows in the SimfiaNeo model. As Capella already applies the principle of direct flows between leaves elements, the abstraction step does not further reduce the number of flows in the *MBSE* abstracted model. Table 2 shows the number of *CL* created along with their cardinality (number of elements from each model in a *CL*). It illustrates the flexibility proposed by the method, which enables to associate any number of elements from each model in a same *CL*. In particular, the possibility

[1] AIDA is a public case study developped by the System Engineering center of competence of IRT Saint Exupéry. It is fully open-source and available here : https://sahara.irt-saintexupery.com/AIDA/
[2] https://www.eclipse.org/capella/
[3] https://www.apsys-airbus.com/

Type of model element	*MBSE* model	Abstracted *MBSE* model	*MBSA* model	Abstracted *MBSA* model
Functions	159	159	148	148
Funct. flows	285	285	438	196

Table 1: Complexity of the *MBSE* and *MBSA* models

Cardinality	Numb. of *CLF*	Numb. of *CLfl*
1 *MBSE* to 1 *MBSA*	33	60
1 *MBSE* to n *MBSA*	2	7
n *MBSE* to 1 *MBSA*	7	20
n *MBSE* to m *MBSA*	2	3
0 *MBSE* to 1..n *MBSA*	1	9
1..n *MBSE* to 0 *MBSA*	6	36
Total	**51**	**135**

Table 2: Complexity of the resulting CLset

to associate elements from one model only to a *CL* (i.e. those for which the cardinality is 0-1..n) is useful for model elements that are relevant in only one of the model. For example :

- The *MBSE* model may contain functions that have no safety impact and are not represented in the *MBSA* model. This can occur when a preliminary analysis, such as a Functional Hazard Assessment (FHA), has been realised, or thanks to "expert" knowledge of the system.
- The *MBSA* model contains *SA* specific artifacts, such as the failure conditions observers, that are not represented in the *MBSE* model.

The Rationale attribute of the *CL* allows to capitalize the modelling choices and associated assumptions. The *cross-review* will particularly focus on the validation of these "not 1-1" *CL*.

The validation activity has shown the following qualitative gains for the proposed method :

- the *CL* comes on top of the existing *MBSE* and *MBSA* models, without generating additional modeling constraints,
- the coverage and consistency rules associated to the *CL* set have shown efficiency in the detection of mismatches in the flow consistency, while being flexible enough to address a large number of model elements at the desired level of details,
- the *CL* are helpful for the detection and propagation of model changes,
- the *CL* offer a structure for an efficient *cross-review* focused on model changes,
- the tooling support for consistency link definition and *cross-review* is feasible outside the captive authoring tool, although the developed tool could be matured for a better user experience,
- the *CL* are relevant for discussions and justifications capitalization.

Globally, the *CL* method has proven to be useful to increase the confidence in the structural consistency of models. The induced workload may be slightly increased, which can be put in balance with the avoidance of running future biased analyses due to inconsistent models.

5 PERSPECTIVES AND FUTURE WORK

The work presented here is a first attempt to ensure the consistency between *MBSE* and *MBSA* models. It focuses on the structural consistency of the functional architecture. The tool implementing the method has the maturity of a Proof Of Concept.

Several axes of improvement can be identified :

- The method could be extended to cover also the logical and physical architectures. Topics such as the allocation of functions on logical or physical components could be addressed. This would be particularly relevant as the safety assessment is usually performed at those levels of representation, and not only on the functional aspects.
- The tool can be improved in order to provide better user experience : rationalization of the displayed information, improvement of navigation and user displayed messages. Additional capabilities such as report edition or assistance algorithm for the creation of *CL* (ex: *CL* suggestion based on the similarities of the objects names in both models) could also be considered.

In addition of the local consistency handled by the CL at structural level, it is important to assess the consistency between the *MBSE* and *MBSA* behavioral level. Two approaches are possible either by simulation on overall models or by static local analysis on common models perimeters.

6 CONCLUSION

With the emergence of model-based approaches for the design of complex systems, and because these systems are sometimes subject to strong safety requirements emitted by the regulation authorities, the problem of ensuring the consistency between *MBSE* and *MBSA* models of a same system arises.

Within the frame of the S2C project, we proposed a method to address the topic of structural consistency. An object called "Consistency Link" (*CL*) has been defined, and particularized to address the functional architecture. These *CL* are constrained by coverage and consistency rules.

For validation purpose, the method has been implemented in a tool. Although some improvements of the tool are needed to make it usable in a industrial context, it helped to assess the validity of the method.

The AIDA study case has been used to experiment the method. It has shown qualitative gains for the consistency *cross-review* activity, and an overall improvement in the trust one can have in the safety assessment of the system.

The method only addresses at the moment the problem of structural consistency of the functional architecture. While some improvement axes for the method have been identified, the S2C project currently focus on possible approaches to evaluate the behavior consistency.

REFERENCES

[1] T. Prosvirnova, E. Saez, C. Seguin, and P. Virelizier. Handling consistency between safety and system models. In *IMBSA 2017 (International Symposium on Model-Based and Assessment)*, pages pp.19–34.

[2] Anthony Legendre. Ingénierie système et sûreté de fonctionnement : Méthodologie de synchronisation des modèles d'architecture et d'analyse de risques.

[3] Michel Batteux, Tatiana Prosvirnova, and Antoine Rauzy. System structure modeling language (s2ml). 2015.

[4] Michel Batteux, Tatiana Prosvirnova, and Antoine Rauzy. Model synchronization: A formal framework for the management of heterogeneous models. In Yiannis Papadopoulos, Koorosh Aslansefat, Panagiotis Katsaros, and Marco Bozzano, editors, *Model-Based Safety and Assessment*, pages 157–172, Cham, 2019. Springer International Publishing. ISBN 978-3-030-32872-6.

[5] Laurent Wouters, Stephen Creff, Emma Effa, and Ali Koudri. Collaborative systems engineering: Issues & challenges. pages 486–491, 04 2017. doi: 10.1109/CSCWD.2017.8066742.

Experimenting with Dynamic Cache Allocation to Improve Linux Real-Time Behaviour

Aléxis Génèrès†,
† LAAS-CNRS
31400, Toulouse, France
Email: ageneres@laas.fr

Michaël Lauer†, Jean-Charles Fabre†
† LAAS-CNRS
31400, Toulouse, France
Email: firstName.lastName@laas.fr

***Abstract*—Embedded systems have an increasing need for computing power. To address this issue, we can implement critical and non-critical tasks in the same multicore processor. A disadvantage of this kind of processor is the indeterminism it involves due to its complexity. New technologies, like dynamic allocation of cache memory, allow reducing the impact of this indeterminism. In this article, we provide an experimental approach to verify if the dynamic allocation of the cache memory of Intel (CAT Intel) is efficient.**

***Index Terms*—multi-core, real time, multiple criticity, cache allocation**

I. Introduction

The growing need for computing power in embedded systems is leading companies to use multi-core processors. The increased number of onboard applications requires combining high criticality tasks with low criticality tasks on the same multi-core processor. Unfortunately, the uses of multi-core processors induce a lack of predictability of their temporal behavior due to their shared resources: memories, communication buses... These factors can provoke deadlines violation for high criticality tasks, and also a sub-optimal use of the computing power for low criticality tasks. [1].

In this paper, we target one factor: the cache memory shared between the cores of a processor. Our goal is to evaluate the gain of our mechanism of dynamic cache allocation regarding deadlines respect and processing power performance. Furthermore, we want to know if the use of these mechanisms can be implemented with a minimum effect on the background tasks. On the one hand, a static allocation allows to isolate the temporal tasks and thus, to limit the interferences linked to the shared resources. However, on the other hand, the cache memory, in case of a static allocation, is then no longer fully available for other tasks, even if the temporal tasks are terminated. This limits the use of the processor. In this study, we will therefore focus on dynamic allocation to avoid this behavior.

In this preliminary work, we will realize a set of experiments using high criticality tasks and low criticality tasks on a single multi-core processor. More specifically, we will use the technology of cache memory allocation to isolate the cache memories of the tasks having a different level of priority. The goal is to evaluate the impact of dynamic low-level cache allocation mechanisms on the side-effect of shared resources.

To achieve this evaluation, we will use the *Earliest Deadline First (EDF)* scheduling of Linux. Based on the works of Lelli [2], we can use the EDF scheduling of Linux and guarantee the real-time requirements if we limit the computing power at 90% on one core. For our experiments, we will also use CAT technology of Intel. In section II of this article we describe this work. In this section, we also define our experimental approach and tools to analyze the last level of cache memory mechanism. In section III, we detail our experimental platform and our protocol to evaluate the mechanism.

II. Experimental approach and tools

A. Experimental approach

The goal behind our experimental protocol is to improve the execution determinism of the high criticality task while also ensuring optimal performances for the low criticality tasks. To this end, we will evaluate the relevance of the dynamic allocation of cache memory on the shared cache memory.
To do this, we will simulate one high criticality task with 12 low criticality tasks. After that, we will measure the response time of the high criticality task to evaluate the non-violation of the deadline and thus the execution determinism. In a second time, we will estimate the computing power that we can use for low criticality tasks by measuring the IPCs of the 12 low criticality tasks.
Therefore, the experimental approach to evaluate the impact of dynamic low-level cache allocation mechanisms for reducing the effect of shared resources will be composed of three phases.

1) Without cache memory allocation.
2) With static cache memory allocation.
3) With dynamic cache memory allocation.

If we do not observe any ”deadline miss” nor any behavior side-effect with dynamic allocation, we can state that dynamic allocation has no negative impact on the high criticality task.

B. Tools

In our experimental framework, the high criticality task is scheduled with the EDF scheduling policy of Linux (named *SCHED_DEADLINE*). This scheduler is characterised by three key parameters: *”runtime”*, *”deadline”* and *”period”*. The ”runtime” parameter represents the budget allocated by the OS to perform the task before its deadline. This budget is refreshed at each new period. If this budget is fully consumed, the task is suspended and we consider its deadline as missed. We manage

the low criticality tasks using the default scheduling policy of Linux: ”time-sharing”.

For cache memory allocation, we use the Intel CAT (Cache Allocation Technology)[3]. CAT allows us to allocate a subspace of memory of the last level cache of the processor. We can use CAT for multiple cores or processes. The allocation is done via a software lock: a mask that changes the write permissions only. It's granularity is determined by the number of *ways* in the CPU. For example, in a 8 *ways* last level of cache, we can allocate from 1 to 8 *ways* (zero is not possible) to one or multiple processes. The maxmium number of allocation depends on the CPU models.

Finally, we use BCC (BPF Compiler Collection)[4] to collect data (IPC and response times). BCC permit our observation mechanisms and measures to be implemented. BCC is a group of elementary tools which allows us to program and use BPF (Berkeley Packet Filter) probes in order to collect the necessary data for our experiments. BCC introduces a low overhead in terms of computing power and provides key elements to implement fine-grained instrumentation.

III. Experimental protocol

The overall approach to address the problem is composed of three phases. Each phase consists of the execution of the high criticality task alone, then, in a second step, the execution of the high criticality task and low criticality tasks simultaneously. This step aims to generate interference. The three phases are the following:

- Without cache memory allocation.
- With static cache memory allocation.
- With dynamic cache memory allocation.

For the realization of the experimental protocol, we will use a two processors system. Each processor is an Intel Xeon Bronze 3204 (named Xeon 3204). A Xeon 3204 has 6 cores without hyperthreading, its last level is an 11-way set-associative cache with a size of 8.25MB. The L2 caches are of a size of 1MB each and are not shared by multiple cores.

Based on the information about the last level cache of our processors, we can explain the use of CAT in this experimentation. Using CAT we can exploit two times 11 *ways* for the allocation. Each *way* is an allocation of 0.75 MB of memory space. Since our high criticality task needs 3 MB of cache memory space, to isolate it, we will use CAT to allocate a sufficient number of ways : 5 ways. We will also deactivate the write authorization in the same memory space for the low criticality tasks.

The operating system we will work on is a 5.12.5 Kernel. It allows us to have access to the last version of BCC. We will use BCC to save the start date of the high criticality task. We will also need, using this compiler collection, to save any violation of the deadline for the high criticality task and, to measure the Instruction per Cycle (IPC) of each core.

A. Step by step protocol

Each of these phases (without allocation, with a static allocation, and with a dynamic allocation) regroup two steps; the first with the high criticality task alone and the second with the high criticality task and the low criticality tasks in parallel.
As we use the EDF Linux schedule for the high criticality task, we must specify the three parameters of this scheduler: runtime, deadline, and period.

For our experiment, the runtime is 150ms and the period is equal to the deadline: 200 ms. Both of the tasks (high criticality and low criticality) use an entry data size of 3 MB. Indeed, to provoke the effect of the shared memory, the entry data size needs to be greater than the size of L2 cache memory size which is 1MB for the Intel Xeon 3204.

1) Without cache memory allocation phase: as mentioned before, the first step is to launch the high criticality task alone with the observation tools. This step intended to collect the reference response times.
The second step is to launch the high criticality task and the low criticality tasks at the same time. The low criticality tasks are represented by 12 stress tasks, one per core (two processors with 6 cores), which will flood the cache memory. Thereby, we could measure the effect of noisy neighbors on our system. This effect is due to the interferences of the memories shared by other tasks.

2) With static cache memory allocation phase: the static cache memory allocation will be effective until the end of this step. To set the allocation, we will allocate five successive *ways* among the 11 available in the processor. We can do this allocation using the CAT intel technology.

This allocation will be exclusive to the high criticality task. The allocated memory size is 3.75 MB (0.75MB by *way*). In fact, it needs to be greater than 3 MB which is the size of the data used by the high criticality task.

The first step, which is to launch the high criticality task alone, will allow us to observe the effect of cache allocation on the high criticality task response time. This effect should be insignificant.

The second step, which is to load the system with our 12 stress tasks, should give us the same results, i.e we will not have any modification of our response times because the allocation protects the high criticality task. Since we use only five or the 11 ways, we suppose that the absence of the six other *ways* will impact the number of instructions per cycle.
To observe this result we compare:

- the average response time.
- the average IPCs of all the cores.

3) With dynamic cache allocation phase: for this phase, we also use five of the existing *ways* for the allocation. We will deactivate this allocation when the high criticality task has terminated and reactivate it at the beginning of a new period.
To analyze the impact of the dynamic allocation, we will compare the dynamic and the static allocation regarding the

average response.

To study the computing power of the low criticality tasks, we will compare the dynamic and the static allocation regarding the IPCs. This comparison will be done while the stress tasks are activated. We should observe a higher IPC in the case of dynamic allocation. This would conclude about the efficiency of this mechanism towards low criticality tasks.

IV. Related works

In this section, we will detail some solutions to make coexist a high criticality task and tasks of low-level criticality on the same multi-core processor.
The first solution is illustrated by the works of Suzuki and others [5]. It suggests allocating for each core a subspace of the cache memory. Then, to set the core affinity of the high criticality task to only one core and assign all the other cores to the low criticality tasks. This solution has been proven to limit the impact of the shared resources. However, it is quite pessimistic due to the proprietary of one core, which can be not fully used, to the high criticality task. This solution showed similarities with the static allocation of cache memory (risk of over-allocation).

Another type of solution, that we call "all-or-nothing" is to execute the high criticality task while we deactivate all the low criticality tasks. Such is the case of the works of Kritikakou and others. [6] which shows how to guarantee the real-time constraint with a two-step method. First, by executing tasks with different levels of criticality in parallel. Then, by deactivating the low criticality tasks based on a computation of the RWCET (remaining worst-case execution time). Other works of Kiritikakou and others. [7] and Girbal and others. [8] uses this same method. In our experimental approach, we want to be less pessimistic and keep the low criticality tasks activated.

Finally, a solution proposed by Xu and others. [9] uses CAT technology to allocate cache memory on Virtual Machines. The number of Virtual Machines is equal to the different levels of criticality. Then, tasks are executed on these Virtual Machines depending on the level of criticality. The disadvantage of this method is that the low criticality tasks will never use 100 % of cache memory due to pre-reservation.

V. Conclusion

Our experimental protocol aims to analyze the dynamic allocation mechanism of the last lever cache, with an experimental computer of two intel processors, using Linux. This experience allows us to decide on the efficiency of this kind of mechanism using a real-time task with a period of around 100ms. Furthermore, our approach is meant to limit the impact of the dynamic allocation on the low criticality task regarding the computing power. At the moment, we are developing dynamic allocation tools for our experimental platform.

CAT has a limited precision when it comes to ensuring the allocation of ways of the processor's last level cache.

The technology used for scheduling, which is Linux EDF, exclude the possibility to manage the core affinity for specific task.

About the first limitation due to CAT technology. If the dynamic allocation is efficient, then we will circumvent this issue by affecting different allocations over the time period of our experimentation. More specifically, we will allocate the sufficient ways for a task during its time execution only and then reallocate the same ways to another task when the first one is over. Thank's to that, we can artificially increase the size of the last level cache available for the tasks.

Regarding the second limitation, Linux is going through a process of adding real time assets on it's new Kernel. We therefore expect this limitation to be resolved in a near future.

After obtaining the experimental results, our outlooks are to add other high criticality tasks in our system as a first step. Then, to use more than one core for the high criticality tasks as a second step.

References

[1] C. Cullmann, C. Ferdinand, G. Gebhard, D. Grund, C. Maiza, J. Reineke, B. Triquet, and R. Wilhelm, "Predictability considerations in the design of multi-core embedded systems," 05 2010, pp. 36–42. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.172.4533&rep=rep1&type=pdf

[2] A. L. Lelli Juri, Scordino Claudio and F. Dario, "Deadline scheduling in the linux kernel," *Software practice and experience*, vol. 46, 2015. [Online]. Available: https://onlinelibrary.wiley.com/doi/epdf/10.1002/spe.2335

[3] K. Nguyen, " cat cache allocation technology " 2016. [Online]. Available: https://software.intel.com/content/www/us/en/develop/articles/introduction-to-cache-allocation-technology.html

[4] "Bpf compiler collection (bcc)," 2021. [Online]. Available: https://github.com/iovisor/bcc

[5] N. Suzuki, H. Kim, D. d. Niz, B. Andersson, L. Wrage, M. Klein, and R. Rajkumar, "Coordinated bank and cache coloring for temporal protection of memory accesses," in *2013 IEEE 16th International Conference on Computational Science and Engineering*, 2013, pp. 685–692.

[6] A. Kritikakou, T. Marty, and M. Roy, "DYNASCORE: DYNAmic Software COntroller to Increase REsource Utilization in Mixed-Critical Systems," *ACM Tran. on Design Automation of Electronic Systems*, vol. 23, 2017. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3149546.3110222

[7] A. Kritikakou, C. Rochange, M. Faugère, C. Pagetti, M. Roy, S. Girbal, and D. G. Pérez, "Distributed run-time wcet controller for concurrent critical tasks in mixed-critical systems," in *Proceedings of the 22nd International Conference on Real-Time Networks and Systems*, ser. RTNS '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 139–148. [Online]. Available: https://doi.org/10.1145/2659787.2659799

[8] S. Girbal, X. Jean, J. Le Rhun, D. G. Pérez, and M. Gatti, "Deterministic platform software for hard real-time systems using multi-core cots," in *2015 IEEE/AIAA 34th Digital Avionics Systems Conference (DASC)*, 2015, pp. 8D4–1 to 8D4–15.

[9] M. Xu, L. Thi, X. Phan, H.-Y. Choi, and I. Lee, "vcat: Dynamic cache management using cat virtualization," in *2017 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2017, pp. 211–222.

VECTOR

Model-based design of high-performance computer-based architectures

Speaker: Alexander Haliulin (alexander.haliulin@vector.com)
Author: Alexander Haliulin
Vector Informatik GmbH
Ingersheimer Str. 24, 70499 Stuttgart, Germany

Conference Domain: Processes, methods and tools, embedded computing platforms and networked systems
Conference Topics: Modes-based system engineering, embedded networks, service-oriented platforms
Keywords: Model-based development, high performance computer, automotive

Short paper

This paper focuses on a domain-specific architecture description language (ADL), that can be utilized in a model-based tool to design automotive Electric/Electronic-architectures. This ADL should be able to describe relevant aspects of modern E/E-architectures holistically with sufficient precision, clarity, and visual expressiveness to be understood by professionals of different disciplines in the automotive industry. Due to the pursued completeness of models, it should enable transformations into various standardized formats and allow metric calculation for optimization purposes. To cover innovations related to centralized computers in the vehicle, both regarding topology and implementation of service-oriented architectures (SOA), the problem of the design and deployment of applications in heterogeneous E/E-architectures on the system level is addressed. For this purpose, the existing domain-specific language Electric/Electronic-Architecture-ADL (EEA-ADL), used in the tool PREEvision, is extended. Furthermore, an integrated workflow using EEA-ADL is proposed, which could help establishing a comprehensive model of various deployment scenarios.

Introduction

Confronted with trends such as autonomous driving and connectivity, the industry is undergoing tremendous changes. Established E/E-architectures are not able to cope with emerging requirements for future automotive applications [1][2][3]. On the one hand, the amount and complexity of software functions are steadily increasing and require powerful computing resources and communication channels with higher bandwidths [1][4]. This results in the introduction of ethernet in automotive networks and centralized computer architectures. Functionality is removed from deeply embedded electronic control units (ECUs) and allocated on heterogeneous high-performance computers (HPCs) [1][5]. On the other hand, functions need to communicate to the outside world with offboard machines providing further services and data. The variety of protocols, that will be employed in these newly introduced communication paths, can only be guessed beforehand [5]. The structure of software architectures is changing as well. The conventional, predominantly signal-oriented manner of communication of automotive applications is complemented by the service-oriented paradigm of communication [6][7][8]. Consequently, future automotive E/E-architectures will be increasingly complex and versatile, thus presenting an obstacle to existing methodology and tooling, used for the design at system level [7].

The answer to system complexity may lie in a consequent use of a model-based approach throughout the development. Existing ADLs usually exhibit either a lack of expressiveness to depict complex software architectures or topology, or a lack of usability, by preventing clear graphical notation [17][18][19]. Therefore, they cannot constitute a model-based workflow in which E/E-architectures can be designed and described comprehensively on different levels of abstraction, establishing clarity in the overview of possible scenarios of software allocation on heterogeneous hardware platforms, the so-called “deployment problem” of software units.

Model-based approaches for handling the deployment problem

Several concepts of Model-Based System Engineering (MBSE) are known in the automotive sector [10][11]. They aim to provide a formal system description and refine it in various steps, allowing to derive further artifacts, preferably in an automated way. Known concepts include code generation, based on formal descriptions, and, by means of model transformations, derivation of the same model into different exchange formats [11], such as AUTOSAR XML. The model needs to be kept in a consistent state on all levels of abstraction. i.e., evolutions of requirements must necessarily lead to corresponding adjustments on dependent artifacts and vice versa [12]. Among the fundamental tasks to be performed by an appropriate model-based tooling, Rohdin et. al. points out notation of requirements, elaboration, and distribution of functions within a network topology [10]. A holistic model of the deployment on system level, may be achieved by a combination of tools or can be seamlessly

integrated into a single model-based tool [13][14][15]. The latter approach facilitates traceability between different abstraction layers and overcomes boundaries between tools. Further advantages and challenges of each of the alternatives are addressed by Herrmann et. al. [12].

MBSE needs to cope with a fundamental challenge to establish appropriate description languages. Hölldobler et. al. [16]. provide an overview of MBSE methods currently considered in research. A comprehensive overview of some of the most important existing and widely used ADLs is covered by Haber [17] and Wortmann [18]. Constantly, the dilemma of the expressiveness of a language is being approached. Generic ADLs can be extended, processed, and applied to different domains more flexibly. Opposed to them, domain-specific ADLs aim to capture systems of their domain, e.g., avionics or automotive, more precisely, at a cost of being difficult to maintain and to extend [16][18]. A notable automotive specific ADL is the Electronics Architecture and Software Technology (EAST-ADL). It provides a metamodel for system modeling covering the four abstract layers Vehicle, Analysis, Design, and Implementation. On the implementation level, however, EAST-ADL fully relies on the metamodel of the AUTOSAR standard, directly including it. There are several tools with different levels of maturity, which apply EAST-ADL. Another integrated approach is proposed by Kugele et. al., who describe an optimization algorithm dealing with the deployment problem in a holistic architectural model, however operating on an own ADL, limited both in terms of software and topology expressiveness [13]. Considerable efforts have been made in graphical representation of models designed in domain specific ADLs [15][19][20]. However, conceptual completeness, usability and visual abilities of the approaches proposed so far are still not mature.

Comparable to EAST-ADL, EEA-ADL is a domain-specific language focusing on automotive and allows a holistic description of E/E-architectures including the notation of requirements, logical (or functional) architecture, software architecture, hardware architecture, network topology and wiring harness. Based on the Meta-Object Facility (MOF), it can be extended according to new requirements. *Figure 1* shows an exemplary formal abstraction of an ECU by means of the EEA-ADL.

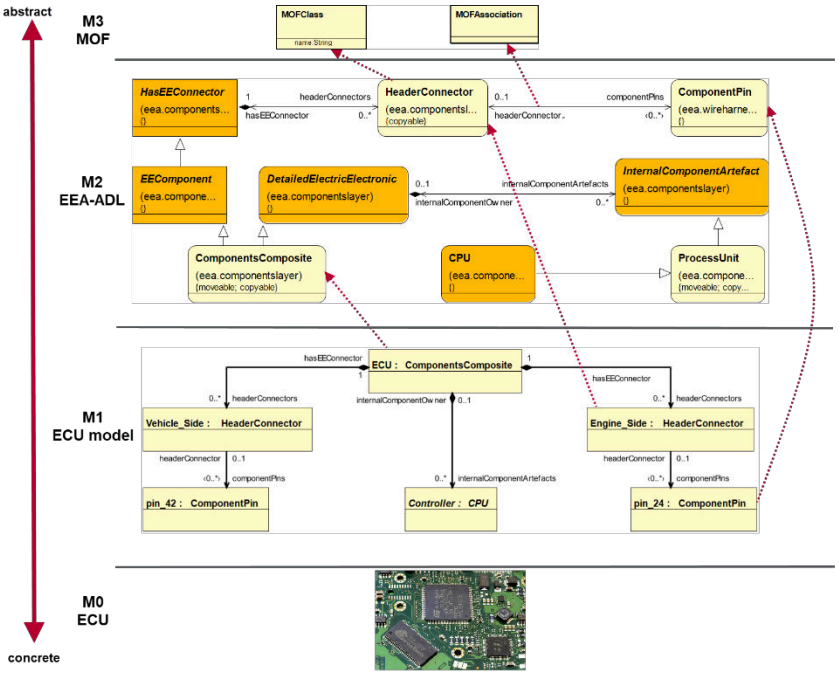


Figure 1: MOF-based approach of the EEA-ADL

It is arguable, which level of completeness and precision models need to be transformable into different formats or visualized in a tool in a comprehensible way. However, only upon a precise model, various metrics can be calculated on the same database providing necessary details for a certain system design scenario, such as bus load, costs, and length of the wiring etc. opening up various ways of optimization.

Modern E/E-architectures

An evolution towards centralized E/E-architectures can be observed in the automotive sector. *Figure 2* shows an overview of a modern architecture consisting of onboard and offboard participants, which can be classified into the following, hierarchically ordered, clusters [2]:

- Sensor/Actuator Layer (Commodity ECUs, Domain ECUs)
- Computing Layer (High Performance Computers)
- IT Backend Layer and External Devices (Offboard)

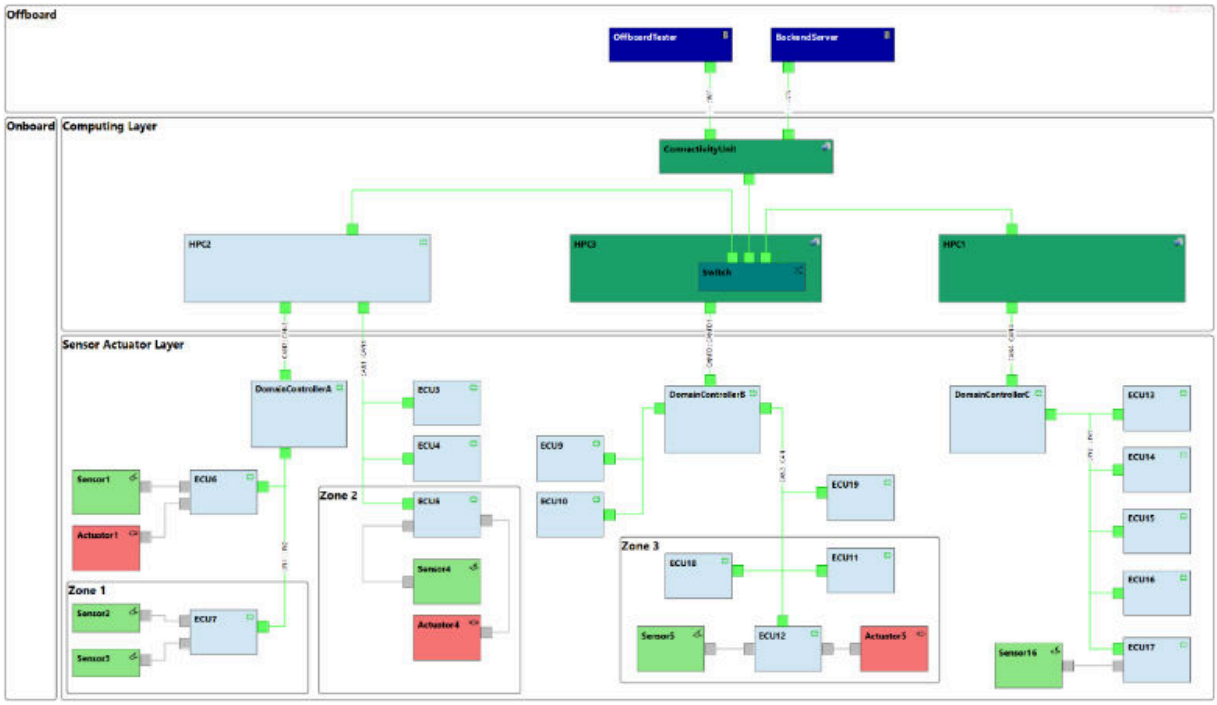


Figure 2: Network diagram of a modern E/E-architecture modeled in PREEvision

Onboard participants are ECUs of vastly differing complexity. While deeply embedded commodity ECUs are handling sensors and actuators directly, and executing simple control, diagnostics, or monitoring functions, domain ECUs integrate functionality of a specific functional domain. They communicate with commodity ECUs, which they are supervising, in a signal-based manner over field buses such as CAN or LIN [8]. Finally, HPCs establish computing platforms of the vehicle. The internal structure of an exemplary HPC is shown in *Figure 3*.They can communicate both in a signal-based way using legacy bus standards and in a service-oriented way via Ethernet. HPCs can combine (multicore)-microcontrollers alongside with microprocessors providing powerful computing resources for automotive applications of high complexity [2][3]. Both, as well as hypervisor technology, allow for several virtual environments with independent, e.g., POSIX-based, operating systems, to be executed on an HPC. Functionality from different domains can be deployed on such independent partitions, and while the layout and significant parts of an HPCs functionality will lie in the responsibility of an OEM, applications of certain partitions will potentially be supplied and developed independently [2]. Between partitions of an HPC, communication can be established by various kinds of inter-process communication (IPC) [8]. Lastly, HPCs can be connected via a connectivity unit to offboard servers, which also must be considered in the overall design [5]. All structural elements mentioned and depicted in *Figure 2* and *Figure 3* have been considered during the extension of the EEA-ADL.

Based on Ethernet, the paradigm of SOA is gaining in importance and must be supported during system level design. SOA treats software as encapsulated in services, i.e., sets of functionalities, provided, or consumed over defined interfaces. Such interfaces can integrate different communication paradigms like Remote Procedure Calls (RPC) or publish-subscribe patterns, and therefore need a specific generic notation, as well as the ability to be assigned to software units. SOA aims to make software development independent from the hardware. This is ensured by using a middleware which abstracts from a specific communication technology. Still, deployment

decisions must be taken during system design for the sake of optimal core assignment of applications. Since flexibility is a key benefit of SOA, enabling software updates during the whole lifetime of a vehicle [1], changes and extensions to services, addition of new services, additional service consumers etc. need to be supported by design tools in a flexible way and consistently aligned with the deployment decisions made.

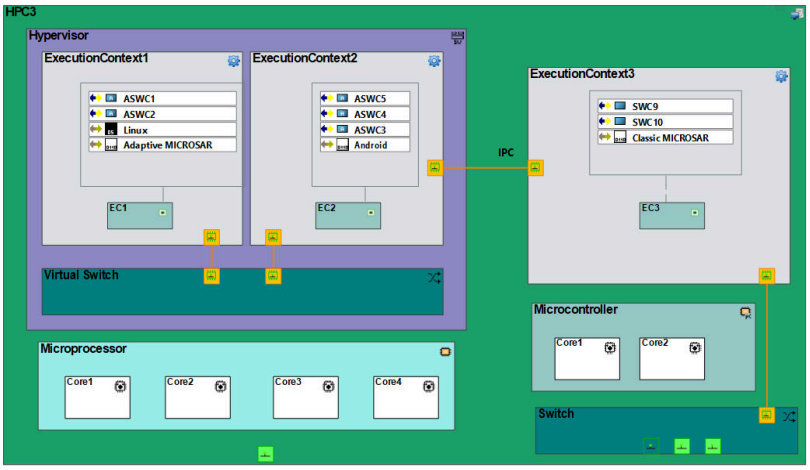


Figure 3: Component diagram of a HPC modeled in PREEvision

To describe a deployment scenario of software units in heterogeneous hardware architectures using EEA-ADL, a workflow involving several conceptual layers of the language, is proposed. A schematic of this workflow is shown in *Figure 4*:

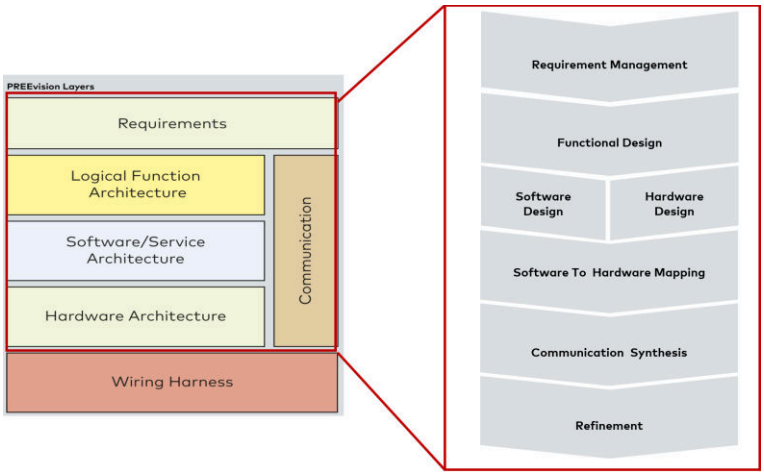


Figure 4: Workflow for completing deployment scenarios in relation to the layer architecture of PREEvision

In a first step, requirements are noted, providing input for a functional design. Based on this implementation-independent design, the software and hardware architectures are elaborated. The actual deployment decisions are made in a subsequent step by mapping of software to hardware. Different possible deployment scenarios can be evaluated using variants. On a holistic model of each deployment variant, formalized in EEA-ADL, various metrics such as static bus loads, costs etc. can be calculated. Based on the calculated metrics, a particular deployment scenario can be chosen. Further steps can include a potentially automated synthesis of communication artifacts and possible optional modeling steps, e.g. specification of communication characteristics, summarized as *refinement* in the workflow overview.

References

[1] AUTOSAR (2020): Explanation of Adaptive Platform Design – AUTOSAR AP R20-1
[2] Reinhardt, Dominik; Dannebaum, Udo; Scheffer, Michael; Traub, Matthias (2019): High Performance Processor Architecture for Automotive Large Scaled Integrated Systems within the European Processor Initiative Research Project
[3] Sommer, Stephan; Camek, Alexander; Becker, Klaus; Buckl, Christian; Zirkler, Andreas; Fiege, Ludger; Armbruster, Michael; Spiegelberg, Gernot; Knoll, Alois (2013): RACE: A Centralized Platform Computer Based Architecture for Automotive Applications
[4] Varas, Marcelino (2019): Service-orientierte Software-Architekturen: Brücken schlagen
[5] Tischer, Mirko (2018): Das Rechenzentrum im Fahrzeug.
[6] Heling, Günther(2019): Neue Herausforderungen für AUTOSAR
[7] Helmling, Markus (2017): Service-orientierte Architekturen und Ethernet im Fahrzeug: Auf dem Weg zum fahrenden Rechenzentrum.
[8] Oertel, Markus; Zimmer, Bastian (2019): E/E-Architekturen mit AUTOSAR Adaptive. Mehr Leistung, bitte!
[9] Rumez, Marcel; Grimm, Daniel; Kriesten, Reiner; Sax, Eric (2020): An Overview of Automotive Service-Oriented Architectures and Implications for Security Conutermeasures
[10] Rohdin, Martin; Ljungberg, Lars; Eklund, Ulrik (2002): A Method for Model Based Automotive Software Development
[11] Schlingloff, Holger; Conrad, Mirko; Dörr, Heiko; Sühl, Carsten (2014): Modellbasierte Steuergerätesoftwareentwicklung für den Automobilbereich
[12] Herrmann, Christoph; Krahn, Holger; Rumpe, Bernhard; Schindler, Martin; Völkel, Steven (2009): Scaling-Up Model-Based Development for Large Heterogeneous System with Compositional Modeling
[13] Kugele, Stefan; Pucea, Cheorghe (2014): Model-Based Optimization of Automotive E/E-Architectures
[14] Eder, Johannes; Bahya, Andreas; Voss, Sebastian; Ipatiov, Alexandru; Khalil, Maged (2018): From Deployment to Platform Exploration. Automatic Synthesis of Distributed Automotive Hardware Architectures
[15] About EAST-ADL http://www.east-adl.info/Specification.html (25.08.2021)
[16] Hölldobler, Katrin; Michael, Judith; Ringert, Jan Oliver; Rumpe, Bernhard; Wortmann, Andreas (2019) Innovations in Model-based Software And Systems Engineering
[17] Haber, Arne (2016): MontiArc – Architectural Modeling and Simulation of Interactive Distributed Systems
[18] Wortmann, Andreas (2016): An Extensible Component & Connector Architecture Description Infrastructure for Multi-Platform Modeling
[19] Lu, Jinzhi; Wang, Jiquang; Chen, Dejiu; Wang, Jian; Törngren, Martin (2018): A Service-Oriented Tool-Chain for Model-Based Systems Engineering of Aero Engines
[20] Grönniger, Hans; Hartmann, Jochen; Krahn, Holger; Kriebel, Stefan; Rumpe, Bernhard (2009): View-Based Modeling of Function Nets

Towards Model-Based Support for STPA as a Capella Add-On

Olivier Constant, Emmanuel Ledinot, Jérôme Le Noir
Thales Research & Technology, 1 av. Augustin Fresnel, 91120 Palaiseau, France

Category: Short paper
Keywords: Safety, MBSA, STPA, STAMP, MBSE, Arcadia, Capella

Introduction

STPA (System-Theoretic Process Analysis) [1][2] is a risk analysis method that considers Safety as a dynamic control problem. It allows identifying, in particular, problems or hidden assumptions in the design of a system that may lead to hazards through inappropriate interactions, even in the absence of breakdowns and cascading failures. By considering sociotechnical control structures, it allows the scope of the analysis to cover technical concerns as well as human factors and organizational issues.

As the evolution of the nature and complexity of systems tends to increase the possibility of inappropriate or badly understood interactions – due to the emergence of Systems of Systems, the increase of autonomy favored by progress in Artificial Intelligence, or the pervasive presence of software and connectivity, for example –, the focus and scope of STPA seem all the more relevant.

Model-Based Systems Engineering (MBSE) provides well-established engineering practices and tools that help design complex systems. By primarily working on models rather than documents, engineers benefit from consistent views, validation, traceability, impact analysis and many other features that contribute to increase trust in the quality of the design.

When applying STPA on the design of a safety-critical system, the presence of an MBSE context is thus likely. Going further, expressing an STPA analysis as (a part of) a model makes it possible to smoothly integrate it with other engineering artefacts and benefit from model-based features that help strengthen its contents. That, however, requires dedicated tool support.

We have investigated tool support in the form of an add-on to the open source Capella MBSE tool [3]. This paper describes the rationale we have followed to design and develop the tool and its main capabilities. The tool is currently in an evaluation and improvement phase.

Rationale

STPA takes as input information about the goals of the system, its intended environment and usage, its design at some level of abstraction. Among other things, this information is used to elaborate a sociotechnical Hierarchical Control Structure which identifies control relations and loops – internal or external to the system – that functionally contribute to the preservation of safety constraints. The method then challenges how critical and trustable these control loops are. As

outputs, STPA provides additional requirements and design constraints, and explicit usage restrictions. STPA is thus both a consumer of, and a contributor to, information that is at least partially present in a typical system architecture model. Support for traceability, consistency, impact analysis and version control is thus of particular relevance.

Capella is an open source model-based system architecture tool. Based on the Arcadia method [4], it proposes complementary modeling "perspectives" to clarify and analyze the needs (operational and system analyses), and elaborate a corresponding solution (logical and physical architectures). Roughly, every perspective is concerned with functional aspects – what has to be performed – and structural aspects – what are the entities or components involved, what functions are allocated to them, how they interact.

Depending on the objectives of the modeling activity and project constraints, perspectives do not necessarily need to be elaborated in their "natural" order nor with the same degree of detail. Every perspective is mostly independent in the sense that when an entity appears in different perspectives, it is technically represented by different model elements. Those elements are bound by traceability links which, all together, form the basis for global consistency checking. The cost of using links is mitigated by tool support that helps synchronize perspectives.

We judged this approach relevant for STPA because an STPA analysis sheds its own light on the system in terms of concerns and abstractions, while it is clearly related to the other perspectives. This approach also provides flexibility in the modeling process and supports "standalone" STPA analyses in the absence of system architecture models. So we developed tool support for STPA similarly to a perspective. A different concern or analysis may require a different approach [5].

While we thus opted for a loose integration on methodological aspects, we chose a tight integration from a technological point of view. We heavily relied on the technological stack of Capella in order to benefit from model-based capabilities and features that are provided by its constituent open source Eclipse technologies and by Capella itself.

Although we obviously had to define a metamodel for STPA [6] and design diagrams and tables [7], a number of features worked essentially out of the box: model/diagrams/tables synchronization, model navigation, basic validation, impact analysis, mass visualization and edition, user-defined queries, diff/merge, version control with Git, reuse by duplication and synchronization ("REC/RPL"). Other features like simplified navigation ("Semantic Browser"), methodological guidance ("Activity Explorer") or HTML documentation generation were obtained through dedicated customizations. All these features contribute to obtaining a full-fledged, albeit experimental, model-based engineering environment supporting STPA.

Related Work

A number of tools supporting STPA are available [8]. Among those that integrate into modeling environments, CAMET/SESSAF [9] is a commercial tool which extends OSATE [10] and its modeling language AADL. Similarly, SAHRA [11] is based on SysML in the Enterprise Architect environment. Other proposals have been made to integrate with SysML [12][13] according to certain

methodological principles. At a sole language level, the Object Management Group is working on RAAML [14], a new specification that extends SysML to cover STPA concepts, among others. It is in beta stage at the time of writing.

Our approach differs from the ones above in that we are not trying to integrate STPA into an existing modeling language. By considering STPA as an independent perspective on system architecture, we impose very few assumptions and constraints on modeling processes and methodology while trying to keep the STPA language (metamodel) as simple as possible to use.

XSTAMPP [15] and STAMP Workbench [16] are open source tools whose technological stack share a small subset with Capella's. Nonetheless, that part is too small to realize the tight integration mentioned above. Finally, ANSYS presented work in progress [17] that seems closer in spirit to our approach, but it has not been released as a product yet, to our knowledge.

Tool highlights

The illustrations below come from the case study of a delivery drone introduced in [18]. All STPA steps are covered. Figure 1 shows tool support for methodological guidance (Activity Explorer) and the edition of hazards. The edition of losses, safety constraints and loss scenarios is similar. We also found useful to add a table for the elicitation of stakeholders and stakes.

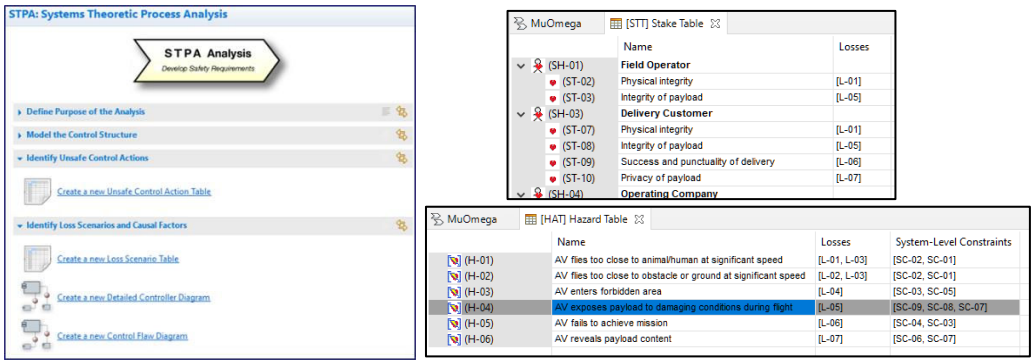


Figure 1: STPA steps in the Activity Explorer, edition of stakes, edition of hazards

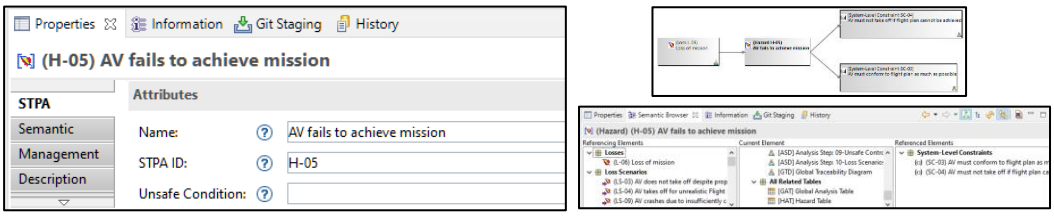


Figure 2: Properties, local graphical view, relation view (Semantic Browser) of a hazard

Figure 2 shows the editable properties of a hazard, its relations with other elements in a diagram and in tree widgets. More globally, the influence and justification of every element in the whole analysis can be visualized (impact analysis, Figure 3). Figure 3 and Figure 4 illustrate the edition of hierarchical control structures, the edition of unsafe control actions and the analysis of causal factors. Figure 5 - left-hand side shows the details of a controller with its process model, responsibilities and mapping to elements of the Logical Architecture perspective (blue squares).

Since the analysis led to the addition of constraints as countermeasures on the controller, and the controller is mapped to a component in the Logical Architecture, that logical component automatically inherits those constraints. This is demonstrated in Figure 5 - right-hand side, where "STPA Design Constraints" appears in the bottom left-hand corner when the component is selected.

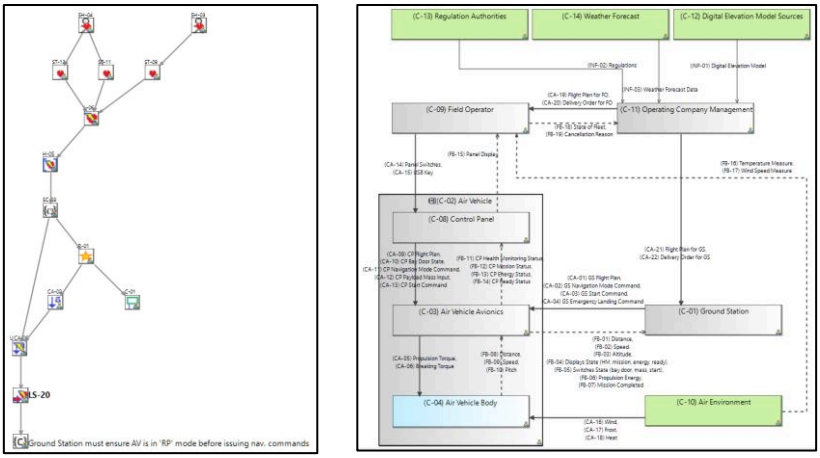


Figure 3: Impact analysis (of a loss scenario in this case), hierarchical control structure

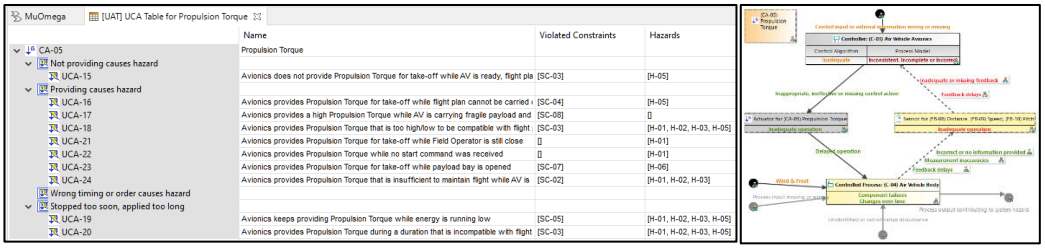


Figure 4: Edition of unsafe control actions, study of causal factors

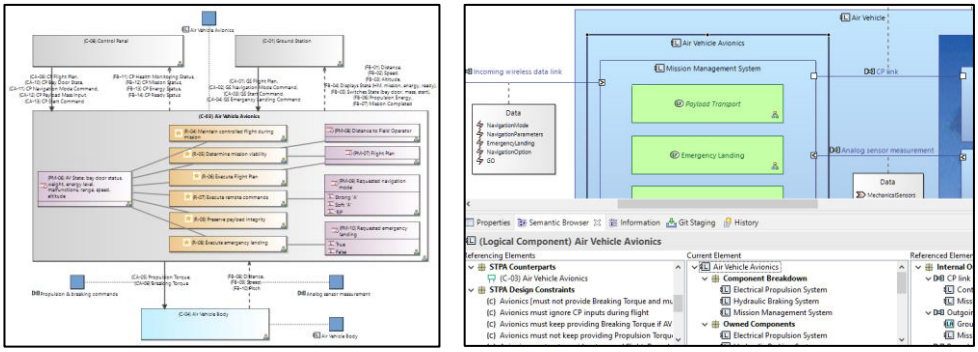


Figure 5: Focus on a controller, impact on the Logical Architecture

Conclusion

We have designed model-based tool support for STPA with the intent that it benefits from a tight technological integration while leaving freedom w.r.t. its methodological integration with system architecture. We see it as a first step enabling practitioners to carry out significant experiments, with the objective to gain maturity on possible design processes involving design iterations and parallel work with perspective synchronization and consistency checking.

Acknowledgements

We thank Juan Navas and his team at Thales Global Services for all the fruitful discussions.

References

[1] Leveson, N.G. *Engineering a Safer World: Systems Thinking Applied to Safety* (2011). MIT Press.

[2] Leveson, N.G., Thomas, J.P. *STPA Handbook* (2018). https://psas.scripts.mit.edu/home/get_file.php?name=STPA_handbook.pdf.

[3] Capella MBSE Tool. https://www.eclipse.org/capella/

[4] Voirin, J.-L. *Model-Based System and Architecture Engineering with the Arcadia Method* (2017). ISTE Press.

[5] Navas, J., Voirin, J.-L., Paul, S., Bonnet, S. *Towards a Model-Based approach to Systems and Cyber Security co-engineering* (2019). INCOSE International Symposium 29(1):850-865.

[6] Steinberg, D., Budinsky, F., Paternostro, M., Merks, E. *EMF: Eclipse Modeling Framework*, 2nd Edition (2008). Addison-Wesley Professional. https://www.eclipse.org/modeling/emf/

[7] Sirius technology. https://www.eclipse.org/sirius/

[8] MIT Partnership for Systems Approaches to Safety and Security (PSASS): STAMP Tools. http://psas.scripts.mit.edu/home/stamp-tools/

[9] Adventium Labs. *How to Conduct Safety Analysis with SESSAF* https://youtu.be/V1fsUdWluCw

[10] OSATE: Open Source AADL Tool Environment. https://osate.org/

[11] Krauss, S., Rejzek, M., Hilbes, C. *Tool qualification considerations for tools supporting STPA*. Procedia Engineering 128 (2015) 15-24. 3rd European STAMP Workshop (2015).

[12] Hurley, M., Wankel, J. *Safety guided design using STPA and model-based system engineering (MBSTPA)*. STAMP Workshop (2019).

[13] Rey de Souza, F.G., de Melo Bezerra, J., Hirata, C.M., de Saqui-Sannes, P., Apvrille, L. *Combining STPA with SysML Modeling*. 14th Annual Systems Conference SYSCON (2020). https://hal.telecom-paris.fr/hal-02933575/document

[14] Object Management Group. *Risk Analysis and Assessment Modeling Language (RAAML) Libraries and Profiles*, Version 1.0 - beta 1 (2020). https://www.omg.org/spec/RAAML/1.0/Beta1/PDF

[15] Abdulkhaleq, A., Wagner, S., *XSTAMPP: An eXtensible STAMP Platform as Tool Support for Safety Engineering*. STAMP Conference (2015). https://github.com/SE-Stuttgart/XSTAMPP

[16] STAMP Workbench (IPA – Information-technology Promotion Agency). https://www.ipa.go.jp/english/sec/reports/20180330.html

[17] Gabriel, N., Holz, E. *SOTIF and FuSa STPA for a Highway Pilot Function of a Passenger Car*. European STAMP Workshop and Conference (2020). https://warwick.ac.uk/fac/sci/wmg/mediacentre/wmgevents/stamp/wmg_eswc_day_2.pdf

[18] Ledinot, E. *CPS Engineering: Gap Analysis and Perspectives* (2021). CoRR abs/2104.13210.

PLATO N-DPU ON-BOARD SOFTWARE: AN IDEAL CANDIDATE FOR MULTICORE SCHEDULING ANALYSIS

Philippe Plasson [1], Gabriel Brusq [2], Frank Singhoff[3], Hai Nam Tran[3], Stéphane Rubini[3], Pierre Dissaux [4]

[1] LESIA, Observatoire de Paris, Université PSL, CNRS, Sorbonne Université, Université de Paris, 5 place Jules Janssen, 92195 Meudon, France, philippe.plasson@obspm.fr
[2] CNES, 18 avenue Edouard Belin 31400 Toulouse France, gabriel.brusq@cnes.fr
[3] Lab-STICC, CNRS UMR 6285, Univ. of Brest, 20, av Victor le Gorgeu, 29200 Brest, France, surname@univ-brest.fr
[4] Ellidiss Technologies, 24 quai de la douane, 29200 Brest, France, pierre.dissaux@ellidiss.com

INTRODUCTION

Each day, space missions require ever more computing power. PLATO mission, parts of ESA's Cosmic Vision program, is no exception. To answer the demand, multiprocessors chips are now used in space industry. In several ways it impacts space engineering habits. Architects have to rethink their software designs to benefit these extra processing resources, while considering shared accesses constraints. Their real-time applications are based on complex dynamic architectures and require a schedulability proof for flight. A model based analysis approach has been selected for PLATO N-DPU ASW. An AADL model representative of the application has been designed, and analysed through AADLInspector and Cheddar tools. In this paper, we present the PLATO N-DPU ASW architecture, and the LESIA GERICOS framework it relies on. We also discuss the model and the tools used to perform schedulability analysis.

1. PLATO MISSION PRESENTATION

1.1. PLATO mission goal

PLATO ("PLAnetary Transits and Oscillations of stars") is an M-class mission of the European Space Agency foreseen to be launched in 2026 [3]. PLATO aims to characterize exoplanetary systems by detecting planetary transits and conducting asteroseismology of their parent stars.
The PLATO payload concept is based on a multi-camera approach, involving a set of 24 normal instruments monitoring stars fainter than mV=8, plus a smaller set of two fast instruments observing extremely bright stars with magnitudes brighter than mV=8.

1.2. PLATO On-Board Data Processing System

The PLATO Data Processing System, called DPS, is the sub-system of PLATO payload module in charge of the on-board data processing (data acquisition, data reduction, data compression, monitoring, etc.) [4]. The DPS is a set of several on-board computer boards connected via a SpaceWire network. The DPS architecture is composed of:

- 12 Normal Data Processing Units (N-DPU) embedding a GR712RC dual-core LEON3FT SPARC V8 processor.
- two Fast Data processing Units (F-DPU)
- two Instrument Control Units (ICU) working in cold redundancy

The N-DPU and F-DPU are connected via SpaceWire interfaces to the front-end electronics of the normal (N-FEE) and fast (F-FEE) cameras at one end and to the ICU at the other.
The N-DPU and F-DPU are in charge of processing respectively the data of two normal cameras for each N-DPU and the data of one fast camera for each F-DPU.
The active ICU is responsible for managing the payload, communicating with the spacecraft service module, and compressing the scientific data before transmission as telemetry packets.

1.3. PLATO N-DPU Application Software Overview

The N-DPU Application Software (N-DPU ASW) is the embedded software deployed in each of the 12 N-DPU boards. Each N-DPU ASW manages two cameras (four 21-million pixel detectors per camera) and collects science and housekeeping data from their N-FEE. During the Observation mode, each N-DPU unit is receiving window segments of observed stars from both N-FEE as inputs. The window segments are transferred every 6.25 seconds through a single SpaceWire link per N-FEE. The window segments are then reconstructed back using a lookup table into windows and are processed by the N-DPU ASW.
The detector exposure time is 25 seconds, but the four detectors are read sequentially every 6.25 seconds with a read-out time of four seconds.
In the observation mode, the N-DPU ASW role is to produce 6x6-pixel square-shaped windows (i.e. raw star windows not transformed on-board) for 21% of the incoming stars (out of 132350 stars per camera) and photometry products using binary mask algorithms for 79% of them.
The photometry products are made up of star flux and centroids of stars. The N-DPU ASW computes also the offset, background, smearing and performs outlier detection.

The N-DPU ASW calculates light fluxes and centres of brightness (COB) every 25 seconds. These photometry products are then averaged over 50 and 600 seconds (samples of two and 24 measurements respectively). The averaged photometry is then sent to the ICU.
The other 21% stars (27500) of each camera are directly transmitted to the ICU.
In addition to the science services, the N-DPU ASW offers services dedicated to calibration activities: full-image acquisition service and window acquisition service.
Last but not least, the N-DPU ASW implements services for checking the telecommand packets, managing the production of housekeeping reports and event reports, handling the time, accessing to the N-DPU and N-FEE memories, managing and monitoring the on-board parameters, etc.

2. SOFTWARE FRAMEWORK AND DYNAMIC ARCHITECTURE

2.1. GERICOS framework

The architecture of the N-DPU ASW is built according to an Asymmetric Multi-Processing approach (AMP) using the GERICOS framework running on top of the RTEMS 4.8 real-time kernel (Edisoft version).
The GERICOS platform (GEneRIC Onboard Software), which is a generic platform for the development of space payload software, is made up of two parts:

- GERICOS C++ framework: a set of layered, reusable and customizable software components,
- GERICOS::TOOLS: a set of tools for automatizing the development process of embedded software (building chain, code generation, UML profile, UML diagram transformation, AMP support, …).

The GERICOS::CORE layer, which is part of C++ framework and which acts as a middleware, offers an extremely lightweight (small memory footprint), optimized (low CPU resources) and space qualified implementation of the active object paradigm on top of a real-time kernel.
With the GERICOS::CORE layer, a real-time application is built as a set of active objects (called "tasks"), each active object having its own message queue and its own computational thread. Each Thread processes incoming messages one by one by executing the function associated with the message [5].
The support of the AMP architecture in the GERICOS framework relies on the intrinsic features of the active object paradigm of the GERICOS::CORE layer. With this paradigm, two objects communicate via marshalled messages. Each object is split in an implementation object (which implements the services offered by the object) and in a stub object which is responsible for the marshalling aspects (message serialization, message unserialization). The marshalling process implemented in the stub has been extended so that the objects are able to communicate from a CPU core to another one using simple communication mechanisms based on shared memory for passing the messages and spin locks based on the LEON3 atomic compare-and-swap operation (CASA instruction) to make safe the inter-core concurrent access to the memory.
The GERICOS::CORE task components include functionalities for recording and reporting either response time statistics or execution time statistics of the various threaded functions. These features are used to assess by measurements the worst-case execution times (WCET) needed by the scheduling analysis model.

2.2. Dynamic architecture

The dual-core AMP architecture of the N-DPU ASW means that two applications coexist.
The first application, which acts as the master, is deployed on the LEON3-FT CPU core #0 and is in charge of managing the ICU interfaces (telecommand and telemetry packets), processing the data of camera #A, managing modes and technical services. This first application is made up of 21 tasks supporting 57 threaded functions. Among these threaded functions, only 23 have been identified as playing a key role in the computational model used for performing the scheduling analysis of the observation mode, which is the most demanding and critical mode of the software.
The second application is deployed on the LEON3-FT CPU core #1 and is in charge of processing the data of camera #B and scrubbing the memory. This second application is made up of 9 tasks supporting 35 threaded functions. Among these threaded functions, 13 have been identified as playing a key role in the computational model.
In terms of real-time sequencing, the fundamental period to consider is 6.25 seconds: the software receives every 6.25s a new set of star windows to process. The transmission of these star windows is spread over the 4.1 seconds corresponding to the readout of the detector. Up to 25 packets of 32 kilobytes are transmitted per second by the camera over the readout period. At the end of the packet reception, the software has to extract the window segments from the packets and reconstruct the windows in less than 2.15 seconds, that is to say before the end of the cycle of 6.25 seconds. Once the reconstruction phase is complete, the software can start the photometric processing. These photometric treatments are carried out during the transmission of the star windows of the next detector and must imperatively be finished 4.1 seconds after the starting of the following cycle. Figure 1 illustrates the sequence of the N-DPU ASW tasks with the main periods and deadlines.
This real-time sequencing and the related constraints are the same for both applications. The main difference is related to the periodic technical functions (data pool management, housekeeping packet production, monitoring of the parameters, transmission of telemetry

packets, etc.) which are not the same for both applications. These threaded functions execute periodically in the cycle of 6.25 seconds with a period that depends on the function (between 50 ms and one second).

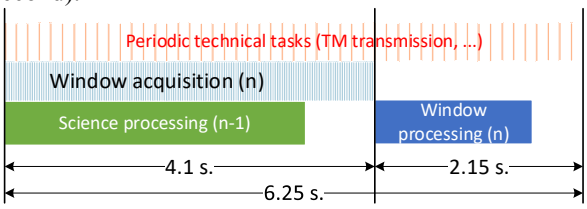


Figure 1: sequence of the N-DPU ASW tasks with the main periods and deadlines

During the observation mode, both applications communicate through the circular buffers containing the telemetry packets. The second application builds the telemetry packets corresponding to the data from camera B and stores them in a shared queue. The first application retrieves these packets from the shared queue and transmits them to the ICU. There can also be inter-core message exchanges when, for example, an event is detected by the second application and a report must be transmitted by the first application.

2.3. Real-time constraints

The N-DPU ASW is a real-time software where failure to conform to timing constraints can result in a loss of science data or of the instrument. Particular attention shall be paid to the window and science processing tasks which are considered critical for the scientific mission. A schedulability proof must be provided to validate that the dynamic architecture design prevents any deadline miss for those critical activities.

2.4. Dynamic model

The real-time application presented above is based on a complex dynamic architecture. A schedulability proof and CPU margins are required for flight in order to comply with ECSS standards. Such results are tricky to obtain because of the multi-core architecture. In particular, every access conflicts and locks must be taken into account. A model of the PLATO N-DPU ASW has been specifically designed to describe all its dynamic properties, and in particular, shared resources access, priority inheritance protocols, synchronization mechanisms, tasks precedence's constraints, WCET, …
Dynamic model design is a trade-off between model representativeness and model complexity. The goal is to propose a model using simplification hypothesis as little pessimistic as possible, to end up with sufficient representativeness to guarantee the schedulability analysis validity. In particular, some modelling hypothesis have been proposed for caches and DMA burst transfers. The model is populated thanks to real-time measurements performed on the on-board computer, via a dedicated GERICOS feature.
The next section details how the model of the application can be analyzed to get a schedulability proof and a worst-case CPU occupation ratio. These results are then confronted with the CPU margin requirements.

3. SCHEDULABILITY ANALYSIS

3.1. Cheddar Scheduling Analysis Tool

Cheddar is an open-source real-time scheduling analysis tool developed and maintained by the University of Brest and Ellidiss Technologies [1]. Recent improvements have been implemented to support scheduling analysis of software running on multi-core architectures [2]. Although Cheddar can be used in a standalone mode, i.e., using its own internal modelling language and editor, it is also semantically compliant with the SAE-AS5506 international standard (AADL: Architecture Analysis and Design Language) and can thus be easily integrated within interoperable systems and software development toolchains.

3.2. AADL Language

The AADL standard [7] has been defined to describe software intensive real-time system architectures and to embed a sufficient level of semantics to enable early analysis at model level. It is used worldwide for mission critical programs in the aerospace, ground transportation and medical device industries.
AADL is supported by a variety of modelling and analysis academic and industrial tools, including Osate [11], Ocarina [12], Masiw [13], Ramses [14], SCADE Architect, FASTAR, TASTE [15], Stood, and AADL Inspector. An alternate approach would have been to use the UML MARTE profile associated with the MAST scheduling analysis tool [8,9]. However, the strength of the AADL ecosystem [6] militates in its favour.

3.3. AADL Inspector Model Processing Framework

AADL Inspector (see Figure 2) is a model processing framework. It can load hand-written or automatically generated AADL models from modelling tools or inward model transformations. It provides dedicated outward model transformations to feed real-time, safety and security analysis tools. For this project, AADL Inspector has been used to load the hand-written AADL representation of the PLATO N-PDU ASW into Cheddar to perform the required multi-core scheduling analysis. A possible enhancement would consist in adding a custom model transformation to directly load the GERICOS UML model into AADL Inspector.

4. FEEDBACK FROM PLATO N-DPU ASW ANALYSIS

AADL offers a large panel of entities and properties to

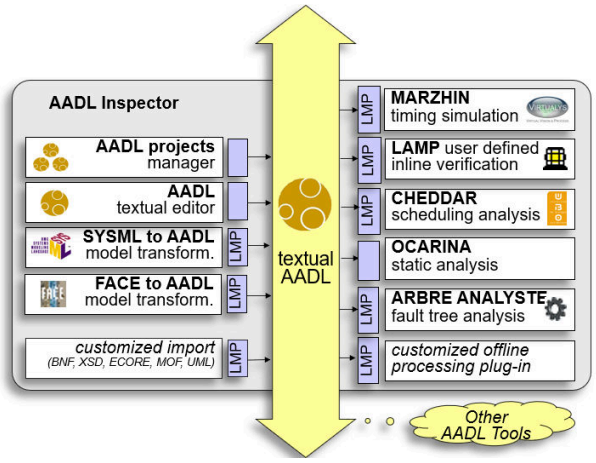


Figure 2. AADLInspector

model dynamic systems, that AADLInspector and Cheddar kernel tools can analyze. There are often several ways to model a dynamic behavior combining different elements of the language. For example, PLATO N-DPU ASW tasks synchronization is performed through messages exchanges. This behavior can be modeled in AADL by events sent through an event port, that can trigger the dispatch of a task. Another example is the modeling of task precedency by the dispatch offset property: to model that a task B starts after a task A, we can set a dispatch offset property for task B with a duration equal to task A period. The challenge is to find a valid model which is a trade-off between model design complexity and its pessimism. Assumptions can make the model more pessimistic, but also easier to design, and sometimes easier to analyze. Another solution is to tune the dynamic architecture model to make it compliant with schedulability analysis. For instance, the task set periods have been reworked to decrease as much as possible the Least Common Multiple (LCM) in order to decrease the time required to perform the scheduling analysis.

Some features that were missing to analyze the complete PLATO N-DPU ASW, such as the management of resources shared between cores, have been quickly integrated in AADLInspector and Cheddar tools. Then, the access to those resources can be accurately represented in AADL combining a resource protected by a spinlock (blocking statement) for core access, and a semaphore protected against deadlock by a priority inheritance protocol for task accesses (See Figure 3).

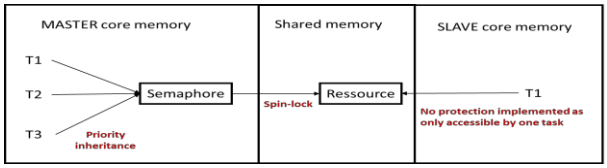


Figure 3. Modeling of core shared resources

For PLATO ASW, the memory accesses slowdown caused by DMA transfer has been computed based on SDRAM burst accesses configuration. A DMA burst access consumes 12 cycles every 153 cycles. It represents 7,84% of the SDRAM access cycles. In the worst case, 3 DMA transfers can occur at the same time, leading to a 24% slowdown ratio. This ratio is added to each task for which the WCET has been measured out of any DMA perturbation.. Conversely, some WCET are obtained from measures on worst case scenarios, and already take into account the DMA impacts. In this case the slowdown ratio is not applied. Regarding caches, no theoretical impact has been computed. Instead, the impact is also taken into account when measuring WCET on worst case scenarios.

The schedulability proof can be obtained with Cheddar by static analysis or by simulation performed on the feasibility interval, which is the interval of time that captures all possible events of the analyzed model. It can be computed with the following rule [10]:
Feasibility interval = [0 .. 2 * LCM + max (first dispatch time)] = 54 second sfor PLATO N-DPU ASW.
Both methods allow to check that the WCRT are always below the deadline, which is the expected schedulability proof.

CONCLUSION

In this paper, we presented how an AADL model of the PLATO N-DPU ASW system has been designed for schedulability analysis. The model has proved than all the tasks of the system, and in particular the more critical ones, theoretically cannot miss their deadlines. Execute window processing has a deadline of 2150 ms, and a computed WCRT of 1552ms on core 0 and 1142 ms on core 1. Execute science processing has a deadline of 4100 ms, and a computed WCRT of 3616 ms on core 0 and 2699 ms on core 1. Moreover, we were able to compute CPU margins from the model. Since the model is significantly pessimistic, due to hypothesis we took, we are confident that real CPU margins are above the one computed from the model.
The dynamic model helped to improve the dynamic architecture of the on-board software, in particular since the design of the model requires a flawless understanding of the real-time architecture and constraints. Outputs of analysis tools were precious to find real time optimizations. For instance, improvements were made by relaxing unjustified too hard timing constraints, or on priority and period assignments. Finally, WCET measurement mechanisms have been improved to be as less pessimistic as possible. Once an optimal dynamic architecture was reached, the model was used to get the analytical proof of the schedulability thanks to dedicated tools. Moreover, the model is helpful to validate any dynamic architecture evolution before its software implementation, and to continuously monitor the schedulability until the software design is frozen.

REFERENCES

[1]. Singhoff, F., Legrand, J., Nana, L., & Marcé, L. Cheddar: a flexible real time scheduling framework. In Proceedings of ACM SIGAda international conference. 2004, November, pp. 1-8

[2]. Rubini, S., Fotsing, C., Singhoff, F., Tran, H. N., & Dissaux, P. Scheduling analysis from architectural models of embedded multi-processor systems. ACM SIGBED Review, 11(1), 68-73. 2014

[3]. Ragazzoni, R., Magrin, D., Rauer, H., Pagano, I., Nascimbeni, V., Piotto, G., Piazza, D., Levacher, P., Schweitzer, M., Basso, S., Bandy, T., Benz, W., Bergomi, M., Biondi, F., Boerner, A., Borsa, F., Brandeker, A., Br• andli, M., Bruno, G., Cabrera, J., Chinellato, S., Roche, T. D., Dima, M., Erikson, A., Farinato, J., Munari, M., Ghigo, M., Greggio, D., Gullieuszik, M., Klebor, M., Marafatto, L., Mogulsky, V., Peter, G., Rieder, M., Sicilia, D., Spiga, D., Viotto, V., Wieser, M., Heras, A. M., Gondoin, P., Bodin, P., and Catala, C., PLATO: a multiple telescope spacecraft for exo-planets hunting, in [Space Telescopes and Instrumentation 2016: Optical, Infrared, and Millimeter Wave], MacEwen, H. A., Fazio, G. G., Lystrup, M., Batalha, N., Siegler, N., and Tong, E. C., eds., 9904, 731-737, International Society for Optics and Photonics, SPIE (2016).

[4]. Ziemke, C., Witteck U., Peter G., Plasson P., Galli E., Ulmer B., Ottensamer R., Ottacher H., Windsor J. PLATO DPS: State of the art on-board data processing for Europe's next planet-hunter. OBDP2021 - 2nd European Workshop on On-Board Data Processing. 2021, June.

[5]. Plasson, P., Cuomo, C., Gabriel, G., Gauthier, N., Gueguen, L., Malac-Allain, L. GERICOS: A Generic Framework for the Development of On-Board Software. In proceedings of DASIA Conference. 2016, May. ISBN: 978-92-9221-301-5. ESA-SP Vol. 736, 2016, id.39

[6]. Boydston A., Feiler P., Vestal S., Lewis B., Architecture Centric Virtual Integration Process (ACVIP): A Key Component of the DoD Digital Engineering Strategy. https://resources.sei.cmu.-edu/library/asset-view.cfm?assetid=634965

[7]. Feiler, Peter H., Bruce A. Lewis, and Steve Vestal. The SAE Architecture Analysis & Design Language (AADL) a standard for engineering performance critical systems. 2006 IEEE Conference on Computer Aided Control System Design.

[8]. Harbour, M. González, et al. Mast: Modeling and analysis suite for real time applications. Proceedings 13th Euromicro Conference on Real-Time Systems. IEEE, 2001.

[9]. Faugere, Madeleine, et al. Marte: Also an uml profile for modeling aadl applications. 12th IEEE International Conference on Engineering Complex Computer Systems (ICECCS 2007). IEEE, 2007.

[10]. Goossens, Joël, Emmanuel Grolleau, and Liliana Cucu-Grosjean. Periodicity of real-time schedules for dependent periodic tasks on identical multiprocessor platforms. Real-time systems 52.6 (2016): 808-832.

[11]. Osate, https://osate.org/

[12]. Hugues, Jerome, et al. From the prototype to the final embedded system using the Ocarina AADL tool suite. ACM Transactions on Embedded Computing Systems (TECS) 7.4 (2008): 1-25.

[13]. Khoroshilov, Alexey, et al. AADL-based toolset for IMA system design and integration. SAE International Journal of Aerospace 5.2 (2012): 294.

[14]. Blouin, Dominique, and Etienne Borde. AADL: A Language to Specify the Architecture of Cyber-Physical Systems. Foundations of Multi-Paradigm Modelling for Cyber-Physical Systems. Springer, Cham, 2020. 209-258.

[15]. Perrotin, Maxime, et al. TASTE: An open-source tool-chain for embedded system and software development. Embedded Real Time Software and Systems (ERTS2012). 2012.

Unboxing the Sand: on Deploying Safety Measures in the Programmable Logic of COTS MPSoCs

Sergi Alcaide[†], Guillem Cabo[†], Francisco Bas[†,‡], Pedro Benedicte[†],
Fabio Mazzocchetti[†], Francisco J. Cazorla[†], Jaume Abella[†]
[†]Barcelona Supercomputing Center (BSC)
[‡]Universitat Politècnica de Catalunya (UPC)

***Abstract*—The lack of sufficient hardware support for functional safety precludes the full adoption of many Commercial Off-the-Shelf (COTS) MPSoCs in safety-related systems, such as those in the aerospace industry. Some recent MPSoCs come along with programmable logic (PL), primarily intended to offload some specific complex functions that can be much more efficiently implemented in hardware than in software, hence being such PL a kind-of-sandbox fully mastered by ASIC cores outside the PL.**

This paper proposes using PL in those COTS MPSoCs to deploy the support needed to implement safety measures efficiently to enable the use of those MPSoCs for systems needing high assurance levels. Hence, the goal is not mastering PL from the cores solely, but also allowing PL to provide monitoring (e.g. contention, diversity, watchdogs) and control (e.g. configuring QoS features) capabilities to enable the realization of a safety concept atop. The early work presented in this paper already provides specific monitoring, diversity, and controlling strategies to allow PL take over safety-related functionalities.

***Index Terms*—safety, observability, controllability, MPSoC, programmable logic**

I. Introduction

Increased automation and autonomy in safety-related systems requires higher performance platforms able to execute those safety-critical tasks within tight time bounds. This can be achieved by using high-performance heterogeneous MultiProcessor Systems-on-Chip (MPSoCs) that include some form of accelerator (e.g. GPU, DSP, vector accelerator, and the like). For instance, platforms such as the NVIDIA Drive PX2 in the automotive domain [1], and the Xilinx Zynq UltraScale+ in the avionics domain [18] emerge as candidates to meet the corresponding performance goals.

Unfortunately, while those platforms provide the raw performance required, they challenge certification against safety standards due to their limited support to implement safety measures atop [10]. In particular, safety-related MPSoCs are generally expected to come along with native hardware support for independent watchdogs, diverse redundancy, error detection, etc. This is, for instance, the case for the Infineon AURIX processor family often used in the automotive domain [8], which on the other hand provides limited performance. Overall, end users face a conundrum between using platforms with appropriate hardware support to implement the safety measures needed at system level [9], [11], but insufficient performance, or using high-performance platforms lacking sufficient native hardware support to deliver mandatory safety measures.

As part of our recent work, we have devised and deployed a number of hardware components highly convenient to implement safety measures on top, as well as to improve testability, such as a multicore interference-aware statistics unit (SafeSU) [5], a module to enforce diversity across cores executing tasks redundantly (SafeDE) [2], and a programmable on-chip traffic injector to test timing and functional behavior during MPSoC validation and during operation (SafeTI) [12]. Those components are undergoing the final steps of their integration [7], [13] in commercial NOEL-V based MPSoCs for the space domain by Cobham Gaisler [6], and are offered as open source components [4]. However, those components rely on being integrated in MPSoCs during the design phases, hence with the ability to introduce some – yet limited – modifications related to the observability of some signals and control of some features.

Some MPSoCs, such as the Zynq UltraScale+ family, include some Programmable Logic (PL) as part of the SoC, typically intended to implement efficiently some functionalities, where the ASIC cores act as masters, and the PL as slave (e.g. working as an accelerator where cores offload some computation). In this context, the PL can be seen as a sandbox just responding to requests from the cores, where the latter truly exercise control over the MPSoC.

This paper contends that, enabling the use of high-performance MPSoCs for safety-related applications can be achieved by leveraging PL as a means to deploy hardware support to implement safety measures in COTS MPSoCs. In particular, we note that, if privileges (e.g. user mode, supervisor mode, etc.) are managed properly, functionalities in the PL can span beyond the sandbox by monitoring autonomously parts of the SoC and taking actions to control a subset of the MPSoC features. To illustrate this approach, in this work, and focusing on the Xilinx Zynq UltraScale+ MPSoC as a research vehicle, we show how safety-related hardware components can be deployed in the Zynq's PL to implement a number of safety measures such as (1) multicore interference monitoring building on the SafeSU [5], (2) support for diverse redundancy building on the SafeDE [2], and (3) support for device diagnostics building on the SafeTI [12], among other features.

The rest of the paper is organized as follows. Section II briefly introduces the Xilinx Zynq UltraScale+ MPSoC, as well as SafeSU, SafeDE and SafeTI. Section III presents the strategies being studied to allow the successful integration of the latter components (and some others to be developed) in the MPSoC. Finally, Section IV concludes this paper with a discussion on the forthcoming developments and opportunities emanating from this work.

II. Background

In this work, we analyze how to deploy safety-related hardware components in the PL of a MPSoC. Without loss of generality, we focus on a Xilinx Zynq UltraScale+ (ZUS for short) MPSoC. Hence, this section introduces the ZUS, as well as the already available safety-related components.

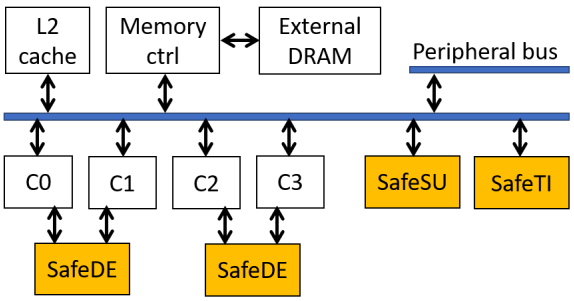


Fig. 1. Safety-related components as deployed in a NOEL-V MPSoC for the space domain.

A. Xilinx Zynq UltraScale+ MPSoC

The ZUS is a powerful MPSoC including a high-performance computing application cluster, referred to as APU, which includes 4 Arm Cortex A53 cores and a shared L2 cache. It also includes a real-time computing cluster, referred to as RPU, which includes 2 Arm Cortex R5 cores. Other computing elements such a GPU and PL are also present in the ZUS. Multiple memories and memory controllers are included in the ZUS, such as a DDR controller, an On-Chip Memory (OCM), and controllers to access flash memories. Multiple peripherals such as PCIe and Ethernet ports are also included. All those components are connected by means of a distributed network, so that traffic across different components can be fully segregated. For instance, independent routes exist from the APU to the DDR controller, from the PL to the OCM, and from the GPU to the DDR controller.

ZUS' interconnect builds on Arm components such as the CoreLink CCI-400 Cache Coherent Interconnect, and the CoreLink NIC-400 Network Interconnect, which include further Arm components, all of them implementing multiple and flexible QoS features, as shown in [14].

B. Hardware Components Supporting Safety Features

The work in this paper focuses initially in three already existing components supporting safety features, although we plan to develop and deploy additional ones. Those components, whose existing deployment in a commercial MPSoC is illustrated in Figure 1, are as follows:

SafeSU. The SafeSU statistics unit [5] includes observability and controllability channels to master multicore interference in MPSoCs. In particular, it collects statistics about how many cycles each master is delayed by each other master in an AMBA Advanced High-performance Bus (AHB) interface. Such information is particularly useful to diagnose timing overruns during operation and to optimize application deployment so that multicore interference is kept low.

The SafeSU also includes a multicore interference quota mechanism so that, if the observed interference caused by one master on another exceeds a user-programmed quota, an interrupt is raised. This allows limiting interference during operation.

Finally, the SafeSU includes specific logic to measure the highest latency experienced by a request in the AHB interface. Such information is collected per request type (e.g., read/write, burst/no-burst, etc.) and allows collecting maximum latencies used for Worst-Case Execution Time estimation, and also allows monitoring during operation whether latencies exceed a user-programmed threshold, which could be used as a form of watchdog.

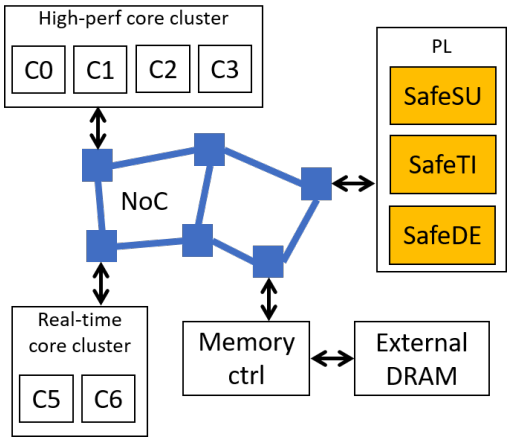


Fig. 2. Safety-related components deployed as proposed in this work in a Zynq UltraScale+ MPSoC.

SafeDE. The SafeDE is a module intended to enforce the staggered execution of a given task running redundantly in two cores. This feature is particularly useful to achieve some form of lockstepping (i.e. efficient diverse redundancy) in processors lacking it natively or, at least, lacking it for its highest performance cores. SafeDE collects the number of instructions executed by the cores running the redundant task, and whenever the advantage of the head core falls below a given threshold w.r.t. the trail core, SafeDE stalls the trail core for a while until the staggering is large enough.

SafeTI. The SafeTI is a sophisticated and programmable traffic injector able to inject specific traffic patterns whose elements are read and write operations, with parameterizable data transmitted, with user programmable source/destination, with a burst/no-burst parameter, with independent and programmable stalls between transactions, and with capability to store multiple independent or overlapping traffic patterns that may be used under different circumstances. As explained before, SafeTI is particularly adequate to test the platform during validation, as well as to test functionality and timing during operation.

While the ZUS MPSoC also includes its Xilinx AXI Traffic Generator (ATG) [17] in the PL, such IP is less flexible than SafeTI, and comes along a restrictive license, hence precluding its use in other platforms. Instead, SafeTI is provided under a highly-permissive open source license [12].

III. Deploying Safety Measures in the PL of the ZUS

Deploying hardware support to implement safety measures in a full-custom design, either deployed as an ASIC or as an FPGA product, provides flexibility to find the most efficient solutions. For instance, in the case of SafeDE, cores can be made to export some signals to let SafeDE easily monitor their progress and stall the trail core whenever needed. Analogously, SafeSU and SafeTI can be attached to any interconnect directly, hence achieving full observability of the on-chip traffic. However, if those components are deployed in the PL of a COTS MPSoC, observability and controllability channels are limited and cannot be changed. Hence, the challenge tackled in this work consists on how to deploy such hardware support

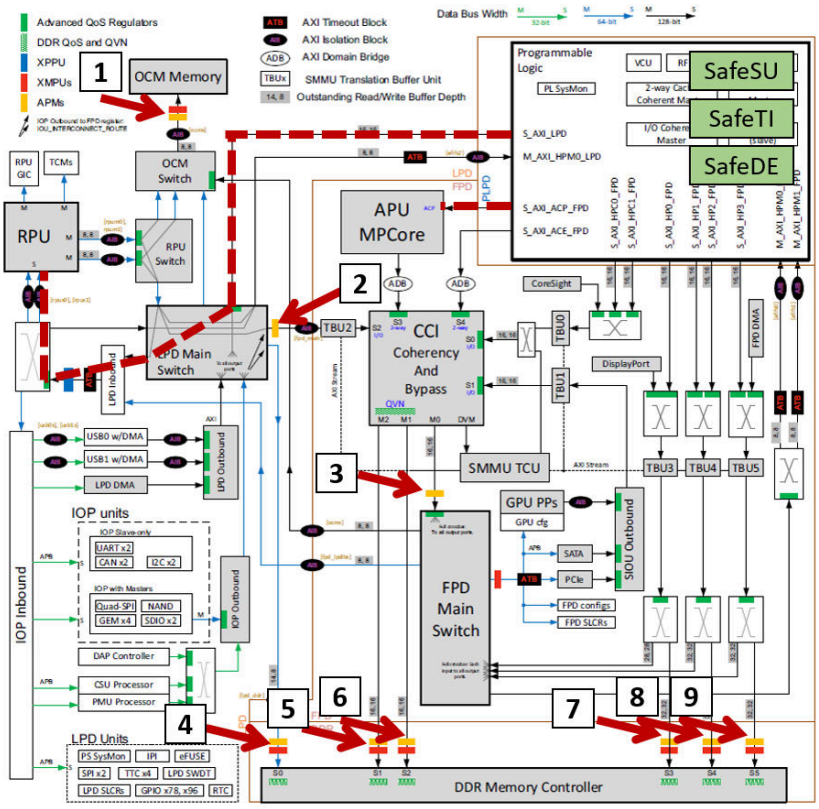


Fig. 3. Schematic of the ZUS MPSoC (baseline picture taken from [18]).

effectively in the PL, as illustrated in Figure 2 for the ZUS MPSoC.

A. Monitoring and Controlling Multicore Interference with SafeSU

Interference monitoring. The first relevant characteristic of the deployment of the SafeSU in the ZUS is that the ZUS has a distributed interconnect (see Figure 3), hence meaning that traffic is distributed rather than centralized in a single interconnect. This fact differs from previous deployments of the SafeSU, which built upon centralized interconnects (e.g. a bus) [15]. The second relevant characteristic is that the SafeSU cannot directly manage the AMBA AXI signals of the NoC, as it did with the AMBA AHB signals in its previous deployments. Overall, the SafeSU needs to collect NoC traffic information from remote locations and without direct access to protocol signals.

In order to properly integrate the SafeSU in the PL of the ZUS, we note that ZUS interconnects include AXI performance monitors (APM) [16], which gather transaction metrics such as the following:

- Read and write transaction counts.
- Read and write byte counts.
- Read and write latencies.
- Number of idle cycles caused by masters and slaves.
- Counts on some additional AXI-related protocol signals.

Moreover, APMs exist in different locations of the MPSoC, such as the interfaces near the DDR, the OCM, and the main switches connecting the APU and RPU computing units. They are shown as yellow squares in the ZUS schematic in Figure 3. They have been further indicated with thick dark red arrows and numbered. APM-1 monitors the OCM (aka as scratchpad memory). APM-2 monitors the traffic arriving from the Low Power Domain (LPD), where real-time cores are located, to the Full Power Domain (FPD). APM-3 monitors the traffic from the FPD, including APU cores, some of the PL ports and the GPU among others, to the LPD. APMs 4 to 9 monitor DRAM traffic for each of its different ports connected to the LPD, APU cores, PL, etc. In Figure 3, we have also included the SafeSU, SafeTI, and SafeDE in the PL for clarity. Hence, our current work focuses on interfacing APMs, as well as core-related memory access counters, to use their information to infer, either deterministically or statistically, how much interference each computing component has caused on each other. Moreover, the SafeSU needs to be extended to further break down interference across main locations to further ease diagnostics in case of deadline overruns.

Interference control. The SafeSU includes interference quota monitoring capabilities, and, upon a quota violation, it raises an interrupt. We note that the ZUS includes a wide variety of QoS knobs in its NoC interfaces, hence allowing to prioritize traffic based on its type and/or source. While not originally developed as part of the SafeSU, part of our work focuses on how to interface those QoS knobs so that the SafeSU can control them to limit specific interference channels whenever needed. This naturally needs being done in close collaboration with the Real-Time Operating System (RTOS), which should instruct the SafeSU on what knobs to set and how under quota violation scenarios, and must grant the SafeSU with appropriate privileges to change such configuration settings. Alternatively, the SafeSU can raise interrupts and be the RTOS the one in charge of configuring the QoS knobs as needed.

B. Enforcing Diverse Redundancy with SafeDE

Originally, the SafeDE has direct access to the instruction counts of the cores executing a task redundantly, so that it

can determine the staggering among them. SafeDE has also access to the stall signal of one of the pipeline stages of the trail core so that it can stop it almost immediately whenever the staggering is too low (e.g. below few cycles) [2].

In the context of the ZUS, the SafeDE can neither snoop instruction count registers nor control pipeline stall signals of the cores. Hence, alternatives are under consideration. Regarding instruction counts, we aim at, in cooperation with the RTOS, having means to read instruction counts from the *virtually lockstepped* cores as software would do. Note that paths to reach the cores in the RPU and APU from the PL exist in the platform. They have been indicated with dashed thick dark red lines in Figure 3. Hence, if the performance monitoring counters of interest are mapped into readable address spaces from the outside, and the RTOS programs privileges properly, SafeDE could read those counters from the PL and take an action whenever needed to preserve the staggering. Regarding stalling the trail core whenever needed, multiple alternatives are being considered such as:

- Modifying QoS knobs in the interconnect to favor the head core at the expense of slowing down the trail one. However, this is only effective if the task being run misses in local caches and accesses those interconnects.
- Issuing specific interrupts to the trail core so that, by programming them properly, the RTOS takes over for a short period intended to be enough to recover sufficient staggering between head and trail cores.

C. Diagnostics and Latency Measurements with SafeTI

The least impacted component due to being deployed in the PL of the ZUS is the SafeTI traffic injector since it issues transactions as programmed by the end user, regardless of whether the component is directly attached to the AMBA interface, or whether the interconnect is a bus or a NoC. Hence, integration of the SafeTI to generate traffic during operation for diagnostics purposes is not expected to bring major concerns.

We note, however, that the SafeSU may have difficulties to measure latencies of different request types by itself due to the lack of access to the AMBA interfaces, and due to the distributed nature of the ZUS' NoC. In this case, we plan to leverage the SafeTI to compensate that limitation since it can be programmed to produce specific traffic patterns triggering latencies that should allow building iteratively the latency to reach additional switches and components. Hence, by operating cooperatively, the SafeSU and the SafeTI will be able to measure the highest latencies experienced in most parts of the NoC.

D. Beyond Existing Components

Part of our ongoing work also includes the development of diversity monitors to complement SafeDE. A first version of those diversity monitors, referred to as SafeDM, has already been released [3]. Those monitors aim at measuring diversity across cores running redundant tasks, but not performing any control of their staggering. Such diversity is measured accessing pipeline information. However, it is still unclear how to measure diversity when pipelines are not visible. This is ongoing work for the ZUS which we expect to solve with as much precision as possible due to the limited observability of the cores.

Other components we intend to deploy in the PL of the ZUS relate to aliveness monitoring of different computing components with some form of watchdogs. Those will likely build on the instruction counts of the cores, and their NoC activity as primary sources of information related to aliveness.

Overall, the strategy is exploiting the (many) observability and controllability features of the COTS MPSoCs in general, and the ZUS in particular, with specific modules deployed in the PL to provide support for safety measures implementation.

E. Safety Considerations

By deploying safety measures in the PL of a COTS MPSoC, the set of assurance (integrity) levels that can be targeted are limited by the native safety support of the MPSoC itself. Hence, if the development process of the MPSoC does not adhere to the requirements of some assurance levels (e.g. DAL-A or DAL-B for avionics), then it is very unlikely that applications with safety requirements at those levels can be deployed on the MPSoC regardless the safety measures deployed in the PL. A sufficient assurance level must be attained at least for those parts of the MPSoC controlling the monitoring capabilities for fail-safe systems, and for those parts providing computing capabilities and monitoring capabilities for fail-operational systems. Else, external solutions may be required, such as the use of multiple MPSoCs with a sufficient degree of redundancy to meet the requirements of the highest assurance levels.

IV. Conclusions and Future Work

High-performance COTS MPSoCs needed for future aerospace systems lack sufficient native hardware support to implement efficiently many of the usual safety measures needed in those systems. We note, however, that some of those MPSoCs include a PL region which, despite generally intended to operate as a sandbox, can be used to deploy hardware components supporting safety features.

In this work, we review some existing such components and analyze how they could be deployed in the Xilinx Zynq UltraScale+ (ZUS) MPSoC, as representative example of high-performance COTS MPSoC, despite the gap existing between their original implementations and the observability and controllability channels available in the PL of the ZUS. In particular, we review the alternatives offered by the ZUS to monitor and control multicore interference with the SafeSU, to enforce diverse redundancy with the SafeDE, and to provide diagnostics with the SafeTI.

Our future work includes performing the integration of those hardware components in the ZUS to enable multiple safety measures in COTS MPSoCs, investigating additional hardware components that could be incorporated, and looking beyond the ZUS to consider even more powerful MPSoCs such as, for instance, the Xilinx VERSAL platform.

Acknowledgements

This work is part of the project PCI2020-112010, funded by MCIN/AEI/10.13039/501100011033 and the European Union "NextGenerationEU"/PRTR, and the European Union's Horizon 2020 Programme under project ECSEL Joint Undertaking (JU) under grant agreement No 877056. This work has also been partially supported by the Spanish Ministry of Science and Innovation under grant PID2019-107255GB-C21 funded by MCIN/AEI/10.13039/501100011033.

REFERENCES

[1] NVIDIA DRIVE PX. Scalable supercomputer for autonomous driving. http://www.nvidia.com/object/drive-px.html.

[2] F. Bas et al. SafeDE: a flexible diversity enforcement hardware module for light-lockstepping. In *2021 IEEE 27th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, pages 1–7, 2021.

[3] F. Bas et al. SafeDM: a hardware diversity monitor for redundant execution on non-lockstepped cores. In *2022 IEEE 25th Design, Automation and Test in Europe Conference (DATE)*, pages 1–6, 2022.

[4] BSC - CAOS. Safety-Related Hardware Components webpage. https://bsccaos.github.io.

[5] G. Cabo et al. SafeSU: an extended statistics unit for multicore timing interference. https://people.ac.upc.edu/jabella/ets21.pdf. In *IEEE European Test Symposium (ETS)*, 2021.

[6] Cobham Gaisler. NOEL-V Processor. https://www.gaisler.com/index.php/products/processors/noel-v.

[7] De-RISC Consortium. De-RISC website, 2021. https://www.derisc-project.eu/ (accessed Feb-2021).

[8] Infineon. AURIX Multicore 32-bit Microcontroller Family to Meet Safety and Powertrain Requirements of Upcoming Vehicle Generations. http://www.infineon.com/cms/en/about-infineon/press/press-releases/2012/INFATV201205-040.html.

[9] International Standards Organization. *ISO/DIS 26262. Road Vehicles – Functional Safety*, 2009.

[10] J. Perez-Cerrolaza et al. Multi-core devices for safety-critical systems: A survey. *ACM Comput. Surv.*, 53(4), 2020.

[11] RTCA and EUROCAE. *DO-178B / ED-12B, Software Considerations in Airborne Systems and Equipment Certification*, 1992.

[12] O. Sala et al. SafeTI: a hardware traffic injector for mpsoc functional and timing validation. In *2021 IEEE 27th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, pages 1–7, 2021.

[13] SELENE Consortium. SELENE website, 2021. https://www.selene-project.eu/ (accessed Feb-2021).

[14] A. Serrano-Cases et al. Leveraging Hardware QoS to Control Contention in the Xilinx Zynq UltraScale+ MPSoC. In *33rd Euromicro Conference on Real-Time Systems (ECRTS 2021)*, 2021.

[15] G. Wessman et al. De-RISC: the first RISC-V space-grade platform for safety-critical systems. In *IEEE Space Computing Conference (SCC)*, 2021.

[16] XILINX. AXI Performance Monitor LogiCORE IP Product Guide. PG037 (v4.0). 2013.

[17] Xilinx. *AXI Traffic Generator v3.0. LogiCORE IP Product Guide*, 2019.

[18] XILINX. Zynq UltraScale+ Device. Technical Reference Manual. UG1085 (v2.1). 2019.

Towards a Novel UAV Position Tracking and Reporting System for Very Low Level Airspace

Bruno Chianca Ferreira*, Guillaume Dufour† and Guthemberg Silvestre‡
*‡ENAC/ReSCo, †ONERA/DTIS
Université de Toulouse, France
Email: *bruno.chianca@enac.fr, †guillaume.dufour@onera.fr, ‡silvestre@enac.fr

Short paper about ongoing research.

***Abstract*—The integration of Unmanned Aircraft Systems into the airspace is a challenging, complex task for both operators and regulators. To ensure a safe and secure UAS integration, they should rely on wide variety of dependable sub-systems, including a tracking and position reporting system. This paper proposes a system architecture for tracking and position reporting of UAS in very-low level airspace. The system leverages low-latency communications and distributed computing to offer highly available, consistent tracking information. Preliminary results suggest that our approach is especially useful for the traffic management and operations in densely populated areas.**

***Index Terms*—utm, uas, edge, emulation, ads-b, mobile, ad-hoc**

I. Introduction

Recently, there has been increasing interest in Unmanned Aircraft Systems (UAS) for many real-life, civilian applications [1], including real-time monitoring, remote sensing, search and rescue, agriculture services and delivery of goods. In addition, a rapid growth in the number of drones in activity is expected, especially at very low-altitude of airspace with high demand in densely populated areas [2]. For instance, a recent study [3] forecasts that 98% of the aircraft operating in the airspace of Paris by 2035 will be autonomous. The integration of UAS will certainly result in fundamental changes of the air traffic management (ATM). Consequently, there is a need for the design of an Unmanned Traffic Management (UTM) system that will be able to manage a large number of autonomous vehicles in a safe and efficient manner.

One of the key components of the emerging UTM system is the the capability to track the vehicles' positions. Currently, tracking and position reporting of aircraft relies on long-range radio communications. Unfortunately, a large number of vehicles communicating through conventional radio frequencies (RF) in densely populated areas is likely to incur mutual interference and, consequently, poor performance. One way of reducing interference is to limit the range of the wireless radio but, which in conjunction with the limitation introduced by the buildings, it can render systems such as Automatic Dependent Surveillance–Broadcast (ADS-B) unfeasible. Furthermore, it remains unclear how the current ATM systems could be redesigned to enforce highly available, consistent position reporting to meet safety and performance requirements of novel UTM systems.

This study aims to investigate the feasibility of a novel tracking and position reporting system that could replace ADS-B in the context of UTM and UAS. The rational behind it is two-fold. First, the new system should allow small autonomous vehicles to communicate with low-cost, low-footprint, short range wireless interfaces and broadcast real-time data with local stations in the vicinity. Second, the system should rely on distributed computing techniques to share positioning data of vehicles with high availability and strong consistency, allowing even aircraft which are beyond line of sight to have updated information, ensuring proper functionality of their own applications.

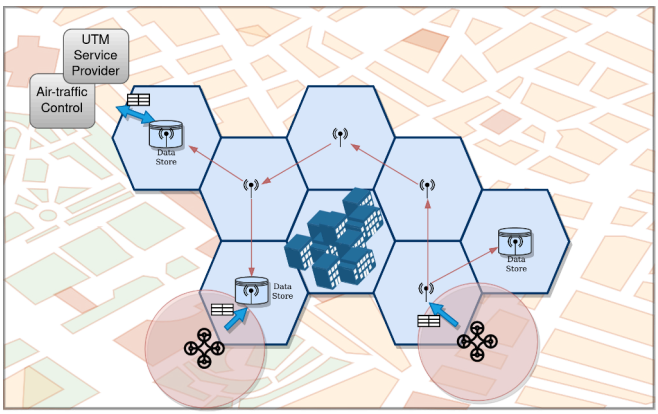


Fig. 1: Distributed UTM Data Service

In this paper, we introduce a system architecture to reach the aforementioned design goals by deploying a distributed tracking and position reporting system throughout stations of Radio Access Network cloudlets, as depicted in Figure 1. The architecture provides performance requirements, such as low-latency, high-throughput data exchange thanks to 5G's communication capability. To meet consistent, highly available positioning data sharing, our architecture leverages the functionalities of a strongly consistent distributed key-value store. Preliminary results based on mobile systems' emulation suggest that this architecture meets both performance and safety requirements of emerging UTM's tracking and position reporting system.

II. State of the Art

A. Unmanned Traffic Management

The rapid expansion of the market of small UAS has created new risks for civil aviation [4][3]. That increases the need for regulation and deployment of a fully functional UTM [5]. Some governmental agencies have been more prominent in UTM proposals, such as USA with NASA/FAA and European Union's SESAR/U-space. In [6], the author described NASA's research initiative related to the concept of operations (ConOps). NASA established a research platform which is used together with their partners to test and evaluate the challenges and solutions proposed for each of the technical capabilities associated with UTM. For NASA/FAA the UTM is set to evolve incrementally according to technical capability levels (TCL), and some research works have been conducted in order to verify the full development and verification of TCL levels. In [7] flight tests demonstrating the most basic TCL levels 1 and 2 were performed even with beyond visual-line-of-sight (BVLOS) for unpopulated areas. BVLOS in populated areas is introduced in [8], where the authors performed a flight demonstration of TCL3. In [9] the author demonstrates TCL4 capabilities using simulation tools, and gave a high emphasis in safety by describing in details the SAFE50 architecture.

The framework on how a UTM system should work and the interaction between the stakeholders is not yet finalized [10] and many aspects are not clear in order to achieve the highest levels of autonomy. However, the main proposed architectures are composed of distinctive applications [11] such as sense and avoid [12], [13], [14], [15], path planning [16], [17], [18], [19], which rely on the fact that trustable data can be shared among stakeholders with low latency.

B. Aircraft Surveillance

A position tracking and reporting system is a crucial system for aircraft surveillance. The Automatic Dependent Surveillance–Broadcast (ADS-B) is a service present in aircraft[1] that broadcasts relevant unencrypted data in real-time. It aids radars in keeping updated and correct information about all the aircraft currently flying. With such information about all aircraft, air traffic management (ATM) authorities can develop applications to improve security, and efficiency of air transportation system and airports. Albeit not mandatory like *ADS-B Out*, other aircraft can also receive broadcasts with *ADS-B In* and use the information to build knowledge about its surroundings in order to improve conflicts avoidance. Aircraft equipped with ADS-B equipment, periodically broadcast their call sign, latitude and longitude, ground speed and flight number with signals modulated in 1090 MHz. Considering the importance systems like ADS-B have for ATM, a similar technology is also expected to play an important role in UTM. Unlike large aircraft, small UAVs are harder to be detected with radars, increasing the importance of a *ADS-B like* system. Previous works have proposed ADS-B alternatives

[1]Mandatory in many countries depending on aircraft weight or airspace class.

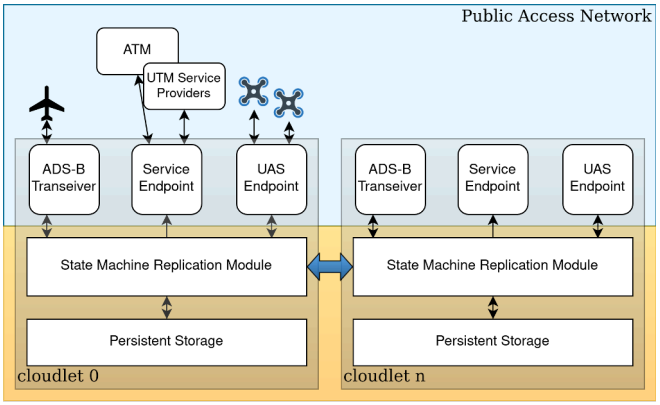


Fig. 2: Our System Architecture

in the context of UTM as explored in [20]. The authors in [21] and [22] proposed an architecture based in low-cost radio transferring data to a central cloud, which does not solve the problem of fast replication across the urban area for low latency and availability to all stakeholders. In [22] the author proposes an ADS-B like system for UTM based on APRS (Automatic Packet Reporting System), which is an approach different to what is proposed in this work, since it is based on relatively long range radio (40Km) with restricted throughput. In [13], the author proposed a solution to inject the position data in the SSID of WiFi and the results has shown that the proposed setup yielded reasonably low latency (under 125 ms) and acceptable throughput (4 messages/s) under an experimental setup. By having a secure ADS-B like data layer, applications such as presented in [15] where the authors proposes a sense and avoid system that used ADS-B data as input, can be built.

III. A Novel System Architecture for Tracking and Position Reporting

Our system architecture is designed to enable fault-tolerant and efficient data sharing for UTM stakeholders. The main blocks of this architecture are shown in Figure 2. It is built on top of two key components: a distributed, strongly consistent key-value store and the next-generation cellular communication technology. The distributed key-value store is composed of distributed UTM Data Service (UDS) units spread across the urban area as shown in Figure 1. We assume that UDSs are deployed in federated cloudlets interconnected through low-latency mobile network, such as 5G or 6G[23]. In this section, we describe the design of our architecture by focusing on the SMR protocol and the architecture's API endpoints. We skip further details about the mobile network due to space constraint.

A. State Machine Replication

To enable fault-tolerant, highly available key-value store, UDSs run state machine replication (SMR) protocol[24]. SMR is a technique to implement fault-tolerant service by replicating servers and coordinating client interactions with server

replicas. In particular, stakeholders (e.g., service providers, operators and vehicles) are clients and servers/replicas are UDSs. Our architecture leverages *etcd* [2], a distributed, strongly-consistent key-value store, to run SMR. This distributed approach solves important problems of ADS-B [25], such as lack of fault tolerance, data inconsistency, and unpredictable availability of shared data.

B. API Endpoints

The API endpoints are used to put or get data from the system.

- ADS-B Transponder - A ADS-B receiver connected to the system can put data from aircraft.
- Service Endpoint - The end-point that can be used by the UTM service providers or operators to get the current and past state of the airspace via restful API, time series or stream.
- UAS Endpoint - Used by UASs to insert broadcast data into the system.

IV. Experimental Evaluation

This section presents the performance evaluation of our system architecture using MACE [26], an emulation framework to study, design and evaluate mobile Ad-hoc applications. Figure 3 shows the architecture of our emulation with MACE detailing how the main blocks communicate with each other.

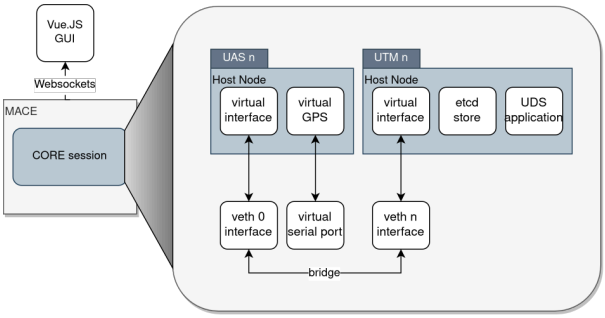


Fig. 3: Emulation Architecture

A. Cloudlets and Aircraft

MACE uses CORE as network emulator, and it emulates each network instance as Linux namespaces serving as minimal containers. Each aircraft client application runs inside these namespaces, and they communicate via *veth* network interfaces with connectivity controlled by the distance between the nodes. The cloudlets are also emulated in such namespaces with a server running the UAS endpoint and interfacing instances of etcd running in the same namespaces. The emulated scenario is run over an area of one square kilometre.

[2]https://etcd.io/

B. Mobility

The *Random Waypoint* mobility model was adopted for the experiment. In this model each aircraft receives a random waypoint and a random velocity to simulate a mission's objective. The movement of the aircraft is emulated in MACE, and the real-time position is injected directly in the network emulator so that it is reflected in the network connectivity. The position is made available to the applications running in the virtual aircraft via UNIX sockets.

C. UAS Broadcasts

For the payload reporting, a client running in each aircraft broadcasts a JSON object containing the position and additional data via IPV4 UDP sockets using the emulated Ad-hoc wireless links. The payload includes also an unique message ID, timestamp, an aircraft ID, velocity and status. The unique message ID is used to verify that the message was received and the timestamp used to calculate the latency between the broadcast and full replication.

D. UDS Cloudlet

To simulate the UDS cloudlets, a server representing them is deployed in each emulated cloudlet. The servers which are in range of a broadcasted message, will receive the new data in the UAS endpoint and replicate it to the other towers which are not in range by using the state machine replication layer. Data is also written in persistent storage for posterior analysis. The consistent replication ensures that the data is reliable across all participant cloudlets, so that distributed third part actors can see this reporting system as a trusted oracle for UTM applications such as asset managers, schedulers and collision avoidance path planners. When a message is fully replicated among all replicas, a callback function records the time stamp so that the total latency can be calculated.

All the experiments were performed on a Linux server whose configuration is described in Table I. Our empirical evaluation is split in the scenarios detailed in Table II.

TABLE I: Test Platform

Processor Manufacturer / Model:	Intel i7-8550U @ 4GHz
Number of cores	8
Memory	16GB DDR4
Emulator	MACE
Operating System	Linux Mint 20 x86 64bits
Routing Protocol	B.A.T.M.A.N IV

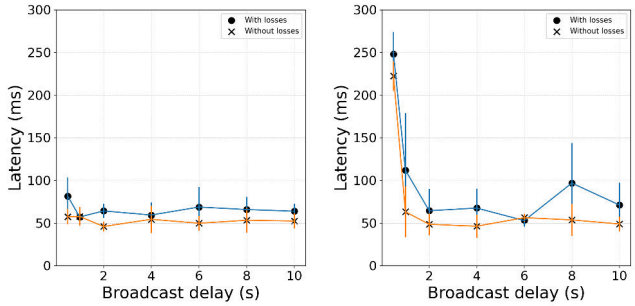
TABLE II: Emulation Scenarios

Setting	Without Losses	With Losses
Cloudlet Range	250m	250m
Number of cloudlets	4	10
Cloudlet Bitrate / Delay	433.3Mbps / 1000us	433.3Mbps / 1000us
UAS Range	90m	250m
Number of UASs	5 and 20	5 and 20
UAS Bitrate / Delay	54Mbps / 3000us	54Mbps / 3000us
Jitter	0 us	2 us
Error rate	0%	1%

E. Preliminary Results

In the first scenario, the main goal was to measure the latency between the data broadcasting and full replication when there are no network losses between cloudlets or in the UAS communication interfaces. One of the side effects of the strong consistency is the communication complexity consequence of the multistep consensus algorithm. So, in addition of the broadcast latency, it is expected also a component for the replication latency.

We assume that ADS-B broadcasts 2 messages per second since detect and avoid / path planning algorithms usually run with frequencies ranging from 1Hz to 2Hz. A previous experiment with only one fixed UAS was run to measure the replication overhead. Running such scenario yielded an replication overhead of 10ms, which is acceptable but is closely related to the number of replicas and, more considerably, to the total coverage area. Larger areas would require more communication hops even if the number of replicas is left unchanged, hence increasing the total latency.



(a) 10 Replicas and 5 UAS (b) 10 Replicas and 20 UAS

Fig. 4: Total replication latency with increasing broadcast delay

In the second scenario, losses are added to emulate a more realistic scenario, but the values of losses, latency and bandwidth are closely related to communication technology adopted. Due to the random nature of the mobility model, each emulation session yields different results, so the median and standard deviation of 5 sessions are shown. Figure 4(a) illustrate the results for 5 UAS, while Figure 4(b) the results for 20 UAS for both scenarios.

The impact of the introduction of losses is perceived differently in the UAS links and intra-cloudlets links. Since in this implementation the UAS broadcasts data via UDP links (more suitable for mobile and lossy links), lost messages do not impact the broadcast latency because there are no timeouts and retransmissions. Since *etcd* is implemented to communicate via TCP, the losses increase the overall replication latency as shown in the Figure 4. It is important to notice in Figure 4(b) that increasing the number of aircraft while leaving the delay between broadcasts low increase significantly the end-to-end latency. This could mean that in order for such platform scale to more representative scenarios of a crowded airspace, the number of replicas cannot be too high. It is also important to highlight that since these were performed with an emulator, all software components related to all cloudlets and UAS, such as the clients, servers and etcd, are running in the same computer and competing for resources, and that the its state can affect the results. It is therefore unfeasible to emulate scalability tests on an emulation platform, since they consume too much computer resources, and more tests with a simplified replication model running on a scalable simulator are required.

V. Discussion, Limitation and Future Work

The emulator helped us to implement and evaluate a very first prototype of our proposal, providing quick, yet preliminary results. For instance, latency measurements with many applications running on a single computer might skew the results. The server implementation (i.e., an UDS unit) used a callback function from a `etcd` library that was not designed to have low-latency capability, specially when running multiple requests simultaneously. In this preliminary study, we also overlooked the impact of the number of replicas in the overall latency while keeping a large number of access points. We also ignored the impact of different failure models, including a Byzantine one. One important aspect of such deployments is how they will act in presence of malicious actors, which is a reasonable assumption for a open air platform with competing actors. We believe that the target system could tolerate attacks such as Sybil attacks [27]. Alternatively, we could consider a distributed data store based on blockchain technique in order to cope with Byzantine faults.

For future work, scenarios with larger areas, different number of replicas are required. This would ensure a more clear view of the latency impact of the replication quorum. Furthermore, scalability studies are also required to simulate scenarios with hundreds of UAS. A fast simulator would also allow the study of optimal placement of such replicas in different realistic maps.

VI. Conclusions

This work presents a novel tracking and position reporting system to enable emerging UTM services for autonomous vehicles operating at very low level airspace. Our preliminary performance evaluation suggests that the latency measurements yielded results that are suitable for building a system architecture for UTM in densely populated areas. The highly available, distributed data store allows even vehicles that are hidden behind constructions to be spotted by all stakeholders in a consistent way. In contrast to the current ADS-B, our architecture enable fast, fault-tolerant data sharing by design.

References

[1] H. Shakhatreh, A. H. Sawalmeh, A. Al-Fuqaha, Z. Dou, E. Almaita, I. Khalil, N. S. Othman, A. Khreishah, and M. Guizani, "Unmanned aerial vehicles (uavs): A survey on civil applications and key research challenges," *Ieee Access*, vol. 7, pp. 48 572–48 634, 2019.

[2] S. J. Undertaking, "European drones outlook study," *SESAR Joint Undertaking: Brussels, Belgium*, 2016.

[3] R. Rumba and A. Nikitenko, "The wild west of drones: A review on autonomous- UAV traffic-management," *2020 International Conference on Unmanned Aircraft Systems, ICUAS 2020*, pp. 1317–1322, 2020.

[4] J. Lundberg, K. L. Palmerius, and B. Josefsson, "Urban Air Traffic Management (UTM) Implementation in cities - Sampled side-effects," *AIAA/IEEE Digital Avionics Systems Conference - Proceedings*, vol. 2018-Septe, no. September, 2018.

[5] M. Elsayed and M. Mohamed, "The impact of airspace regulations on unmanned aerial vehicles in last-mile operation," *Transportation Research Part D: Transport and Environment*, vol. 87, no. August, p. 102480, 2020. [Online]. Available: https://doi.org/10.1016/j.trd.2020.102480

[6] P. Kopardekar, J. Rios, T. Prevot, M. Johnson, J. Jung, and J. E. Robinson, "Unmanned aircraft system traffic management (UTM) concept of operations," *16th AIAA Aviation Technology, Integration, and Operations Conference*, pp. 1–16, 2016.

[7] J. Rios, D. Mulfinger, J. Homola, and P. Venkatesan, "NASA UAS traffic management national campaign: Operations across Six UAS Test Sites," *AIAA/IEEE Digital Avionics Systems Conference - Proceedings*, vol. 2016-December, pp. 1–6, 2016.

[8] J. Rios, A. Aweiss, J. Jung, J. Homola, M. Johnson, and R. Johnson, "Flight Demonstration of Unmanned Aircraft System (Uas) Traffic Management (utm) at Technical Capability Level 3," *Aiaa Aviation 2020 Forum*, vol. 1 PartF, 2020.

[9] A. Chakrabarty and C. Ippolito, "Autonomous flight for multi-copters flying in UTM -TCL4+ sharing common airspace," *AIAA Scitech 2020 Forum*, vol. 1 PartF, no. January, 2020.

[10] T. McCarthy, L. Pforte, and R. Burke, "Fundamental elements of an urban UTM," *Aerospace*, vol. 7, no. 7, 2020.

[11] N. S. Labib, G. Danoy, J. Musial, M. R. Brust, and P. Bouvry, "A multilayer low-altitude airspace model for UAV traffic management," *DIVANet 2019 - Proceedings of the 9th ACM Symposium on Design and Analysis of Intelligent Vehicular Networks and Applications*, pp. 57–63, 2019.

[12] D. Scott, M. Radmanesh, M. Sarim, A. Deshpande, M. Kumar, and R. Pragada, "Distributed bidding-based detect-and-avoid for multiple unmanned aerial vehicles in national airspace," in *2019 International Conference on Unmanned Aircraft Systems, ICUAS 2019*, 2019, pp. 930–936.

[13] F. Minucci, E. Vinogradov, and S. Pollin, "Avoiding Collisions at Any (Low) Cost: ADS-B like Position Broadcast for UAVs," *IEEE Access*, vol. 8, pp. 121 843–121 857, 2020.

[14] J. H. Park, S. C. Choi, J. Kim, and K. H. Won, "Unmanned Aerial System Traffic Management with WAVE Protocol for Collision Avoidance," *International Conference on Ubiquitous and Future Networks, ICUFN*, vol. 2018-July, pp. 8–10, 2018.

[15] Yucong Lin and S. Saripalli, "Sense and avoid for unmanned aerial vehicles using ads-b," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 6402–6407.

[16] J. Besada, I. Campaña, L. Bergesio, A. Bernardos, and G. de Miguel, "Drone flight planning for safe urban operations: UTM requirements and tools," *Personal and Ubiquitous Computing*, pp. 924–930, 2020.

[17] Q. Tan, Z. Wang, Y. S. Ong, and K. H. Low, "Evolutionary optimization-based mission planning for UAS traffic management (UTM)," *2019 International Conference on Unmanned Aircraft Systems, ICUAS 2019*, pp. 952–958, 2019.

[18] P. B. Sujit and R. Beard, "Multiple uav path planning using anytime algorithms," in *2009 American Control Conference*, 2009, pp. 2978–2983.

[19] A. Chakrabarty, V. Stepanyan, K. Krishnakumar, and C. Ippolito, "Real-time path planning for multi-copters flying in UTM-TCL4," *AIAA Scitech 2019 Forum*, no. January, 2019.

[20] G. L. Orrell, A. Chen, and C. J. Reynolds, "Small unmanned aircraft system (SUAS) automatic dependent surveillance-broadcast (ADS-B) like surveillance concept of operations: A path forward for small UAS surveillance," *AIAA/IEEE Digital Avionics Systems Conference - Proceedings*, vol. 2017-Septe, 2017.

[21] C. E. Lin, "An ADS-B like communication for UTM," *Integrated Communications, Navigation and Surveillance Conference, ICNS*, vol. 2019-April, pp. 1–12, 2019.

[22] Y. H. Lin, C. E. Lin, and H. C. Chen, "ADS-B Like UTM surveillance using APRS infrastructure," *Aerospace*, vol. 7, no. 7, pp. 1–14, 2020.

[23] M. M. Azari, S. Solanki, S. Chatzinotas, O. Kodheli, H. Sallouha, A. Colpaert, J. Fabian, M. Montoya, S. Pollin, A. Haqiqatnejad, A. Mostaani, E. Lagunas, and B. Ottersten, "Evolution of Non-Terrestrial Networks From 5G to 6G : A Survey," pp. 1–35.

[24] F. B. Schneider, "Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial," *ACM Computing Surveys (CSUR)*, vol. 22, no. 4, pp. 299–319, 1990.

[25] Y. Kim, J. Y. Jo, and S. Lee, "ADS-B vulnerabilities and a security solution with a timestamp," *IEEE Aerospace and Electronic Systems Magazine*, vol. 32, no. 11, pp. 52–61, 2017.

[26] B. C. Ferreira, G. Dufour, and G. Silvestre, "Mace: A mobile ad-hoc computing emulation framework," in *2021 International Conference on Computer Communications and Networks (ICCCN)*, 2021, pp. 1–6.

[27] B. Yu, C. Z. Xu, and B. Xiao, "Detecting Sybil attacks in VANETs," *Journal of Parallel and Distributed Computing*, vol. 73, no. 6, pp. 746–756, 2013.

A cross-domain framework for Operational Design Domain specification

Guillaume Ollier[1], Morayo Adedjouma[1], Simos Gerasimou[2], Chokri Mraidha[1]
[1]*Université Paris-Saclay, CEA LIST, Dept. Ingenierie Logiciels et Systemes, P.C. 174, Gif-sur-Yvette, 91191 Cedex, France*
[2]*University of York, Department of Computer Science, United Kingdom*
guillaume.ollier@cea.fr

***Abstract*—This paper presents a method to generalize the concept of "Operational Design Domain" (ODD) used in the automotive domain to any cyber-physical system. The approach proposes to use domain-level and meta-theories taxonomies to develop a cross domain ontology for the definition of the ODD.**

***Index Terms*—ODD, ontology, taxonomy, logical scenario, Autonomous System**

I. Introduction

To solve the challenge of the specification of the intended capabilities and limitations of Autonomous Systems (ASs) based on AI models, a solution is to capture the scenario space covering all possible Usage Scenarios (USs) of the system. Such scenario space is defined in the automotive domain with the concept of Operational Design Domain (ODD) [1]. Within ODDs, USs are decomposed into Operating Conditions (OCs) which might include environmental conditions (e.g, illuminance, weather, traffic), conditions on the ego system (e.g, speed limitations, maneuvers), etc. The OC terms and their relations can be formalized through an ontology, i.e, *"a representation, formal naming, and definition of the categories, properties, and relation between the concepts, data and entities that substantiate one, many, or all domains of discourse"* [2]. In other words, an ontology can represent the body of knowledge in a given field. It seems, essential to be able to generalize this concept of ODD from automobile to any other domain relative to ASs.

Indeed, the approach could provide a huge benefit for all safety-oriented and scenario-based methods for systems development at various phases, e.g., for the specification and design phase, the ODD could support the identification of the capacities and limitations of the system and refine it as the safety analysis advances. The ODD could also be a source for the verification and validation scenario specification and help to guide the testing process.

However, while current approaches supporting the ODD specification are only adapted to one specific domain, it may be interesting to define a commonly controlled vocabulary that may embody the knowledge related to ASs from different application domains in a harmonized way. In this paper, we present a cross-domain approach for ODD specification. Our goal is to define a method to formalize the overall scenario space relevant for any application domains of ASs. In Figure 1, we illustrate how the different scenario spaces are related. The outer oval represents the overall scenario space for any AS. We refer here to the world in which any of these ASs might operate, no matter their capability. We represent this scenario space with a global ontology. The second most external oval represents the domain-specific scenario space. This scenario space is a subset of our AS scenario space and it is represented with relevant aspects for a domain extracted from our global ontology. The third oval represents our Operational Domain (OD) as an ADS-specific scenario space and it is formed of all OCs of this ADS. Then, the oval in it is for the ODD of the system. This scenario space represents the intended AS capability to handle OCs. We exclude from this space the specific OC combinations unsafe for our system. This space is refined through the development process. Lastly, the inner oval represents a US, i.e, any scenario for which our system

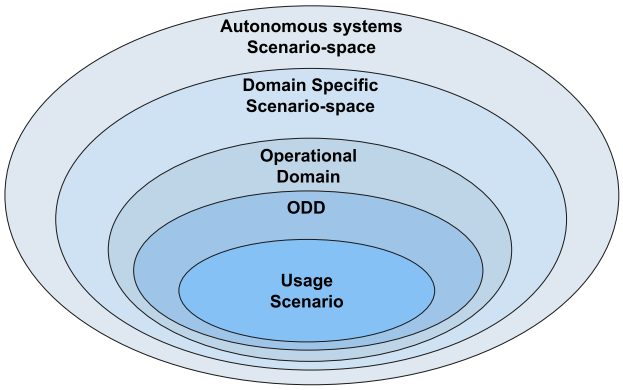


Fig. 1. A representation of the different scenario spaces.

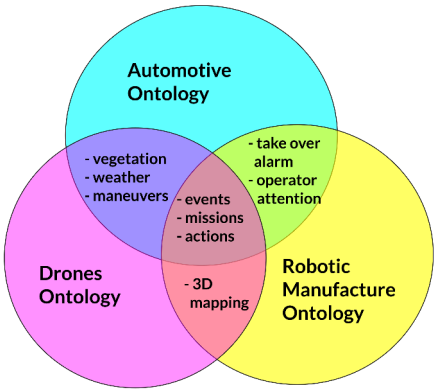


Fig. 2. A representation of concepts in common for several domains.

is specifically designed to function.

We ensure that our global ontology contains all the concepts from the most common safety-critical AS domains, e.g, automotive, avionics, and manufacturing. We compile and structure these concepts into ”upper ontologies”, i.e, ontologies reusable to specify the ODD from new domains systems.

Our multi-domain formalization aims to facilitate the definition of the scenario space for a new application domain using the captured knowledge from existing ones. We aim to validate the approach on ASs from various UCs concerning different domains, including the domains used to extract the generic concepts but also new domains which combine concepts from other domains, e.g, urban rescue robot which combine urban environment from automotive domain and object manipulation [3]. The notion of “generic concepts” is illustrated in Figure 2. Each circle represents a specific-domain ontology. At the crossing of two circles, we list as examples some common concepts for these two corresponding domains, e.g, the drone and automotive systems are designed to work outside then the concepts relative to weather and vegetation are relevant for these two domains. In the space shared by all circles, we list some concepts relevant for any AS domains, e.g, events or actions.

II. Background

To address the problem of the harmonization of scenario representation through different domains, we use upper ontologies [4], i.e, general concepts which can be reused to express knowledge for several different domains, e.g, space, time, weather, infrastructure. The knowledge transfer from a well-defined domain to new domains is resolved by these upper ontologies that ensure completeness of semantic representation. These upper ontologies can be cross-cutting to all domains, e.g, ATIC for time representation [5], RCC-8 for space representation [6], or they can concern only a domain set, e.g. CORA [7] for the autonomous robotic domain. We also integrate domain standards (e.g, the PAS1883 [8] which defines a taxonomy for safe automated driving systems, the taxonomy of unmanned aircraft, and their operations [9]).

III. Approach

Our approach follows several steps as presented below. The activity and its output concerning the compilation of the concepts used to describe OCs (S1 and O1) are independent of the activity and its output concerning the formalization language used to structure these concepts (S2 and O2). Figure 3 presents how these steps are organized. The output “Taxonomies” (O1) is illustrated by a portion of the figure from the PAS1883 standard whereas the output “ontology formalization” is illustrated by a modified version of the UFO meta-model [10]. This model is UML-based and we present here the different stereotypes used with the class stereotypes in rectangle shapes and association stereotypes in stadium shapes. The stereotypes in green, yellow, and red shapes model respectively the OCs, events and, intentional entities. This representation of events and intentional entities is not part of the ODD itself but it complements OCs to allow US description for safety analysis.

S1 Theories & Standards: We list all standards and meta theories which present concepts to describe usage scenario for ASs from various domains. We use all the acquired standards to build a taxonomy that lists all OCs extracted from theories and standards. The knowledge from expertise related to ASs has to be represented, e.g, computer vision, human operator factors, system engineering.

S2. Ontology Language Definition: We define a domain-specific language to capture our knowledge representation as ontologies. We specify additional modeling constraint and change the vocabulary to adapt our ontology representation to US description.

S3. Meta-domains Knowledge: We formalize our taxonomies (O1) with our ontology language (O2). The obtained ontology has to cover all relevant concepts to analyze ASs from any domain.

S4. Generics domains Knowledge: Any new AS domains can be represented using upper ontologies, e.g, a drone on-

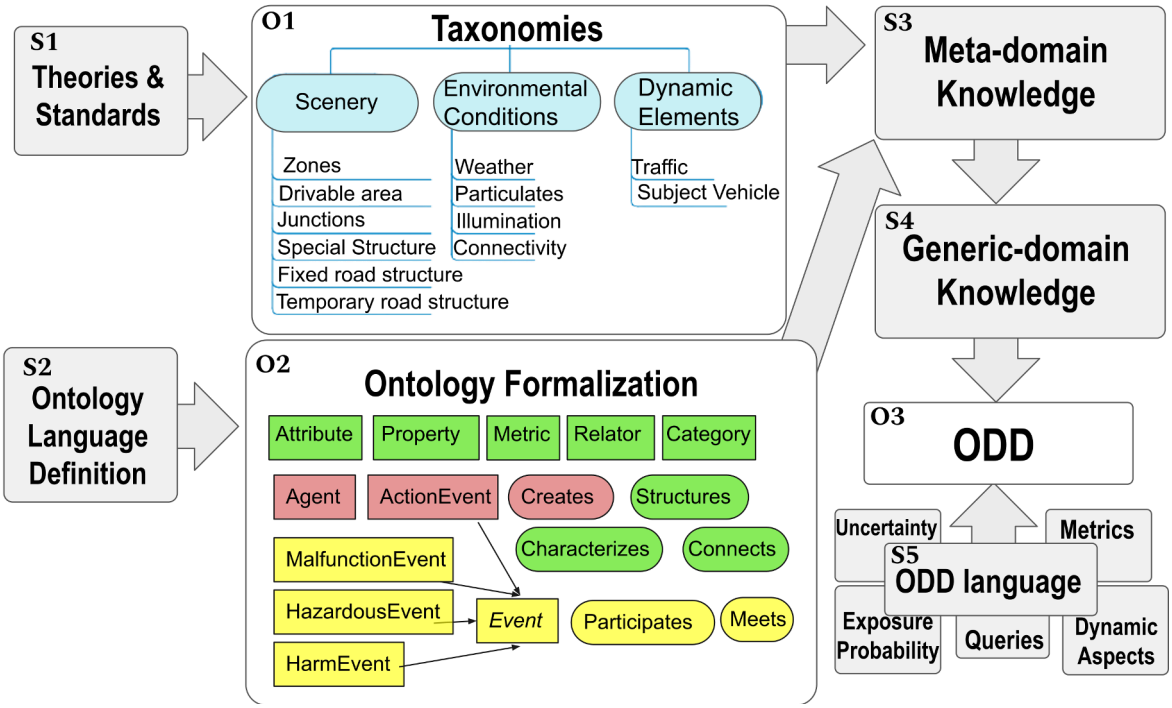


Fig. 3. Our approach steps.

tology can be built using a weather ontology and an aerial environment ontology. The upper ontology representation is completed with domain-specific concepts, e.g, the drone maneuvers. The classic domains (e.g, automotive, aviation, robotic manufacture) are defined in a similar way to make them compatible with the ODD formalization.

S5. ODD Specification: For the ODD definition of a specific UC, we extract from the corresponding domain ontology (defined at S4) all the required concepts to characterize the scenario space of the UC. Furthermore, the ODD specification includes the OCs together with their properties and applicable range or limit values, e.g, the OC "moderate rain" may be included in the ODD with 2.5 mm/h as minimum values and 7.6 mm/h as maximum values. It is also completed by information to help the safety analysis, e.g. the exposure probability and acceptable uncertainty thresholds.

IV. Related Work

We propose three main contributions in our paper: a cross-domain AS taxonomy definition, the ontology language formalization adapted to US description, and an ODD language formalization. The following subsections present related work for each of these contributions.

A. Autonomous Systems Taxonomy

The taxonomy definition for an autonomous system has been presented in [11]. A meta-domain knowledge is extracted from meta-theories for known cross-cutting domains i.e, time, space, communication, etc. and it might be used to represent plain domains. The article doesn't provide an exhaustive list of meta-theories. A framework to categorize the OCs of a UC is given in [12]. This classification was defined for the automotive domain but it is enough general to be adapted to other domains. Another framework to define safety-relevant scenarios at a logical level is presented in [13]. It is based on a 6-layer model to represent the road environment including the structural and dynamic aspects. This notion of layers for scenario representation could be adapted and generalized on other domains.

B. Ontology Languages

For the ontologies formalization, the most common approach is to use the Web Ontology Language (OWL) [14]. It is capable of representing classes, properties, defining instances and its operations and it structures the semantics for reasoning and inferences. But although this language family is employed in practice for conceptual modeling, it is not designed to be specifically truthful to reality, i.e, there are no modeling

constraints to guarantee that our ontology can be used to analyze the scenario space. For example, for these modeling constraints, the property classes shall be connected to exactly one bearer class, or a rigid attribute (i.e, attribute that must instantiate a given type in all possible scenarios in which it exists) shall not subclass a state. The approach proposed by the Unified Foundational Ontology (UFO) [10] aims to solve this conceptual modeling problem. It is an extension of Unified Modeling Language (UML) and it is implemented in OntoUML [15]. A formal ontological framework developed for hazard identification of autonomous systems and adapted from UFO is raised in [16]. This paper proposes a method to incorporate ideas from other models to build their "ESHA ontology" but the complete ontology language is not presented.

C. ODD Languages

Some initiatives to define a Domain Specific Language (DSL) to represent ODD for vehicles has been developed with the projects OpenODD [17] and [18]. These formats are designed to be machine-processable throughout the vehicle development and lifecycle thanks to a well-defined syntax and semantics. It also includes pre-defined attributes, metrics, scenario exposure probabilities. The standard OpenODD also includes representation of scenarios uncertainty, and a query language to access and manipulate a scenario database.

V. Conclusion & Future Work

We have presented an approach to formalize and build the ODD of autonomous systems for any domain. We detailed all the needs to achieve this formalization from the domain representation to the system-specific constraints representation. We further need to implement the tool for domain and ODD specification. For the choice of the ontology language, we have to compare the existing ones to select the most appropriate given our requirements and extend it if needed. Our tool has to be compatible with the existing DSLs (e.g, OpenODD). To make our approach usable even for non-experts, a user interface could guide stakeholders through the domain description and ODD boundaries specification. The OC selection could be achieved with predefined questions on the system. Finally, our validation process includes evaluating our approach on UCs from various domains.

VI. Acknowledgements

This work was partially supported by the European H2020-ECSEL CPS4EU project under grant no 692474 and the Confiance.ai project of the Grand Défi "Securing, certifying and enhancing the reliability of systems based on artificial intelligence" launched by the French Innovation Council.

References

[1] SAE Mobilus. SAE J3016 Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles. Technical report, Society of Automotive Engineers International, 2018.

[2] George H Mealy. Another look at data. In *Proceedings of the November 14-16, 1967, fall joint computer conference*, pages 525–534, 1967.

[3] K. Osuka, Robin Murphy, and Alan Schultz. Usar competitions for physically situated robots. *Robotics & Automation Magazine, IEEE*, 9:26 – 33, 10 2002.

[4] Viviana Mascardi, Valentina Cordì, and Paolo Rosso. A comparison of upper ontologies. In *Woa*, volume 2007, pages 55–64. Citeseer, 2007.

[5] James F Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.

[6] Anthony G Cohn, Brandon Bennett, John Gooday, and Nicholas Mark Gotts. Qualitative spatial representation and reasoning with the region connection calculus. *GeoInformatica*, 1(3):275–316, 1997.

[7] Joanna Isabelle Olszewska, Marcos Barreto, Julita Bermejo-Alonso, Joel Carbonera, Abdelghani Chibani, Sandro Fiorini, Paulo Goncalves, Maki Habib, Alaa Khamis, Alberto Olivares, et al. Ontology for autonomous robotics. In *2017 26th IEEE international symposium on robot and human interactive communication (RO-MAN)*, pages 189–194. IEEE, 2017.

[8] British Standard Institution. PAS 1883 Operational Design Domain (ODD) taxonomy for an automated driving system – Specification. Standard, British Standard Institution, 2020.

[9] Anna Masutti and Filippo Tomasello. *International regulation of non-military drones*. Edward Elgar Publishing, 2018.

[10] Giancarlo Guizzardi. Ontological foundations for structural conceptual models. 2005.

[11] Leonard Petnga, Mark Austin, and Mark Blackburn. Semantically-enabled model-based systems: Engineering of safety-critical network of systems. *Insight*, 20(3):29–38, 2017.

[12] Magnus Gyllenhammar et al. Towards an operational design domain that supports the safety argumentation of an automated driving system. In *10th European Congress on Embedded Real Time Systems (ERTS 2020)*, 2020.

[13] Hendrik Weber, Julian Bock, Jens Klimke, Christian Roesener, Johannes Hiller, Robert Krajewski, Adrian Zlocki, and Lutz Eckstein. A framework for definition of logical scenarios for safety assurance of automated driving. *Traffic injury prevention*, 20(sup1):S65–S70, 2019.

[14] Holger Knublauch, Daniel Oberle, Phil Tetlow, Evan Wallace, JZ Pan, and M Uschold. A semantic web primer for object-oriented software developers. *W3c working group note, W3C*, 2006.

[15] OntoUML metamodel & definition. https://ontouml.org/ontouml/metamodel-definitions/. Accessed: 2021-10-02.

[16] Christopher Harper and Praminda Caleb-Solly. Towards an ontological framework for environmental survey hazard analysis of autonomous systems. In *SafeAI@ AAAI*, 2021.

[17] ASAM OpenODD project details. https://www.asam.net/index.php?eID=dumpFile&t=f&f=4544&token=1260ce1c4f0afdbe18261f7137c689b1d9c27576. Accessed: 2021-09-27.

[18] Daniel Hillen and Jan Reich. *Model-based Identification of Operational Design Domains for Dynamic Risk Assessment of Autonomous Vehicles*. PhD thesis, 08 2020.

Towards Real-time Adaptive Approximation

Raheleh Biglari
Cosys-lab, University Of Antwerp
Flanders Make@Uantwerpen
Raheleh.Biglari@uantwerpen.be

Joost Mertens
Cosys-lab, University Of Antwerp
Flanders Make@Uantwerpen
Joost.Mertens@uantwerpen.be

Joachim Denil
Cosys-lab, University Of Antwerp
Flanders Make@Uantwerpen
Joachim.Denil@uantwerpen.be

***Abstract*—Cyber-physical systems (CPS) are real-time systems that operate in dynamic and non-deterministic environments. Models are often used for control and prediction, however do not reason on the trade-off between real-time constraints and uncertainty. This paper presents a conceptual model to reason on adaptive approximation in such systems. Furthermore, we envision a framework to allow the adaptivity of models, balancing between uncertainty and the real-time behavior of the system.**

***Index Terms*—cyber physical systems, real-time, uncertainty, adaptation, abstraction**

I. Introduction

Cyber-physical systems (CPS) are engineered systems that have tight integration between the cyber part (computation and networking) and its physical components [1]. Examples include but are not limited to industry 4.0, automotive, and aerospace. Engineered systems have a goal to achieve in the context of the system, e.g. an autonomous vehicle needs to pilot the environment while not harming anyone. To achieve this goal, multiple decision models are needed and combined. For example, an autonomous vehicle has low-level control to accelerate and brake, tactical decision-making models for path planning, and strategic models to decide which roads to avoid, e.g., as an accident occurred. All of these decision models implement some form of (feedback) control.

Cyber-physical systems of systems are CPS that exhibit the features of a system of systems (SoS). They are large and spatially distributed, have distributed control, and autonomic behavior where parts of the SoS can join or leave the system [2]. As such it is a system composed from different CPS where each part of the system contributes to the overall goal of the CPSoS. The engineering of such a CPSoS has to address the complex situations and, environments of the system, which is characterized by ambiguity, high uncertainty and emergence [3]. CPSoS have to allow for collaborative decision making and, as such, need to be aware of the state of the other constituents of the system [2].

CPS and CPSoS operate in a very dynamic environment where lots of uncertainty is present [4]. Uncertainty points to the lack of information that is available about the system or its environment. An autonomous vehicle might have uncertainties about its own position in the system and about the direction and velocities of other vehicles and road users. Cyber-physical systems are also real-time systems which means that the time at which a decision is made is as important as the decision itself. This means that during design time, the system is analyzed to ensure that all the deadlines of the control components are met in the worst-case.

Raheleh Biglari is funded by the BOF fund at the University of Antwerp. Joost Mertens is funded by the Research Foundation - Flanders (FWO) through strategic basic research grant 1SD3421N.

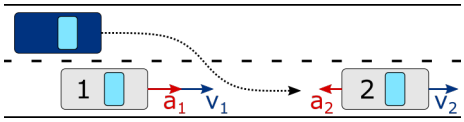


Fig. 1. Lane changing scenario.

One way to deal with the contradiction of better performance and reduced cost in CPS is to allow adaptivity at run-time and to change the underlying decision and estimation models such that they are sufficient for computing the control actions but at the same time computationally less intensive. Techniques that abstract or approximate models are commonly available in the literature, e.g., surrogate modeling [5].

In this short paper, we look at different dimensions of the problem to allow for such a run-time adaptation of the underlying control and decision models with more abstract and approximate models. The rest of the paper is organized as follows: section II introduces our running example, section III describes the challenges in our work. We also present use case evaluation results. In section IV we describe our strategy to dealing with the challenges. We provide related works in section V and section VI presents our conclusion and discusses future directions.

II. Running Example

We use a lane change control algorithm to show the challenges of introducing adaptive abstraction and approximation in a real-time context. While we do not aim precisely at this class of control algorithms, lane changing allows for visual and intuitive reasoning over the problem space.

Lane changing algorithms control the lateral direction of the vehicle. The scenario throughout this paper is shown in Figure 1. The ego car (in dark blue) tries to change the lane in between two other cars. The velocity and acceleration vectors of the two cars is shown as vectors v_1,v_2, a_1 and a_2. The front car is lightly decelerating while the back car is lightly accelerating. Multiple algorithms have been proposed in the literature to solve such lane changing problems, e.g. [6]. In the context of this paper, we use a Simulink provided model

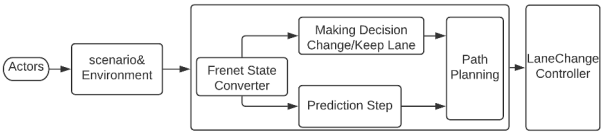


Fig. 2. Causal Block Diagram of the Lane Change Algorithm

to simulate the lane changing behavior. Figure 2 shows the high-level architecture of the lane changing algorithm. The algorithm uses Frenet Coordinate System, which represents the position of the car on the road more intuitively than traditional (x, y) coordinates.This algorithm also uses a prediction step to predict the trajectories of the different actors. Afterward, path planning takes care of finding a good path in the world. Finally, a model-predictive controller steers the vehicle over the planned path.

In the rest of the paper, we focus on the prediction step of the other vehicles. The prediction step simulates a trajectory relative to the ego car for each vehicle in the environment. The component receives information on the lateral and longitudinal position, velocity, and acceleration of the vehicles. In the default case, the prediction step uses a constant velocity model to predict the vehicle's position at multiple time-steps in a three-second window:$s(t) = v * t + s_0$ (with s the relative distance to the ego-car, v the relative speed to the ego vehicle, and s_0 the initial distance). However, we can imagine much more detailed models that also take the vehicle's acceleration into account or even more detailed models that simulate the complex decision-making in each of the vehicles.

III. Challenges

Using a model of a car performing a lane change such as the scenario in Section II, we elaborate on identified challenges. The model is simulated with Simulink.

A. Model Prediction Uncertainty

Each of the different models has a different amount of predictive power. As such, more detailed models typically have lesser prediction uncertainty. Figure 3 depicts the prediction of the scenario with two different models over three time-steps. The top control bar is the uncertain position of the cars using a constant velocity model. The lower control bar shows the uncertain position using a constant acceleration model.

To reason over using different alternative models, we need to map the prediction uncertainty of all the different models. Two types of uncertainty are typically present in modeling and simulation: aleatoric and epistemic uncertainty. Aleatory uncertainty is known as stochastic uncertainty and is due to probabilistic variability. Epistemic uncertainty is the uncertainty that occurs because of the lack of knowledge [7].

With the simulation model, we demonstrate how the constant velocity and constant acceleration models have different predictive power. Figure 4 shows the velocity profile of car 2 and the L2-norm (Euclidean distance) between 4 predictions and the true location of that car. The number behind each

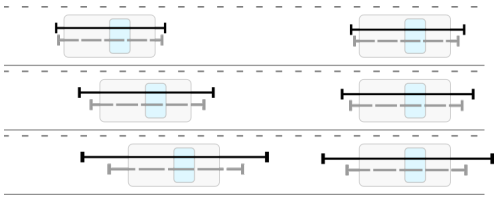


Fig. 3. Prediction of the car position on three time steps within the time window, relative to the leading car. On each car, the prediction uncertainty of two different models is shown, one solid, one dashed.

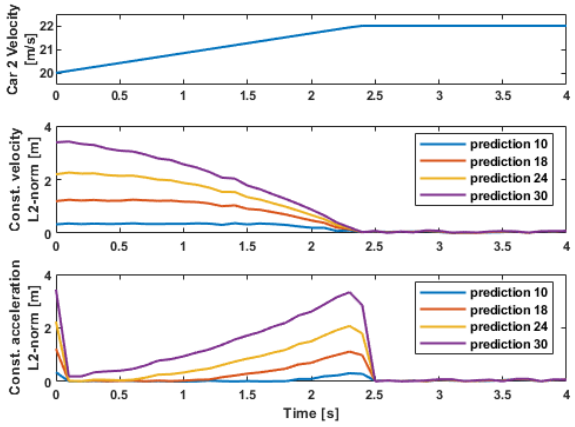


Fig. 4. Car 2's velocity profile, compared with the L2 errors made by the constant velocity and acceleration prediction models.

prediction tells us how many steps (of $0.1s$) in the future this prediction is. The simulation includes uncertainty on the sensor inputs of the ego vehicle, which results in slightly jagged error traces. The velocity profile shows that the car accelerates to 22 m/s. Afterward, it holds a constant velocity. We observe that under acceleration, the constant velocity model has a continuous error that only diminishes as the vehicle approaches constant velocity. The constant acceleration model performs better when conditions are constant, that is, positive, negative, or 0 acceleration, and shows the largest error when the predictions cross transitions in acceleration. Such transitions can be seen at $t = 0s$, where the initial acceleration measurement is 0, yet the car is already accelerating at $0.8m/s^2$, and at $t = 2.4s$, when the car stops accelerating. Under constant velocity, both models perform equally. From the results, we can say that the constant acceleration model has better predictive power, given the generally smaller errors.

B. Dynamic Environment

The environment in which systems operate is dynamic. Cars enter and leave the operating environment of the system. Some highways have more lanes than others. Even more, not all of the actors within the environment are of the same type. In a traffic environment, we have pedestrians, bicycles, cars, trucks, and buses that all behave differently. The prediction component of the lane change algorithm has to set up this dynamic

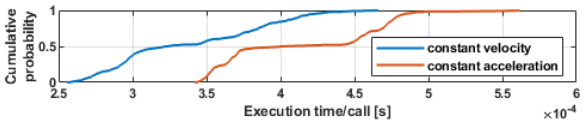


Fig. 5. Experimental CDF of the profiled execution times of both models.

environment each time and select the most appropriate models to include.

C. Real-time Constraints

Often our cyber-physical systems are also real-time systems. This means that the time at which the computation result is available is as important as that result itself. During the design of such systems, care is taken that the computations are finished before the expiry of the deadline of the computation. Different analytical techniques are available to check that this is correct. However, in the dynamic environment described above, this is difficult. How many cars, pedestrians, and bicycles are in the system's environment is unknown at design time. However, we still need a prediction of the behavior of each of these actors in the example within a certain time-span.

With the simulation model, we can demonstrate the impact computations can have on the number of predictions. Figure 5 shows the experimental cumulative distribution function of the profiled execution times of the state predictor for the constant velocity and acceleration models. 400 simulations were run for each model. Although profiling a Simulink model does not yield real-time results, it does allow us to relatively compare the computational burden of its blocks, We observe that, on average, the constant acceleration model requires 22.74% more computation time than the constant velocity model. This implies that for predicting 4 other cars with constant acceleration, 5 cars with constant velocity could be computed.

IV. Approach

This section details our approach to handle these challenges. In the first part of this section, we look at a conceptual manner on reasoning over substitutability of models. We look at the effect of approximating a model with a surrogate model and how it impacts a decision making algorithm. The conceptual framework is used as the foundation for a framework that allows for the run-time adaptation of a system where models can be swapped by more approximate models based on the context of the system and the available library of models.

A. Language Engineering Framework

To reason on the challenges, we propose an adapted version of the conceptual framework proposed by Barroca, Kuhne and Vangheluwe [8]. The conceptual framework in Figure 6 integrates ontological and language engineering. We extended the framework with two different models with different approximations, and the layer of the decision-making algorithms. The linguistic level starts with a simulation model. The simulation model has semantics (denoted by [[.]]), which results in a

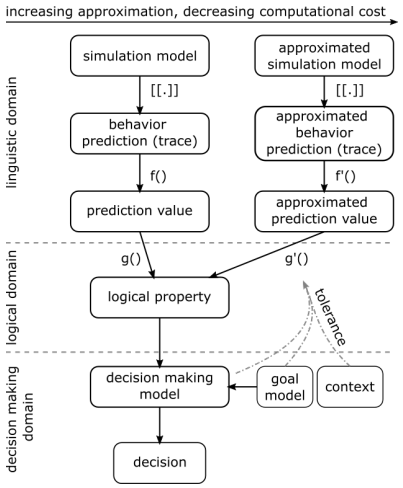


Fig. 6. Conceptual framework.

simulation behavior trace. However, the trace might not be the quantity of interest of the decision-making algorithm or for reasoning over the logical behavior of the system. A function $f()$ (e.g. integrating the signal) transforms the trace into some quantity of interest, here called the prediction value. A logical property is on the ontological level where it gets a real-world meaning, e.g. will the two cars collide? The logical property is a Boolean value; either the cars collide, or they do not collide. Another function $g()$ is used to transform between the quantity of interest and the logical property. Decision algorithms typically work directly with the prediction value but implicitly encode the transform within its algorithm. We, therefore, show the direct link between the logical property and the decision-making model.

The framework is analog for a model that is approximated. It reasons on the same logical property, however, using a more approximate model introduces uncertainty. This means that the function $g'()$ should give the same answer as the original model, except with more uncertainty. If the uncertainty is within bounds, the original model can be substituted with the more approximate one. This bound is defined by some metric, which is the tolerance to this change in the logical property (and hence later in the decision that was made by the decision-making model). The tolerance depends on a number of factors. In our lane changing example, the tolerance is based on (a) the goal model: do I actually want to change lanes, or do I want to stay in the same lane? When staying in the same lane, the cars on other lanes are maybe less important. (b) the context or environment of the system: is the car close-by (less tolerance) or far away (more tolerance) (c) the decision-making model itself: is the algorithm itself more or less tolerant to uncertainties?

B. Adaptive abstraction and approximation for real-time systems

Based on the insights provided by the conceptual framework, we propose an architecture for adaptive abstraction and

approximation for real-time systems. The architecture is shown in Figure 7. The high-level architecture is based on the MAPE-K architecture [9]. MAPE-K is a high-level control loop for self-adaptive systems. The *managed system* is, in our case, the embedded system running the control application for lane changing.

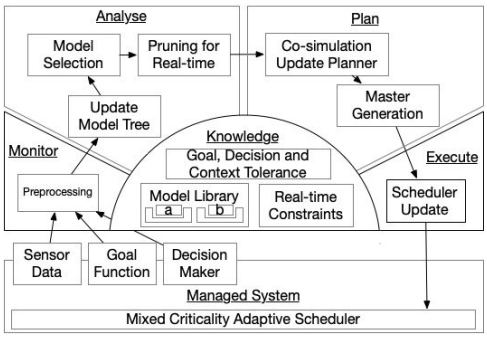


Fig. 7. Real-time Adaptive Abstraction and Approximation Architecture

The MAPE-K loop starts with a **M**onitor-phase where the necessary (processed) sensor data, the current goal function and the decision-maker are communicated to the adaptation mechanism. Here some processing occurs to allow for easier **A**nalysis. The analysis phase starts by updating a model tree. This model tree contains, in the ordered branches, all real-world objects that are passed by the sensor data. In the running example, we only have two objects: the two cars. It uses the (processed) sensor data to add or remove objects from the model tree incrementally. The Model Selection activity updates or selects the different models for each object in the model tree. The update is based on the goal, context and decision model, the quantified tolerances, and the available models with uncertainty quantification. The combination of these **K**nowledge items result in a set of rules to decide if a certain model can or cannot be used in this specific instance. Furthermore, the tree is also ordered with the most important objects in the first branches, and the most appropriate model in the first branch. In our case, the acceleration model is placed before the constant velocity model. If another car was present at a larger distance, the constant vehicle model would be first, and an empty model second branch. Using the accelerated model would not make sense as the tolerance for the decision-maker is high. The empty model is present, as the non-computing of the model reduces the execution time significantly but reduces the performance of the system. Finally, real-time constraints are imposed on top of the model tree to prune infeasible solutions. After the analysis phase, a set of feasible solutions is left. Based on this set of solutions, the co-simulation scheme is adapted and a master is generated that enables execution of the simulation. Note that each co-simulation unit contains the different possible models (possibly an empty model) for fast adaptation. Finally, in **E**xecute, the scheduler in the Managed system is updated.

The scheduler itself is an adaptive mixed-criticality scheduler based on [10]. This allows for responsiveness to overload conditions and imperfections in the execution time measurement of each co-simulation unit.

C. Discussion

This paper shows how to deal with the trade-off between uncertainty and real-time behaviour using models at different approximation levels. Consider if the number of cars increases in a scenario, the ECU will not have sufficient time to compute a prediction of all the different models.

The proposed conceptual model solves the dynamic environment where you cannot reason over real-time behaviour as worst-case bounds or where fallbacks in the decision logic are necessary. This solution needs computation of the MAPE-K loop and switching of models.

In this research, we address Tolerance Quantification. We need to be able to evaluate the tolerance of the decision making algorithm, since it is one of the model selection criteria. Off-line experiments are needed to see how a decision making algorithm responds to uncertainty. In the example, it is the tolerance to the uncertain distance and velocity vector that must be quantified.

V. Related Work

In previous work [11]–[13], we worked on the adaptivity of large scale simulations with surrogates. However, none of these techniques use a quantification of tolerance or reason over the real-time behaviour of the system.

The two most related techniques to our defined methods are mixed criticality systems and the imprecise computation model. Most of the complex embedded systems like automotive and avionic industries are mixed criticality systems. These systems deal with task-priority scheduling regarding execution time [14]. The imprecise computation is also a technique to deal with transient overload and improve real-time fault tolerance. This approach ensures that all critical tasks never miss their deadlines [15]. Both techniques do not take the uncertainty of the models into account but allow for adaptation based on real-time citeria.

VI. Conclusion and Future Work

In this paper, we presented a conceptual model for reasoning on adaptive approximation in a dynamic environment. We used a Simulink model for the simulation of the lane change control algorithm as a visual and tangible use case. Guided by it, we identify 3 main challenges faced for real-time adaptive approximation: model prediction uncertainty, dynamicness of environments and real-time constraints. To handle those challenges, we envisioned an integrated framework for adaptive approximation in CPS that balances the uncertainty and real-time behavior of the system. In the future, we aim to implement the framework with a supporting architecture, methods, and techniques to reason over the use of self-adaptive approximations and abstractions at runtime. We also want to create an appropriate modeling language and supporting techniques to find better runtime deployment solutions.

REFERENCES

[1] E. A. Lee, "Cyber physical systems: Design challenges," in *2008 11th IEEE international symposium on object and component-oriented real-time distributed computing (ISORC)*. IEEE, 2008, pp. 363–369.

[2] S. Engell, "Cyber physical sos-definition and core research and development areas," Working paper of the Support Action CPSoS. Retrieved from http://www. cpsos …, Tech. Rep., 2014.

[3] A. Sousa-Poza, S. Kovacic, and C. Keating, "System of systems engineering: an emerging multidiscipline," *International Journal of System of Systems Engineering*, vol. 1, no. 1-2, pp. 1–17, 2008.

[4] M. Zhang, B. Selic, S. Ali, T. Yue, O. Okariz, and R. Norgren, "Understanding uncertainty in cyber-physical systems: a conceptual model," in *European conference on modelling foundations and applications*. Springer, 2016, pp. 247–264.

[5] D. Caughlin and A. F. Sisti, "Summary of model abstraction techniques," in *Enabling Technology for Simulation Science*, vol. 3083. International Society for Optics and Photonics, 1997, pp. 2–13.

[6] D. Bevly, X. Cao, M. Gordon, G. Ozbilgin, D. Kari, B. Nelson, J. Woodruff, M. Barth, C. Murray, A. Kurt, K. Redmill, and U. Ozguner, "Lane change and merge maneuvers for connected and automated vehicles: A survey," *IEEE Transactions on Intelligent Vehicles*, vol. 1, no. 1, pp. 105–120, 2016.

[7] W. L. Oberkampf and C. J. Roy, *Verification and validation in scientific computing*. Cambridge University Press, 2010.

[8] B. Barroca, T. Kühne, and H. Vangheluwe, "Integrating language and ontology engineering." in *MPM@ MoDELS*. Citeseer, 2014, pp. 77–86.

[9] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[10] F. Guan, L. Peng, L. Perneel, H. Fayyad-Kazan, and M. Timmerman, "A design that incorporates adaptive reservation into mixed-criticality systems," *Scientific Programming*, vol. 2017, 2017.

[11] S. Bosmans, S. Mercelis, P. Hellinckx, and J. Denil, "Towards evaluating emergent behavior of the internet of things using large scale simulation techniques (wip)," in *Proceedings of the 4th ACM International Conference of Computing for Engineering and Sciences*, ser. ICCES'18. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3213187.3213191

[12] ——, "Reducing computational cost of large-scale simulations using opportunistic model approximation," in *2019 Spring Simulation Conference (SpringSim)*, 2019, pp. 1–12.

[13] S. Bosmans, T. Bogaerts, W. Casteels, S. Mercelis, J. Denil, and P. Hellinckx, "Adaptivity in multi-level traffic simulation using experimental frames," *Simulation Modelling Practice and Theory*, vol. 114, p. 102395, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1569190X2100099X

[14] A. Burns and R. Davis, "Mixed criticality systems-a review," *Department of Computer Science, University of York, Tech. Rep*, pp. 1–69, 2013.

[15] J. W. Liu, W.-K. Shih, K.-J. Lin, R. Bettati, and J.-Y. Chung, "Imprecise computations," *Proceedings of the IEEE*, vol. 82, no. 1, pp. 83–94, 1994.

STARTREC: Verification of a safety-critical system for autonomous vehicles

Marwan Wehaiba El Khazen
Inria and Statinf, Paris
marwan.wehaiba-el-khazen@inria.fr

Slim Ben Amor
Statinf, Paris
slim.ben-amor@statinf.fr

Liliana Cucu-Grosjean
Inria and Statinf, Paris
liliana.cucu@inria.fr

Arnaud Dumérat
EasyMile, Toulouse
arnaud.dumerat@easymile.com

Xavier Jean
EasyMile, Toulouse
xavier.jean@easymile.com

Kossivi Kougblenou
Statinf, Paris
kos@statinf.fr

Benjamin Monate
Trust-in-Soft, Paris
benjamin.monate@trust-in-soft.com

***Abstract*—In this paper, we present our on-going work on verification activities of the software used in a safety-critical embedded system dedicated to autonomous vehicles. These activities are focused on the use of formal methods for the verification of functional properties on the embedded code, and statistical methods for the analysis of its Worst-Case Execution Time (WCET). The project's goal is to address some technical barriers of software verification that will impact the safety demonstration of future autonomous driving systems. These barriers are challenging because of the high complexity of an embedded hardware and software, and appeal for methods and tools reaching the highest level of rigorousness.**

***Index Terms*—autonomous vehicles, formal verification, timing verification**

I. Introduction

The safety of autonomous vehicles is one of the numerous challenges this industry will face in the decade to come. In the latest years, the way to tackle this problem has become consensual [10]. An applicant needs first to establish an Operational Design Domain for its use case, summarizing the assumptions of use, hazard analysis and validation strategy at the system scale. This analysis is then refined to the subsystem level, assigning to each component a target integrity level (ASIL) associated to a safety function. The safety demonstration against this function at the appropriate integrity level is conducted under functional safety norms, such as the ISO 26262 [11]. The STARTREC project contributes to this second phase, by addressing the validation of a software component against the ISO 26262.

Even if its functional scope is clearly defined, the verification of a piece of software implementing an autonomous driving function constitutes a major issue with regard to many aspects [2]. It covers topics such as the correctness of algorithms [8], the correct implementation of these algorithms as well as their infrastructure software, being an operating system, a middle ware or third-party libraries, and finally the validation of their real-time properties when deployed on embedded hardware [1].

The component under verification in the STARTREC project is a safety relevant algorithm taking emergency decisions with regard to the environment, the sanity of the autonomous vehicle and its navigation stack. As part of the safe design process, this algorithm is designed to be explainable and verifiable in a white-box approach. It takes as inputs the perception infrastructure including the sensors of the vehicle, and various data produced by the navigation stack. It is interfaced with a safety relevant system capable of executing emergency maneuvers.

The activities planned during the STARTREC project are centered around the three following topics:

- Verify the correctness of the implementation of the algorithm embedded in the system. The correctness of the algorithm itself does not fall in the scope of the project. Related activities are centered around the use of sound static analysis methods and their combination with unit tests and integration tests. Sound methods can prove the absence of some categories of bugs [15], which makes them highly suitable for high integrity systems, but hard to deploy in practice on large pieces of software.
- Evaluate the Worst Case Execution Time (WCET) of the embedded software implementing this algorithm. Related activities are centered around statistical and probabilistic WCET evaluation methods [5].
- Demonstrate the compliance of the tools involved in the verification activities with regard to the ISO 26262 requirements, at the appropriate *Tool Confidence Level* (TCL). This compliance allows the applicant to consider the results provided by the tools as trustworthy.

This paper summarizes the overall problem, details the position of the project's stakeholders on the activities in progress, and presents preliminary results on the probabilistic WCET estimation tool.

II. Problem Statement and Related Work

The STARTREC project addresses the functional verification of the embedded software with the use of sound static analyzers, namely frama-c [14], and TIS-Analyzer [25]. These tools are used to prove (i) that the software is free from runtime errors, and (ii) that it complies with its functional specification. One challenge to address is the seamless integration of these tools with standard verification and validation processes:

radical changes to traditional processes are costly in terms of project management and risky in term of certification [19].

Runtime errors cover a large class of software misbehavior. Such errors encountered in embedded software are typically divisions by 0, memory overflows, out-of-bounds accesses, violations of flow integrity, typically dangling pointers or uninitialized variables. The numeric stability of floating point operations is also under consideration. Runtime errors are classically tackled by defensive programming techniques, but this raises a problem of software testing. Indeed, defensive programming introduce a systematic verification of each condition that could trigger a runtime error. This makes the test activity painful, as every branch shall be tested. For each condition a test case shall be specified and developed according to the quality insurance process. The proof of runtime errors absence, presented by Moy et al. [18] as the *"silver level"* of formal verification, allows to reduce the need of defensive programming by removing unnecessary checks at runtime. The method used in the project is abstract interpretation. It can be deployed on large pieces of code with a minimal effort of code annotation, but the analyzer's accuracy trends to decreases while progressing in the software analysis. Therefore, the main challenge is to analyze the algorithm while providing a reasonable set of annotations to keep a sufficient accuracy.

The proof of functional correctness, presented by Moy et al. [18] as the *"gold level"* of formal verification, relies on deductive verification. Frama-c implements this method in the Wp plugin. However, its use demands a high amount of code annotations, and reaches the limits of automatic provers on various aspects, like the handling of floating point numbers or the memory model associated with the representation of pointers. Although the STARTREC project will support the improvement of tools in these areas, proving the functional correctness of the whole software is considered not achievable in the frame of the project. A more pragmatic approach combining unit tests and unit proofs [14] is preferred, with the use of executable annotations.

The second theme of STARTREC is the evaluation of the software's Worst Case Execution Time (WCET). This activity is complicated for the following reasons:

- The software has many inputs impacting the execution time. Typically, sensors generate point clouds whose size and disparity varies with the environment, the meteorological conditions, and the sunlight. The scenario leading to the WCET of the algorithm is far from being reproducible on a real scene, so that the complexity of these inputs remains empirical. It is necessary to understand, build and justify the diversity of the situations encountered in the vehicle's environment to gather a set of recordings sufficient to conduct a meaningful WCET analysis. Note that every situation at a vehicle level does not need to be considered as long as it is demonstrated that they are equivalent to other situations, from a WCET point of view.
- The embedded software is to be deployed on a multi-core processor, able to execute several tasks in parallel. This leads to interference in the underlying hardware, when several cores compete to access shared resources, typically the memories, with complex patterns [3], [21]. At a high-level, this can be observed as a slow-down on every task. Estimating an upper bound for the interference is known to be an open problem in general case, with pathological situations that can be considered as denial-of-service attacks [17], the slow-down factor can be an order of magnitude [20]. We tackle this problem with the following restrictions: (i) the whole stack of embedded software is known, (ii) no piece of software has been designed to maximize the interference, and (iii) the scheduling of tasks might be reconsidered to limit the impact of interference.
- For WCET analyses and interference analyses, identifying the potential sources of interference as well as the size of the benchmarks set is also complex [12]. One typical problem of interference analyses is justifying why their results are trustworthy and provide certification credit.

Statistical methods deployed in the project, as described thereafter, are focused on the detection of inputs that impact the execution time, so that they do not need to be considered independently in the benchmark set. A similar approach is developed for interference analysis on multi-core microprocessors.

Any measurement protocol of the execution time of a program should take into account the fact that the actual WCET might not, or rather almost certainly will not be achieved. Rather, a set of highly unlikely but observed "extreme" worst-case values should be considered to represent the general behavior of all worst-case values. Then, assuming an appropriate theoretical foundation whose hypotheses would be checked, a statistical estimation of the WCET can be produced, along with an appropriately small probability that the actual WCET is higher than that estimation. Extreme Value Theory (EVT) provides such a probabilistic framework, much needed for the estimation of the statistical WCET. Its central result, the extreme value theorem, was given by Gnedenko [9] and describes the possible distributions of the extreme values. With a finite set of observed extreme values, one can then estimate the parameters of the whole distribution of extremes and give probabilistic guarantees pertaining to execution times much larger than the largest observation. The use of probabilistic performance guarantees and of distributions of execution times was originally suggested for scheduling by Tia et al. [22]. Later, these results have been consolidated and expanded to different aspects of real-time systems beyond scheduling by an important thread of results [5]. By using EVT, the authors provide estimations of the distribution bounding a sequence of measurements, when such bound exists. However, previous work [16] underlines that there are two associated problems while applying EVT to the statistical WCET estimation problem. The first one concerns the measurement protocol and whether or not it produces measurements that are representative of the actual distribution of the execution times

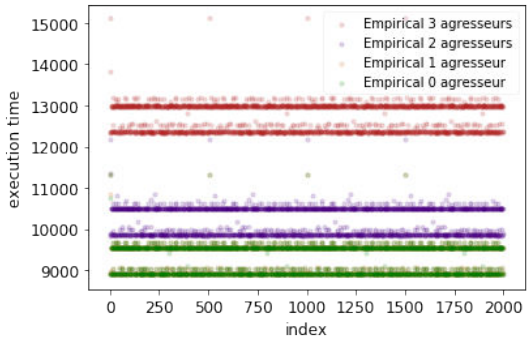


Fig. 1. The execution time sequence of the *minver* program on T1040

of a program. The second problem increases the tightness of the statistical WCET estimation while of course keeping the same level of probabilistic guarantees. Indeed, it may be counter-intuitive, but statistical estimators can be more pessimistic if they lack sufficient information on the sequence of measurements.

III. First results

In this section we provide first results of statistical methods applied to the interference analysis integrated within the WCET estimation.

Within the STARTREC project, we use a new Python library that has been designed by StatInf to fill the gap of EVT-based estimators of the statistical WCET in Python, while increasing their robustness as described in [24] using a combination of existing statistical estimators and novel methods. Moreover, the tightness of the statistical WCET EVT-based estimators has been improved by proposing a new statistical test, the WKS test [23]. The open problem that we will solve within the STARTREC project is the representativity one. We split this problem into three main parts:

- The representativity with respect to the program structure,
- The representativity with respect to the variation of the input variables,
- The representativity with respect to the interference analysis due to the presence of several cores.

The first problem may be solved by a hybrid approach which implies engineering effort putting together existing results, while the second problem has been formalized through a recent preliminary result [4]. In this paper, we deal with the representativity of measurements with respect to the third problem, the interference analysis.

To illustrate our strategy and for the sake of simplicity, we consider the program *minver* of the TACLeBench [7] executed on the QorIQ T1040 which is a Quad-core processor designed by NXP [13]. It belongs to the T1 family of QorIQ using the e5500 Power PC 64-bit single threaded core at 1.4 GHz with a private L1 cache (data and instruction caches) and a private L2 unified cache. It also has a CoreNet Platform Cache (shared L3 cache) associated to a DDR controller. In this paper, we consider e5500 cores and its private caches, as well as on the shared cache. The program *minver* is executed by imposing a specific memory location for the matrix to be inverted, while the adversaries are accessing the same memory location from different cores. The sequence of measured execution times is illustrated in Figure 1, where the horizontal axis corresponds to the order of execution times, while the vertical axis corresponds to the values of the execution times. We may note that such strategy is applied to the Easymile programs and other future publications may include comparison to other existing benchmarks [6], if publicly available.

In Figure 2, we represent the statistical WCET estimation obtained by using two possible EVT estimators, integrated within the StatInf tool and described in [24].

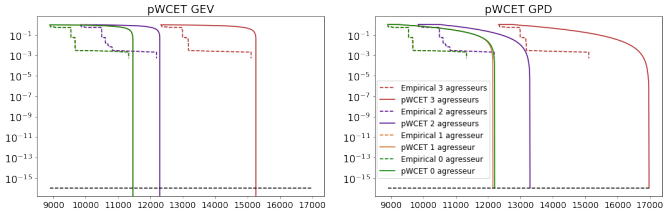


Fig. 2. The statistical WCET estimation using two possible implementations for EVT as implemented by the StatInf tool [24]

There are two possible EVT estimators, GEV and GPD, that first select maxima from an ordered sequence of execution times, and then they estimate the WCET distribution. The GEV estimator uses the block maxima approach by dividing data into several blocks with the same size and select the maximum of each block. While the GPD estimator selects the largest values above a given threshold (see Figure 3).

The statistical WCET estimator allows to quantify the impact of programs ”assaulting” the execution of the target program, *minver*. This aggression has been manually built for the particular case of the *minver* program. The purpose of STARTREC is to automate such a process with respect to a given configuration of a processor. Existing work focuses on providing interference analysis results under the hypothesis of worst-case processor configurations. In reality, the user comes with a configuration of the processor that has been tailored to meet other design constraints, such as security concerns, and a reduced configuration space is to be explored.

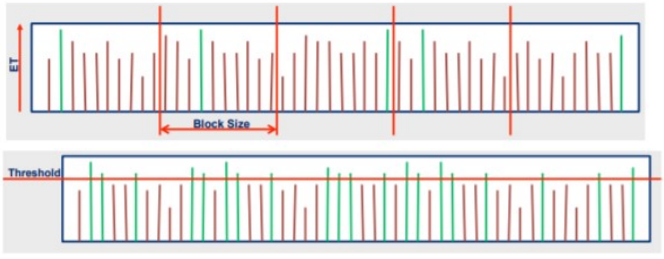


Fig. 3. The two EVT approaches are either picking the appropriate block size or the threshold.

IV. Conclusion and future work

Ongoing work within the STARTREC project shall bring answers concerning future development projects of safety critical software, with an objective to comply with normative activities, ISO 26262 or any other norm dealing with functional safety. The authors believe that the specification of software safety requirements will be enriched by the use of formal specification. The tooling scalability on large examples will determine whether this may be used or not for software verification. Embedded software testing will also benefit from formal specification as it allows the production of testing code for the specified pre-conditions and post-conditions.

The STARTREC project will contribute to tools and methods that extend the existing processes incrementally. The scale-up challenge for the tools and frameworks will take into account the following constraints:

- An industrial process based on Continuous Integration and Continuous Validation,
- Training and mentoring needs for embedded software developers, and auditors,
- All necessary activities to achieve the confidence in the use of software tools and comply with the normative framework, at the appropriate confidence level.

Within the STARTREC project, we also aim at performing Worst Case Execution Time analyses on the software used in the safety critical system, and justify their statistical significance by relying on the Extreme Value Theory. This aspect is challenging because of the complexity of the software and the hardware, leading to computation intensive phases interleaved with numerous data access and bus access patterns that are not humanly addressable. Here the authors claim that the progress of data analysis, combined with a deep knowledge of the underlying hardware, will offer the necessary support for engineers to produce optimized and predictable software with stringent timing constraints.

This approach will leverage the trust that formal verification and statistical WCET analysis brings to safety critical software while limiting the risk of software developments in terms of budget, timeline and certification.

V. Acknowledgment

This research is partially funded by the FR PSPC STARTEC supported by Bpifrance and La Région Occitanie, and the CIFRE StatInf-Inria agreement.

References

[1] Miguel Alcon, Hamid Tabani, Leonidas Kosmidis, Enrico Mezzetti, Jaume Abella, and Francisco J. Cazorla. Timing of autonomous driving software: Problem analysis and prospects for future solutions. In *IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS*, pages 267–280. IEEE, 2020.

[2] D. Buttle. Real-time in the prime time, keynote talk. In *Euromicro Conference on Real-Time Systems (ECRTS)*, 2012.

[3] Cédric Courtaud, Julien Sopena, Gilles Muller, and Daniel Gracia Pérez. Improving prediction accuracy of memory interferences for multicore platforms. In *2019 IEEE Real-Time Systems Symposium (RTSS)*, pages 246–259. IEEE, 2019.

[4] Liliana Cucu-Grosjean, Avner Bar-Hen, Yves Sorel, and Hadrien Clarke. The impact of the period variation on execution time distributions of programs. In *the 27th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, Aug 2021.

[5] R. I. Davis and L. Cucu-Grosjean. A survey of probabilistic timing analysis techniques for real-time systems. *LITES*, 6(1):03:1–03:60, 2019.

[6] F. Cazorla et al. https://people.ac.upc.edu/fcazorla/archives/muBT-brochure-jan2017.pdf, 2017.

[7] Heiko Falk, Sebastian Altmeyer, Peter Hellinckx, Björn Lisper, Wolfgang Puffitsch, Christine Rochange, Martin Schoeberl, Rasmus Bo Sørensen, Peter Wägemann, and Simon Wegener. TACLeBench: A benchmark collection to support worst-case execution time research. In *the 16th International Workshop on Worst-Case Execution Time Analysis (WCET 2016)*, volume 55, pages 2:1–2:10, 2016.

[8] W.H. Freeman and Company, editors. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. 1979.

[9] B.V. Gnedenko. Sur la distribution limite du terme maximum d'une seris aleatoire. *Annals of Mathematics*, 44:423–453, 1943.

[10] Road vehicles — safety of the intended functionality. Standard, International Organization for Standardization, Geneva, CH, 2019.

[11] Road vehicles — functional safety. Standard, International Organization for Standardization, Geneva, CH, 2018.

[12] Xavier Jean, Laurence Mutuel, and Vincent Brindejonc. Assurance methods for cots multi-cores in avionics. In *2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*, pages 1–7, 2016.

[13] Kossivi Kougblenou, Rihab Bennour, Adriana Gogonel, and Liliana Cucu-Grosjean. Work-in-progress: Towards representative measurement protocols. In *the 41st IEEE Real-Time Systems Symposium (RTSS)*, pages 419–422, 2020.

[14] Viet Hoang Le, Loic Correnson, Julien Signoles, and Virginie WIELS. Verification Coverage for Combining Test and Proof. In *12th International Conference on Tests and Proofs*, Toulouse, France, June 2018.

[15] Benjamin Livshits, Manu Sridharan, Yannis Smaragdakis, Ondřej Lhoták, J Nelson Amaral, Bor-Yuh Evan Chang, Samuel Z Guyer, Uday P Khedker, Anders Møller, and Dimitrios Vardoulakis. In defense of soundiness: A manifesto. *Communications of the ACM*, 58(2):44–46, 2015.

[16] Cristian Maxim, Adriana Gogonel, Irina Mariuca Asavoae, Mihail Asavoae, and Liliana Cucu-Grosjean. Reproducibility and representativity: mandatory properties for the compositionality of measurement-based WCET estimation approaches. *SIGBED Rev.*, 14(3):24–31, 2017.

[17] Thomas Moscibroda and Onur Mutlu. Memory performance attacks: Denial of memory service in multi-core systems. In *USENIX Security Symposium*, 2007.

[18] Yannick Moy. Climbing the software assurance ladder - practical formal verification for reliable software. *Electron. Commun. Eur. Assoc. Softw. Sci. Technol.*, 76, 2018.

[19] Yannick Moy, Emmanuel Ledinot, Hervé Delseny, Virginie Wiels, and Benjamin Monate. Testing or formal verification: Do-178c alternatives and industrial experience. *IEEE Software*, 30(3):50–57, 2013.

[20] Jan Nowotsch, Michael Paulitsch, Daniel Bühler, Henrik Theiling, Simon Wegener, and Michael Schmidt. Multi-core interference-sensitive wcet analysis leveraging runtime resource capacity enforcement. In *2014 26th Euromicro Conference on Real-Time Systems*, pages 109–118. IEEE, 2014.

[21] Ahsan Saeed, Daniel Mueller-Gritschneder, Falk Rehm, Arne Hamann, Dirk Ziegenbein, Ulf Schlichtmann, and Andreas Gerstlauer. Learning based memory interference prediction for co-running applications on multi-cores. In *2021 ACM/IEEE 3rd Workshop on Machine Learning for CAD (MLCAD)*, pages 1–6, 2021.

[22] T.-S. Tia, Z. Deng, M. Shankar, M. Storch, J. Sun, L.-C. Wu, and J.W.-S. Liu. Probabilistic performance guarantee for real-time tasks with varying computation times. In *Real-Time Technology and Applications Symposium*, pages 164–173, 1995.

[23] Marwan Wehaiba El Khazen, Liliana Cucu-Grosjean, Adriana Gogonel, Hadrien Clarke, and Yves Sorel. WKS test, a local unsupervised statistical algorithm for the detection of transitions in timing analysis. In *the 27th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, August 2021.

[24] Marwan Wehaiba El Khazen, Adriana Gogonel, and Liliana Cucu-Grosjean. Work in Progress: Lessons learnt from creating Extreme Value Libraries in Python. In *the 41st IEEE Real Time Systems Symposium (RTSS)*, Dallas / Virtual, United States, December 2020.

[25] Jakub Zwolakowski. Trust-in-soft ebook, 2020.

Session We.3.A

Memory Management

Wednesday 1st June

15:00

–

Amphithéâtre

Dynamic Memory Management in Critical Embedded Software

Category: regular paper
Authors: Cyrille Comar (AdaCore), Claire Dross (AdaCore), Florian Gilcher (Ferrous Systems), Yannick Moy (AdaCore)
Keywords: critical embedded software, dynamic memory management, program proof

Memory management has always been a delicate issue in critical embedded software because memory is often a scarce resource and many of the typical software errors jeopardizing the integrity of the execution of the software are related to memory mismanagement. Furthermore, critical software has had a tendency to grow in size and complexity in recent years, because it is using more and more complex algorithms in the critical parts of a system. The push towards autonomous mobility is a good example of the drivers for complexity reaching the most critical parts of software controlling such systems. This added complexity requires added flexibility in memory management that is not compatible with the traditional memory management techniques used for critical embedded software. In this paper we will first go over the traditional memory management limitations and the reasons behind them, we will then explore possibilities for going beyond them while being able to provide a high level of guarantees of correctness with regard to memory usage.

Dynamic Memory Management in Critical Software

We usually distinguish different kinds of data depending on the complexity of their memory management life cycle. The simplest is statically-allocated data, which is allocated once at the start of the application and is never deallocated. The case of dynamically allocated data is far more complex. It includes both stack-allocated and heap-allocated data. Stack-allocated data is data locally allocated by a subprogram in its activation frame, so there might be multiple instances of the same data if the subprogram is recursive, and the maximum memory usage depends on the call-graph of the program. Heap-allocated data can be allocated at any point, and deallocated at any later point, with associated risks of accessing deallocated data, losing access to allocated data or even deallocating already deallocated data.

The only requirement for safe use of statically-allocated data is that it fits in memory. This is similar to the requirement that the code fits in memory, and is checked by comparing the size of the corresponding segment in the executable with the available memory size on the target platform. The corresponding requirement for stack-allocated data is that it fits in memory at all times during the execution of the application. This is checked by performing a worst-case analysis of local memory usage, taking into account the call-graph of the application, and comparing it with the size allocated to the stack(s) on the target platform. Such a worst-case analysis can be made slightly difficult by the presence of cycles in the call graph, the existence of dynamically sized variables on the stack or the use of indirect calls but professional grade tools exist to perform this kind of analysis (e.g. GNATstack for Ada). As stack-allocated data is deallocated at subprogram exit, there is another requirement that such data is not accessed after subprogram exit, which could happen when the address of such local data is stored in pointers. This requirement can be enforced by programming languages (in Ada, Java, OCaml, Rust) or by static analysis (in C, C++). There are many more requirements for heap-allocated data, besides avoiding the

errors previously mentioned. One also needs to make sure that fragmentation doesn't hinder memory allocation of large data pieces and more generally that enough memory is available at any time for the needs of the application.

Given the difficulty of guaranteeing that these requirements are satisfied, many coding standards for critical software forbid heap allocation either completely or after initialization. The most common pattern allowing heap allocation in critical software consists in an initialization phase where memory is dynamically allocated, and never deallocated. Thus, the only requirement for such a pattern is that there is enough memory for allocations to succeed during this initialization phase which is relatively easy to meet.

However, more liberal use of dynamic memory should also be possible in critical software, provided the associated requirements are adequately addressed. In the following, we refer to the subsections and risks of section OO.D.1.6.1 of the DO-332/ED-217 Object-Oriented Technology and Related Techniques Supplement to DO-178C and DO-278A avionics standards. It lists the following risks, phrased here alternatively as criteria that a memory management should meet:

a. **Risk:** Ambiguous references. **Criterion:** The allocator returns a reference to a valid piece of memory, not otherwise in use.
b. **Risk:** Fragmentation starvation. **Criterion:** If enough space is available, allocations will not fail due to memory fragmentation.
c. **Risk:** Deallocation starvation. **Criterion:** An allocation cannot fail because of insufficient reclamation of inaccessible memory.
d. **Risk:** Heap memory exhaustion. **Criterion:** The total amount of memory needed by the application is available (that is, the application will not fail because of insufficient memory).
e. **Risk:** Premature deallocation. **Criterion:** An object is only deallocated after it is no longer used.
f. **Risk:** Lost update and stale reference. **Criterion:** If the memory management system moves objects to avoid fragmentation, inconsistent references are prevented.
g. **Risk:** Time-bound allocation or deallocation. **Criterion:** Allocations and deallocations complete in bounded time.

Those risks and associated criteria can be grouped as follows:

- Criteria related to temporal memory safety, to ensure that accessed data is allocated (a.k.a. use-after-free, or Risk (e) "Premature deallocation") and that data is not deallocated multiple times (a.k.a. double-free)
- Criteria related to memory availability, which requires in particular that access to allocated data is not lost (see Risks (b) "Fragmentation starvation", (c) "Deallocation starvation" and (d) "Heap memory exhaustion")
- Criteria related specifically to the garbage collection techniques. The time at which memory is deallocated, and the running time for deallocation, are in general unpredictable, which is a problem for real-time critical software. See also Risk (g) "Time-bound allocation or deallocation". The garbage collector may move data to avoid memory fragmentation, which requires all references to the data to be updated to its new location. See also Risk (f) "Lost update and stale reference".

DO-332 lists three main techniques for dynamic memory management: object
pooling, activation frame based object management (divided into stack allocation and scope allocation), and heap based object management (divided into manual heap allocation and automatic heap allocation).

The following table summarizes how criteria are addressed with each technique, detailing whether the application code (AC) or the memory management infrastructure (MMI) is responsible for it:

Table OO.D.1.6.3 Where Activities Are Addressed For DMM

Technique	Activities (OO.6.8.2)						
	a	b	c	d	e	f	g
Object pooling	AC	AC	AC	AC	AC	N/A	MMI
Stack allocation	AC	MMI	MMI	AC	AC	N/A	MMI
Scope allocation	MMI	MMI	MMI	AC	AC	MMI	MMI
Manual heap allocation	AC	AC*	AC	AC	AC	N/A	MMI
Automatic heap allocation	MMI	MMI	MMI	AC	MMI	MMI	MMI

*AC = application, MMI = memory management infrastructure, N/A = not applicable, and * = difficult to ensure by either application or MMI*.

Ease of use correlates with the larger dependency on MMI for addressing criteria: automatic heap allocation (garbage collection) only leaves to AC the need to verify that there is indeed enough memory for the application to run, while manual heap allocation only leaves to MMI the need to verify that allocation and deallocation operate in bounded time. Garbage collection is thus far superior in terms of usability, at the cost of offsetting a lot of responsibilities to the MMI, which comes with major challenges for certification.

Use Case: Message Handling

In order to explain the main memory management techniques, we consider a use case of message handling. The application receives a message which it stores as an array of bytes. For efficiency, this array of bytes should not be copied, but instead a pointer to the dynamically allocated array should be passed to parts of the application that need access to the message. When handling of the message is complete, the corresponding memory should be reclaimed. Let's consider how the five memory management techniques introduced previously address this use case.

In object pooling, a different memory pool must be allocated statically for all objects of a given type. Thus, this requires creating a memory pool for all the different sizes of arrays, or at least enough intermediate values of sizes so that not too much memory is wasted as padding. Then, it's entirely up to the AC to ensure correct pool usage, making sure stale references to arrays are not used for example after the corresponding array has been reclaimed. When objects of a pool are of the same size, object pooling helps address Risk (b) "Fragmentation starvation" as it eliminates fragmentation within a pool, but the memory remains fragmented between pools.

Stack allocation and scope allocation are not applicable here, as we need to return the allocated array from the activation frame in which it was created.

In manual heap allocation, we are in a similar situation as with object pooling, only with one pool. In particular, objects are not all of the same size, so Risk (b) "Fragmentation starvation" is particularly

difficult to address with this technique, as outlined in the table from DO-332. Allocators like jemalloc mitigate fragmentation by internally using multiple arenas for different allocation sizes [JEMALLOC].

In automatic heap allocation, memory is allocated as needed when creating the data structure from the incoming message, and from that point on, the AC needs not be concerned with dynamic memory management. The MMI is in charge of ensuring that all other criteria (except availability of enough memory) are ensured. Addressing fragmentation in particular requires moving allocated data to ensure larger contiguous patches of memory are available for allocation, which greatly increases the complexity of the MMI. Similarly for addressing time-bound allocation and deallocation, which makes it necessary for the garbage collector to operate in chunks instead of in one go.

Use Case: Current Practice and Challenges

In cases where it applies, garbage collection clearly provides the best user experience. But the cost of certifying the MMI prevents this technique from being used in many cases. Real time Java was the only effort to adapt garbage collection to the needs of critical real-time software, by providing the means to exempt regions of code from garbage collection and use region-based memory management instead, but it has not seen much adoption. [SCJAVA, USCJAVA]

This leaves object pooling and manual heap allocations as the only two applicable techniques, with the same assignment of most responsibilities to AC for ensuring criteria a-b-c-d-e are satisfied. This is a major challenge in certification, which explains why dynamic memory management remains difficult to use in critical software, beyond the previously mentioned pattern of use only during initialization.

The Ownership Approach to Dynamic Memory Management

Ownership is an approach whereby, at any program point, an entity within the program is statically identified as the “owner” of a piece of dynamically allocated memory. Only the owner of a piece of memory can deallocate it, and an analysis tool ensures statically that the deallocated memory cannot be accessed afterwards through the owning pointer or one of its aliases. Ownership rules are generally useful to describe data structures that only create lists or trees, and never direct acyclic graphs or cycles or arbitrary graphs.

The central principle in the ownership approach is that assigning *moves* the ownership from the source to the target of the assignment. The owner of a piece of memory has exclusive read-write access to the memory, which includes the (implicit or explicit) right to deallocate the memory. The owner of a piece of memory can also grant temporary read-write access to a single other object, or temporary read-only access to multiple objects, after which ownership is transferred back to the original owner, this is often called *borrowing* or *lending*. This is typically done when passing a pointer as a parameter to a call.

Adhering to the ownership approach has several benefits for ensuring the safe use of dynamic memory:

- It guarantees temporal memory safety (no use-after-free or double-free) - Risks (a) and (e).
- It guarantees that memory is not lost (no memory leak) - Risk (c).
- It does not rely on garbage collection for releasing memory safely - Risks (f) and (g).

Thus, ownership only leaves Risks (b) and (d) that should be addressed at the level of the AC. In addition, it provides the basis for addressing safe concurrency (no data races) and formal verification (with pointers). Two programming languages used in critical software, Rust and SPARK, have implemented the ownership approach, with a focus on safe concurrency in Rust, and on formal verification in SPARK. We're going to consider both.

Rust Approach to Ownership

Rust is a general purpose programming language originally developed by Mozilla Research. Coming from an organisation with a large C++ codebase, Rust aims to make systems programming at scale safer by enforcing strict rules around memory usage. It does so at a *type level*.

Rust is often characterised around the borrow checker, allowing safe referencing of values. But Rust uses ownership as the governing principle for the whole programming language. Every value introduced into a program has exactly one *owner*. Ownership is gained by introducing the value (independent of the location) and lost by *dropping* the value or *passing* it. In general, dropping happens if a value reaches the end of a scope without being passed on. The regions where a value is alive define the *lifetime* of the value. Those lifetimes are later used by the borrow checker to prove that references are safe. Rust strictly bans double ownership of an allocation. Rust also disallows *giving up ownership* while there are still references existing on the owned value.

As an example, Rust provides the *Box<T>* type for allocating a single value on the heap. The Box type *logically owns* its contents and is responsible in its implementation to ensure that it behaves like an owner. The implementation of Box is private and internally unsafe. Consider a possible expression in Rust of the message handling use case:

```
use std::sync::mpsc::channel;

type Message = [u8];

fn main() {
	let (sender, rcvr) = channel::<Box<Message>>();
	let task1 = move || {
		let msg: Box<[u8]> = Box::from([1,2,3,4]);
		sender.send(msg); // ownership is moved
		// payload is inaccessible here
	};
	let task2 = move || {
		if let Ok(msg) = rcvr.recv() { // ownership is gained
			println!("{:?}", msg);
			drop(msg); // drop is inserted implicitly, inserted for clarity
		}
	};
	task1();
	task2();
}
```

The use of the “move” keyword moves ownership of the referenced “sender” and “rcvr” endpoints of the channel to task1 and task2 respectively. Ownership of the message is passed on from task1 to task2 through the channel, ending with the message being deallocated by calling “drop” in task2, an action automatically inserted by the compiler when not done explicitly like here. This example illustrates a number of points: The correctness of this code relies on the correctness of the implementation of *Box* and *mpsc::channel. Box<u8>* expresses ownership of a range of bytes of dynamic size on the heap. Giving up ownership of a *Box* triggers immediate deallocation. *mpsc::channel* on the other hand is a primitive that passes owned types between potentially concurrent components. To the user, this is expressed through owning a *sender* and a *receiver.* The internals of both primitives are private and often internally unsafe.

The Rust language expresses certain assumptions (that Box *owns* the type it is constructed with its contents), the implementation of *Box<T>* ensures this by transparently heap-allocating the value and deallocating it when ownership of the Box is given up. It is also the burden of the library programmer to make sure that the internal pointer is always valid to dereference, never dangling while in use, always pointing to a valid heap allocation and that there is only one way to deallocate the value (by letting it run out of scope, giving it up). It is also the burden of the library programmer to ensure that destructors of logically owned values are called, if necessary. If correctly implemented in the library, the type cannot be misused.

Rust as such pushes the burden of implementing valid, memory-safe types to library implementers, giving them a number of rules to uphold to implement safe components. It then constructs a *facade* around the internal complexity, by using visibility rules - the internal are inaccessible to the user. Because *Box<[u8]>* only has a single owner, by proxy, we also know that the byte field (*[u8]) only* has a single owner (the owner of the box).

Rust's background of being developed for large codebases informs its outside-in approach: it trades ease of systems construction against implementation complexity of components. Heap-allocating types like Box and Vec have far more rules to uphold in their interface than in other programming languages such as C, for example, they are also not allowed to be moved while referenced. Rust strongly suggests reuse of known-safe components, a practice that is both found in its standard library and outside of it.

We can use this information to build a generic and safe to use API to express passing data between components:

```
fn send<T: ‘static>(t: Box<T>) {
    // ‘static declares that T  is not allowed to hold inner references
    // internal implementation
}
```

This signature does not only declare that a value of *Box<T>* is taken, but also that if passed, the caller *gives up* ownership, passing it to the sending component (which later passes it to the receiver). This holds true in both single-threaded and concurrent scenarios. The ability to fully pass ownership between components is crucial to Rust's concurrency guarantees, as it allows to avoid pessimistic locking and confusion at deallocation time. Other usages include resource modelling approaches like RTIC [RTIC-BOOK] that use ownership to model exclusive access to hardware resources.

Rust addresses Risk (a) by requiring the same behavior of an allocation system in use (that is, the underlying call to the memory allocator should not return a pointer to memory already in use). Rust prevents Risks (c) and (e) by providing a type system that makes it easy to pair allocations with

deallocations and clarifies the responsible component for the deallocation operation. Risk (f) is mitigated by the Rust language strictly disallowing references existing on data being moved. Given a memory allocator with the required behaviour, Risk (g) is prevented by Rust providing clear deallocation points and not deferring memory deallocation.

SPARK Approach to Ownership

Ada is a general purpose language designed for safety critical applications. SPARK is a subset of Ada aimed at formal verification. One of the major restrictions imposed by SPARK over Ada is the absence of aliases. The latest releases of SPARK support pointers without introducing aliasing, thanks to the use of an ownership model [CAV].

In SPARK, ownership is only concerned with pointers. Pointers in Ada are called *access types*. They are basically references, pointer arithmetics is only possible through a library and is not supported in SPARK. Their design is low-level, a pointer is null at declaration, it is possible to manually allocate data on the heap, or to take a reference to a stack-allocated object which has been explicitly marked as potentially aliased. The data allocated on the heap is not reclaimed automatically (Ada does not have a garbage collector), it has to be explicitly deallocated. In addition, to enforce some level of safety, Ada introduces a notion of *accessibility level* associated to all access types. This level is used to statically enforce that data allocated on the stack in a function can never be referenced by a pointer of a type declared outside of the function, therefore ensuring that the data will not be accessed once it has been popped from the stack. Finally, Ada makes a difference between *pool-specific access types*, which are necessarily heap-allocated, and *general access types* which may be either heap-allocated, stack-allocated, or even statically allocated.

Pointer support in SPARK was designed to stay compatible with the usual semantics of pointers in Ada. As a result, pointers are allowed to be null, and, if they have been allocated on the heap, they should be deallocated manually (no automatic reclamation is done when the scope of an object is exited). To be able to verify that stack-allocated data is never deallocated, deallocation is only allowed in SPARK for pool-specific access types.

An ownership policy is used to ensure that pointers cannot create visible aliases in the program, namely, either there exists only one pointer that can be used to access the data and modify it, or there exist several pointers that can access the data but only for reading. As discussed above, this is necessary so that formal analysis remains tractable and efficient (the analysis tool can continue to assume that there can be no aliases in the program). Together with the Ada semantics, this allows to ensure the memory integrity of SPARK programs and prevent Risk (e) as follows:

- *A pointer designating stack-allocated or statically allocated data is never freed.* This comes from the fact that objects of a general access type can never be deallocated in SPARK.
- *A pointer which is not null always designates valid data.* This follows from the accessibility rules of Ada for pointers designating stack-allocated data. For heap-allocated data, this is a result of the ownership policy, which ensures that when something is deallocated through a pointer, it cannot be accessed through any other pointers, together with the fact that deallocation nullifies the deallocated pointer in Ada.

The properties above are ensured in SPARK by construction, that is, by the semantic checking of the SPARK language, without running the formal verification tool.

In addition, the static verification tool for SPARK programs considers both dereferencing a null pointer and exiting the scope of an object when it still owns some heap-allocated memory as run-time errors, and will therefore attempt to verify that it never occurs. This prevents Risk (c).

SPARK addresses Risks (a) and (g) with the same requirements as Rust on the underlying allocation system to return valid pointers on allocation, and perform allocation and deallocation in bounded time. As just seen, SPARK verification prevents Risks (c) and (e). Risk (f) is mitigated by the SPARK language strictly disallowing references existing on data being moved. That leaves Risks (b) and (d) that should be addressed at the level of the AC.

Thanks to ownership, it is possible in the verification tool associated with the SPARK language to completely ignore the indirection associated to the pointer, and instead to consider pointers as composite types which are either null, or hold the value that they reference (similarly to *option* or *maybe* types commonly used in functional programming languages). This allows users to verify relatively complex heap-manipulating programs, involving for example recursive data-structures such as lists and trees, in an efficient way. The ownership policy allows the specifications to remain simple, as separation of memory segments remains implicit (contrary to explicitly having to state that all owned memory blocks are distinct from one another), at the cost of only being able to verify alias-free programs (no doubly linked lists or Directed Acyclic Graphs).

Here is how we could implement in SPARK the small example presented as a case-study. We define a type Bytes to be an array of an unknown number of elements of type Byte. We then define a type Ptr for pointers to such an array of bytes. Here we want to allocate data on the heap, so we use a pool-specific access type.

```
type Byte is mod 2**8;
type Bytes is array (Natural range 1 .. <>) of Byte;
type Ptr is access Bytes;
```

The function Alloc allocates an array of bytes of a specific size given as a parameter. The rules of SPARK require that the newly created pointer is stored in an object. During formal verification, the tool will make sure that this value is never discarded before being moved away or properly deallocated, so we can know that the memory will never be leaked.

```
function Alloc (Size : Natural) return Ptr is
   (new Bytes'(1 .. Size => 0));
```

The procedure Free deallocates a non-null pointer. Using an instance of the Ada.Unchecked_Deallocation generic is the normal way to deallocate data in Ada. It also sets its pointer parameter to null. Because of the ownership rule, we know that no other object can hold a reference to a pointer when it is given to Free, so the deallocated memory cannot be accessed after the free. So, while deallocation is indeed “unchecked” in Ada, it is fully formally verified in SPARK.

```
procedure Free is new Ada.Unchecked_Deallocation (Bytes, Ptr);
```

As an example of use of pointers, Swap exchanges two pointers without copying the memory around. The ownership of the memory initially designated by X is moved to Tmp and then Y, after the ownership of the memory initially designated by Y has been moved to X. Note that the ownership rules of SPARK will

require that Swap is always called on distinct pointers, even though its body would also handle the case where X and Y are the same correctly.

```
procedure Swap (X, Y : in out Ptr) is
   Tmp : constant Ptr := X;
begin
   X := Y;
   Y := Tmp;
end Swap;
```

SPARK formal verification tools can be used on this procedure to show that it correctly implements swapping of the underlying values of two non-null pointers:

```
procedure Swap (X, Y : in out Ptr) with
  Pre  => (X /= null) and (Y /= null),
  Post => (X.all = Y.all'Old) and (Y.all = X.all'Old);
```

Comparison Between the Rust and SPARK Approaches

Rust and SPARK approaches, while closely related, bring different benefits, which we will explore in this section.

Rust was the first language to popularize the ownership approach. It has evolved around this core principle, and consequently offers the most features around ownership. Ownership in Rust is implemented as a core concept of the language, enabling it in all contexts, allowing the modelling of lifetimes everywhere. Rust takes advantage of ownership to provide automatic deallocation of dynamically allocated memory. Rust aims at providing safe concurrent access to dynamically allocated memory and similar resources, and defines several standard libraries implementing standard approaches to dynamic memory management such as collections and a pointer module. A key benefit of the Rust approach is that it is implemented in the compiler, hence is enforced on all programs.

SPARK has adapted the ownership approach to the existing Ada rules regarding pointers and dynamic memory, using the existing notions of scopes and types of pointers. Thus, SPARK does not provide automatic deallocation of dynamically allocated memory, instead it allows checking that explicit deallocation does not introduce errors and does not leak memory. A key benefit of the SPARK approach is that it allows to prove properties of programs with pointers, starting with absence of runtime errors and extending to arbitrary security or safety properties expressed as contracts.

In a nutshell, program objects subject to ownership are not the same in Rust and SPARK: this concerns all objects in Rust, and only objects containing pointers in SPARK. Deallocation of dynamically allocated memory is automatic in Rust, while its correction is verified in SPARK. And enforcement of ownership is done by the compiler in Rust, by the verifier in SPARK. But both programming languages make it possible to create and reclaim objects subject to ownership via various means: this can be through a standard library like std::boxed::Box in Rust or a user library; this can be through standard allocation/deallocation in SPARK or a user library. Finally, it is possible in both Rust and SPARK to use libraries that internally violate the ownership principles, but should expose an API to client code in Rust or SPARK that is safe to use from code subject to ownership principles. This is done in Rust by using so-called “unsafe code” and in SPARK by using plain Ada code.

While Rust focuses on safe concurrency and SPARK focuses on formal verification, these shared benefits of ownership apply to both. The Rust example of sending a message across tasks can be written in SPARK too, and there are academic formal verification systems that can prove the Swap function in Rust too.

Applicability of Ownership to Other Languages

The term *ownership* was picked by Rust and later adopted in SPARK as it is a frequent concern also expressed in other languages, such as C/C++ and Java.

For C++, a common concern is the following question: given a pointer to a memory location, is the code handling that pointer allowed to deallocate (implying ownership) and mutate it (implying the question of aliasing). Both are sources of mistakes. Structured solutions for this exist. Particularly Rust's notion of ownership being paired with destruction is bringing the common C++ concept of RAII (Resource Acquisition Is Initialization) into the language directly. Rust also implements *moves* as an implicit core language operation coupled with passing ownership, expressed in C++ through the *std::move* function. Still, a lot of concepts exist in C++ as library concepts, such as *std::unique_ptr* for pointers that should not alias and hence should destroy the pointee when they are themselves destroyed (similar to the above mentioned Box), and *std::shared_ptr* for reference counted objects. C++ is lacking a general solution for ensuring reference validity, while having a very structured approach to using its modern std library components to signal intent. Pitfalls in those API still exist, as illustrated in [UNIQUEPTR].

For C, the situation is similar to SPARK, namely, there is little support in the language for checking memory safety of programs, but some can be added using an external analysis tool. There exist several tools targeting the C language, be they general-purpose analysis tools [FRAMAC] or specific tools targeting memory safety [CLOUSEAU], which could be used for that purpose.

Ownership concerns are also a large issue in concurrent programming, particularly the ability to cleanly move data and responsibilities from one component and thread to another. Early approaches to solving this problem can for example be found in introducing the notion of owned pools into Java in JCoBox [JCOBOX], where passing objects between actors in Java also forces them to give up all references to those objects for the passing actors. This work points to Java lacking a general solution for unwanted aliasing.

Given that ownership is an implicit concern in many languages and settings, raising it to a language level concern if possible is useful. This can be done as part of the core language like in Rust, or as part of a language subset like in SPARK. The former requires influence over the committee presiding over the language evolution, which is a large endeavour for most programming languages. The latter is easier to adopt, but it puts constraints on the features supporting ownership, as they must be compatible with the base programming language.

Application of Ownership to Critical Software

The ownership approach does not address all issues with the use of dynamic memory in critical embedded software. Memory availability remains an issue, both to ensure that there is enough available memory at all times, and that memory fragmentation does not make it unavailable for allocating larger

objects. Traditional solutions to address these issues have relied on static memory pools for objects of different sizes, with a suitable analysis of the program needs for each memory pool.

We will review the use of Rust and SPARK in critical software, and how ownership plays a role, concluding with a roadmap to further the applicability of both technologies to critical software.

Use of Rust in Critical Software

While Rust is not used in safety critical applications with regulatory concerns to date, it is used at many locations with major security concerns today (often referred to as “mission critical”). Rust being built for use in Firefox (a program with a massive user base and strong security concerns as an entry point for exploits) informs the ethos and the goals of the language. This reflects in project policies such as very strong requirements for supported software targets of the highest tier [TARGETPOLICY], requiring committed developers and fully automated testing of all changes, blocking release of the whole compiler, should bugs arise. Examples of further usage include use at many cloud providers such as AWS and Microsoft Azure. There is an expressed interest in moving Rust further into the critical spaces, with gaps to be filled, particularly enabling developers to better understand the tradeoffs that Rust makes and enabling them to easier implement safe base abstractions. The Ferrocene project [FERROUS] is currently underway to ensure that.

Use of SPARK in Critical Software

Since its inception in 1987, SPARK has been adopted in numerous large industrial projects to get as close as possible to zero-defect software. Typically, only critical parts of the software were proved “correct” with respect to full functional specification. More generally, SPARK was used to prove specific properties of interest about the software, like the absence of all possible run-time errors (no division by zero, no buffer overflow, etc.) and some user-specified safety or security properties. Thanks to its benefits for increasing software quality (and thus safety and security), SPARK has been a language of choice for the most stringent levels of certification domains: level A for avionics (DO-178), space (ECSS-E-ST40C), level 4 for railway (EN 50128) and automotive (ISO-26262).

Conclusion

To this day, certification standards for developing critical software applications strongly discourage the use of dynamic memory, due to the associated risks and the difficulty of demonstrating with sufficient confidence that these risks are adequately addressed. The avionics certification standard supplement DO-132 published in 2011 was the first effort to describe in detail the risks associated with the use of dynamic memory, and acceptable means of compliance. Since then, Rust has emerged as a new language for mission critical software, and it has popularized the ownership approach to dynamic memory management. This approach is not limited to Rust, as the example of SPARK has shown that it can be adapted for adoption in other languages, even supporting formal verification of pointer programs in SPARK.

Industrial users of Rust and SPARK are currently adopting this approach in the automotive domain. It remains to see how easily the objectives of ISO-26262 can be addressed when using ownership in both programming languages. The extension to other certification domains is a challenge for the near future, that others have started investigating for the avionics domain [VFS].

References

[SCJAVA] Thomas Henties, Siemens Ag, James Hunt, Doug Locke, Kelvin Nilsen, Aonix Na, Martin Schoeberl, and Jan Vitek. Java for Safety-Critical Applications. Electronic Notes in Theoretical Computer Science - ENTCS 2009.
https://www.researchgate.net/publication/228806774_Java_for_safety-critical_applications

[USCJAVA] Kelvin Nilsen. Unification of Safety-Critical Java. Embedded Real Time Software and Systems (ERTS2012), Feb 2012, Toulouse, France. https://hal.inria.fr/ERTS2012/hal-02263468v1

[FERROUS] https://ferrous-systems.com/ferrocene/

[SPARKPRO] https://www.adacore.com/sparkpro

[TARGETPOLICY] https://doc.rust-lang.org/rustc/target-tier-policy.html#tier-1-with-host-tools

[CAV] Claire Dross and Johannes Kanig. Recursive Data Structures in SPARK. CAV 2020

[JCOBOX] Jan Schäfer and Arnd Poetsch-Heffter. JCoBox: Generalizing Active Objects to Concurrent Components. 24th European Conference on Object-Oriented Programming (ECOOP 2010).
https://softech.informatik.uni-kl.de/homepage/publications/SchaeferPoetzschHeffter10jcobox.pdf

[UNIQUEPTR] https://bartoszmilewski.com/2009/05/21/unique_ptr-how-unique-is-it/

[JEMALLOC] https://engineering.fb.com/2011/01/03/core-data/scalable-memory-allocation-using-jemalloc/

[VFS] Max Taylor, Josh Ehlinger, Jeff Imig, Massimiliano De Otto: Rust for Safe and Secure Avionics and Mission System Software, Vertical Flight Society Forum, May 2021

[RTIC] https://rtic.rs/1.0/book/en/

[CLOUSEAU] David L. Heine and Monica S. Lam. A practical flow-sensitive and context-sensitive C and C++ memory leak detector. Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation.

[FRAMAC] Florent Kirchner, Nikolai Kosmatov, Virgiles Prevosto, Julien Signoles, and Boris Yakobowski. Frama-C: A software analysis perspective. Formal Aspects of Computing.

Certifiable Memory Management System for Safety Critical Partitioned System

Alexy Torres Aurora Dugo, Jean-Baptiste Lefoul, Serge Harnois, Felipe Gohring de Magalhaes and Gabriela Nicolescu

Abstract—Aerospace systems are safety-critical systems that need to respect tight constraints in terms of execution time, resource usage and predictability. This industry is currently transitioning from predictable single-core processors to less predictable multi-core architectures. This transition reveals multiple challenges due to interferences. The contention of different cores on shared resources introduces interferences. This phenomenon prevents the required isolation between applications and the estimation of their worst-case execution time. To prevent interferences and ease the certification of robust partitioned multi-core systems, guidance documents, such as the CAST-32A, provide objectives on resource isolation and management. In this paper, we propose a memory manager to mitigate memory interferences generated in shared cache, main memory and memory bus. Our results show an increase in timing predictability by 68.1%. Aside the memory manager, based on our results, we provide a set of recommendations to assist system integrators' decisions and ease the certification process by conforming to the current guidance.

Keywords—Aerospace, ARINC-653, Certification, Critical systems, Interference, Resource management, RTOS

I. Introduction

The aerospace domain relies on highly critical software to ensure the reliability and safety of the system. The first generation of computer-assisted functionalities in planes is based on the federated architecture. This type of architecture is designed such that each computer-assisted functionality has its controller or computer, called Line Replaceable Unit (LRU). LRU communicate through different networks and buses. Due to the increased number of computer-assisted features in planes, the use of federated architecture is impossible to sustain. The increase in communication nodes often results in network contention and poor fuel efficiency of the plane due to the equipment weight [1].

To leverage the size, weight and power (SWaP) of federated architectures, the Integrated Modular Avionics (IMA) architecture was proposed. The IMA architecture is designed such that multiple functionalities are gathered in the same module. This design eases the communication between applications, reduces the SWaP and facilitates the maintenance of the equipment. Multiple IMA units can be scattered in the plane and communicate via networks and buses.

IMA architectures rely on Real Time Operating Systems (RTOS) to manage the different components in the system and the different applications running concurrently. Real time systems are subject to strong constraints in terms of execution time and response latency. Hence, it is imperative to ensure the Worst-Case Execution Time (WCET) of all applications. Nowadays, IMA architectures are implemented relying on Commercial Off-The-Shelf (COTS) hardware [1]. This reduces the cost of the equipment but also reduces the design's flexibility.

Despite the determinism of single-core processors, more energy-efficient and powerful multi-core architectures slowly replace them [2]. Multi-core processors provide better performance by the ability to execute multiple applications at the same time. With the increasing production of multi-core architectures, single-core processors slowly disappear from the market. Aerospace system designers need to transition to multi-core architectures to keep the equipment up to date.

However, optimizations brought by multi-core processors make execution times less predictable and impact WCET. The execution of multiple parallel applications also introduces contention on the shared resources. These phenomena are called interferences [2]. Interference channels exist at different levels of the architecture. In this paper, we focus on memory interferences for which channels are the shared caches, shared memory bus and shared DRAM. We propose a memory manager to mitigate memory interferences. For this purpose, we extend the cache, memory and bus management methods proposed by the Single-Core Equivalent framework [3] to allow better integration with certifiable RTOS and without the use of virtualization. We propose a flexible design to accommodate with different architectures and certification processes. Our results show an increase in timing predictability by 68.1% on average while increasing the execution time of applications by 22.3% on average. The Certification Authorities Software Team (CAST) published a guidance paper, the CAST-32A, that provides objectives to be met when certifying multi-core critical systems. Based on this paper, we propose a set of rules and insights to the system integrator (SI) to improve safety and performance of the system [4]. Our work brings the following contributions:

1. We propose a novel memory manager enabling to reduce cache and memory interferences as well as the bus interferences. By integration of most efficient methods for cache and memory partitioning and bandwidth management, we obtain a solution applicable to certifiable systems in a context where hardware-assisted virtualization is not available;
2. Improving the bus bandwidth management by proposing an extension of MemGuard [5], a memory bandwidth manager, to allow more flexibility and I/O bandwidth

A. Torres, JB. Lefoul, F. Gohring de Magalhaes and G. Nicolescu are with the Department of Computer Engineering, Polytechnique Montreal, Montreal, QC, CANADA. email: alexy.torres-aurora-dugo@polymtl.ca

Serge Harnois is with Mannarino Systems & Software Inc., Montreal, QC, CANADA. email: Serge.Harnois@mss.ca

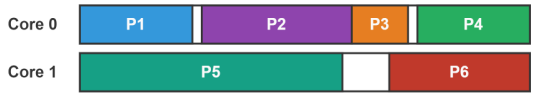


Fig. 1. Multi-core same-length MAjor Time frame defined on two cores

management by extending configuration capabilities, per application budgeting, and an initial budget pool;

3. The proposition of guidelines, named safety net as per CAST-32A [4], to ensure quality of service (QoS) and safety in the RTOS and ease the certification process in avionic systems. We further formalize the notion of robust partitioning for shared memory and bus.

II. Background and Context

To allow their certification, civilian aerospace systems must follow strict rules. Multiple standards exist to ease the development process and RTOS use (e.g., ARINC-653 [6]). In this section we present those requirements and the current challenges of certification in multicore systems.

A. Multicore Safety Critical Systems Challenges

The notion of robust partitioning implies that an application cannot impact any other applications in the system. This is true from a software perspective, where the undefined behavior of the application cannot block or impact the execution of another application [6]. This is also enforced in the hardware aspect, where an application should not change the state of the hardware to the point that it might impact another application. In this context, applications are called partitions [6].

ARINC-653 compliant systems must conform to the following constraints: (1) Time isolation: a partition must execute during a given time slot without being preempted. As shown in Figure 1, the scheduling relies on a MAjor time Frame (MAF) that is repeated indefinitely. (2) Space isolation: a partition can only access resources it has been explicitly allocated. For instance, a Memory Management Unit (MMU) can enforce space partitioning. CAST-32A [4] also proposes the integration of a safety net that should provide means to monitor and recover from failure in space and time isolation.

To allow the use of multi-core processors in critical systems two approaches are foreseen. The first is to take interferences into account during system analysis to consider their impact on timing. With this approach, one could account every access to the cache as a cache miss. This provides overly pessimistic results. The second approach is to bound or eliminate them to allow the use of known analysis methods. In this work, we allow robust partitioning of the shared resources. The objectives provided by the CAST-32A are abstract are sometimes difficult to transpose into system's constraints. Thus, we propose to formalize these objectives regarding shared memories and bus for robust partitioned systems in Section 5.

B. Interferences

In this section, we present the interferences that we consider in our work. We also describe the interference channels studied and their impact on the system's execution.

1) Cache Interferences: Cache interferences in multi-core systems appear in two contexts: shared caches and private caches [2]. Shared caches interferences occur when multiple cores share the same cache. Two types of interferences might appear in this context: (1) Contention interferences, when multiple cores try to access the cache at the same time, only one core is granted access and other core wait to access the bus. (2) Eviction interferences, when an application on a core evict data owned by another application on another core.

2) Bus Interferences: Bus or interconnect interferences occur when multiple components (cores, coprocessors, etc.) try to access the bus at the same time [5]. This contention results from the arbitration of the controller on the bus. Only one component may access and process transaction on the bus at a given time. Thus, the system puts other components that request the access on hold and wait for their turn to use the bus. Although multiple arbitration policies have been proposed to reduce or remove bus interferences, they are rarely available on COTS hardware.

3) Memory Interferences: Main memory interferences can be observed at different levels. Memory can be accessed through different independent channels without showing any interference [7]. However, if two core access the memory through the same channel, the contention on that channel generates interferences.

The second type of interferences appears at the bank level [8]. Each bank comprises a row buffer, which stores the content of the last accessed row in the DRAM. The row buffer acts as a cache that allows accessing the data contained in the same row faster. When multiple core access the same memory banks, the row buffer unpredictably changes its content to accommodate cores access patterns. Thus, affecting the execution time of partitions executing on the different cores.

C. Interferences Mitigation Means

Different mechanisms are present in COTS hardware to isolate components and make the platform more predictable. Table I provides a summary of the mitigation methods we discuss in this article. We also provide the availability of the solution, where it is said to be low when additional hardware is required and such type of hardware is usually not present in COTS processors. A high availability means that the hardware required to apply such technique is usually present in COTS processors (e.g., Performance Monitoring Counter (PMC), Memory Management Unit (MMU), etc.).

Cache way partitioning relies on hardware facilities to allocate ways of the cache to designated cores. Hardware specific registers must be set to allow segregation of the cache.

Cache set partitioning or cache coloring, relies on the structure of cache memories and the physical address of the data to know where the data will be placed in caches. Figure 3 shows the physical address layout that allows to know in which set the data will be loaded. Based on this information, the RTOS can wisely choose the virtual to physical translation to place pages in selected sets of the cache.

Finally, cache line partitioning is a combination of ways and set partitioning, where the RTOS allocate a set of cache ways and cache sets to a partition.

TABLE I
LIST OF INTERFERENCE MITIGATION MEANS AVAILABLE IN COTS HARDWARE

Mitigation method	Interference	Availability
Cache way partitioning	Shared cache eviction	Low
Cache set partitioning	Shared cache eviction	High
Cache line partitioning	Shared cache eviction	Low
Memory bank partitioning	DRAM Row buffer	High
Memory channel partitioning	DRAM channel	Low
Bus budgeting	Bus contention	High

Memory banks and channels partitioning work the same way as cache set partitioning. By wisely choosing the physical address of a data, the RTOS knows in which DRAM bank and channel the data are stored. Placing data of different partitions into different banks will remove bank interferences (row buffer interferences). Channel partitioning allows assigning unique channels to cores. When a core accesses the DRAM, it can do so without contention on its channel due to another core requesting data.

Bus budgeting relies on the monitoring of memory accesses done by a core. When the number of accesses made during a given period of time exceeds a certain amount, the core stops its execution by scheduling an idle task or halting the core. This method has the effect to stop the contending core and release the bus bandwidth it uses.

All the previously mentioned methods provide the same amount of isolation and can be used for safety-critical systems. The choice of a technique over another should be based on the availability of the hardware and the overhead it introduces.

III. RELATED WORK

In this section we present the current state of the art and position our work with regard to the interference channels we study in this article.

A. Cache Interferences

In [9], the authors present a survey of the different techniques for cache partitioning. Cache way partitioning is shown as the most used implementation. While line partitioning offers finer granularity, it also brings a higher execution and development overhead. Finally, set partitioning is known to be more complex to implement but offer better portability.

In [10], the UCP (Utility-Based Cache Partitioning) approach was introduced. It defines cache partitioning and monitoring hardware module to update the cache configuration at run-time. In the avionic domain, all configurations should be static and validated prior to system deployment as certification requires it [11]. Thus, we cannot apply dynamic approaches in this context. In [12] and [13], semi-partitioned caches are explored to improve performance while keeping the system in a more predictable state. The work of [14] extends this concept by clustering applications and partitioning caches on a per-cluster basis. In [15], the authors present a scheduling technique to account of the cache usage and reduce the performance hit introduced by cache partitioning.

In [16] and [17] cache partitioning is proposed alongside prefetching techniques. In our context, the system design often disables performance improvement facilities such as prefetching or branch prediction to reduce non-determinism produced by them. In [18], the authors use different processor management methods to reduce energy consumption. Literature also studied super pages with cache partitioning but the existing solutions use additional hardware [19].

In [20] and [21], hypervisors and the notion of virtual CPUs are proposed to ease cache partitioning implementation. The use of an hypervisor allows the application of cache partitioning to existing OS without modifying their code. However, it yields to an increased overhead in memory size and execution time.

B. Memory Interferences

In [22] the authors give a comprehensive list of memory interferences in COTS platform. The authors propose a methodology to analyze and understand the impact of the different mechanism in DRAM on predictability and latency. In [23] the authors propose a multi-policy resource allocation method. They use DRAM and cache partitioning together to leverage interferences in the system. Their approach relies on the analysis of a huge data set (2000 workloads) of execution (over 10000 experiments).

In [24], [25], [26] scheduling is explored to reduce interferences in the memory hierarchy. This approach allows reducing contention on the shared resources and thus, increase the throughput of tasks naturally. However, in [25], only one application can execute on multiple cores. Even-though scheduling reduces interferences, it does not ensure isolation between the partitions and makes certification more difficult in systems where robust partitioning is required [4].

In [27] the authors propose a software memory coloring approach to separate applications' memory between DRAM banks and channels. The method relies on a dynamic approach that changes the number of allocated banks and channels to threads at run-time. However, we cannot apply this method to hard real-time systems as the dynamic behavior would introduce non-determinism and make WCET estimation more complex due to the added factors to consider.

In [8], the authors coordinate the use of cache coloring and bank coloring. The duality of the approach and the conflict between cache and bank coloring is explained. In this paper, we review the approach proposed in [8] to allow faster integration and certification in safety-critical systems.

In [28] the authors propose a bank partitioning method to improve performance while considering the profile of the applications. The method relies on application profiling and clustering to categories their memory usage. However, we cannot use additional hardware to gather the metrics needed by the online algorithm. Similarly, in [29] the authors propose to isolate concurrent threads to use different memory banks for data sampling applications. Performances are further improved by balancing the load between the different memory banks.

C. Bus Interferences

Bus bandwidth management is a widely studied approach to leverage bus interferences. In [5] the authors present an approach called MemGuard. MemGuard regulates bus bandwidth

and propose different mechanisms to increase performances. Each core has an allowed amount of access to the bus during a quantum of time. If the core exceeds its budget, the RTOS puts it on hold until the next quantum of time. MemGuard uses per-core budget allocation and lacks configuration flexibility. These two main limitations that are further explained and addressed in Section 4.2.

In [25], [30] and [31], scheduling frameworks to leverage predictability in multi-core systems are proposed. The approaches use memory bandwidth throttling mechanisms to ensure bus and memory interferences bounding. In [32], the authors propose to isolate bus resources when hard real-time applications execute, thus, giving them exclusive access to the bus. In [33], critical applications are mapped to a single core, leaving the other cores to use by best effort applications.

In [21], the authors design a resource management framework based on virtualization to leverage cache partitioning and memory bandwidth limitation. The approach is based on the use of virtual CPU (vCPU).

Execution models were introduced to manage bus and memory interferences. In [34] and [35] the authors use the Acquisition Execution Restitution model (AER). The objective of this method is to separate the computation (Execution phase) from memory accesses (Acquisition and Restitution) during run-time and schedule the memory phases so only one partition can be in those phases at a time. Thus, the RTOS schedules access phases to ensure no contention occurs.

Other works rely on hardware mechanism to reduce, bound and remove interferences. In [24], [36], [37] and [38] memory controllers and arbiters are proposed to improve predictability while increasing the performance. However, these approaches are not compatible with COTS hardware as it requires additional hardware that is not certifiable in some cases. In [39] the authors study three techniques to manage memory bandwidth: thread packing, clock modulation and Intel MBA technology.

We cannot apply the exclusive use of resources at a given time such as the method proposed in [32]. We consider all the partitions in the system as highly critical. Applying this method would be useless as only one application could execute at a time. The approach provided in [30] reduces the overhead and improves the performance but we cannot afford to change the criticality of the applications nor provide different execution modes. Unfortunately, we cannot rely on the additional hardware components used in [40]. Moreover, contrary to the proposed method, isolation must be maintained at any moment in the execution window. Finally, our objective is to reduce applications WCET contrary to optimizing the average performance. Work such as [39] does not provide strict isolation of resources and thus, makes certification more tedious. In [3] the Single-Core Equivalence principle was proposed. We base our work on this approach and refine it to enable easier integration with non-virtualized environments. While virtualization allows a better management of resources, this method is not applicable in our context. We aim to rely on a bare-metal RTOS that does not allow virtualizing the system's resources and to reduce the overhead introduced by these techniques.

Using the AER model is not applicable in our context as we need to keep compatibility with legacy applications that do not make use of such model.

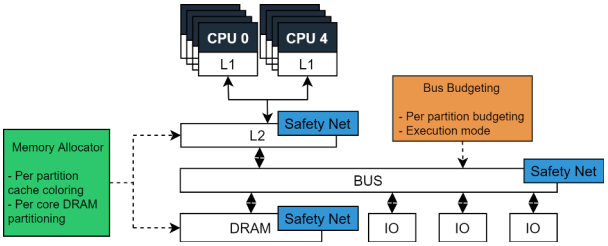


Fig. 2. Memory hierarchy with key points where our management framework applies.

In [41], the authors show that per application budgeting increases the system's performance and scheduling feasibility. In this paper, we provide an extension to MemGuard that allows per application budgeting instead of the per core budgeting previously allowed by MemGuard.

In this paper, we choose to use set based cache partitioning alongside with DRAM bank partitioning. Both approaches are portable to any architecture with virtual to physical address translation and are applicable to COTS hardware. For the same reason, we choose to extend the MemGuard approach that relies on COTS available features such as PMC (Performance Monitoring Counters) [5]. To enable portability, our solution can be extended for cache way partitioning use.

IV. Global Memory Manager

In this section, we present our centralized memory manager. This module extends the state-of-the-art approaches to ease their certification in the context of aerospace systems. Figure 2 provides an overview of the system's architecture with key points where our method applies. While the memory allocator manages how shared caches and DRAM is allocated across partitions, the bus budgeting module monitors and controls bus usage by the partitions. Finally, the safety net module is scattered across the memory hierarchy, to monitor the system's health. In case of an error, the safety net triggers an error to the heal monitoring manager. Throughout this section, we explain in detail each module and how they interact with each other.

A. Memory Management

Three main approaches exist to partition the cache. Way partitioning allows segregating caches in ways, by allocating them across cores. Set partitioning, also named cache coloring, profits from the virtual to physical translation mechanism to arrange the memory layout and allocate cache sets to different applications. Finally, line partitioning allows allocating cache lines to applications. Line partitioning is available when both set and way partitioning are possible. Set partitioning is privileged over way partitioning as it allows more cache partitions and is easily portable to different architectures. It is worth to be mentioned that our approach can be extended to use way or line cache partitioning with few modifications.

As discussed in [8], cache coloring (set partitioning) may interfere with memory coloring. Figure 3 shows the different memory layouts encountered during our study on architectures.

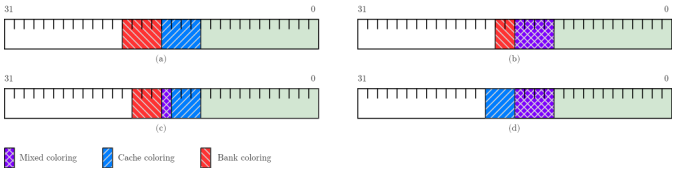


Fig. 3. Memory coloring layouts found in 40 different architectures. (a) allows separate allocation of banks and cache partitioning, (c) show overlapping bits while (b) and (d) show that bank coloring completely overlaps cache coloring and vice versa.

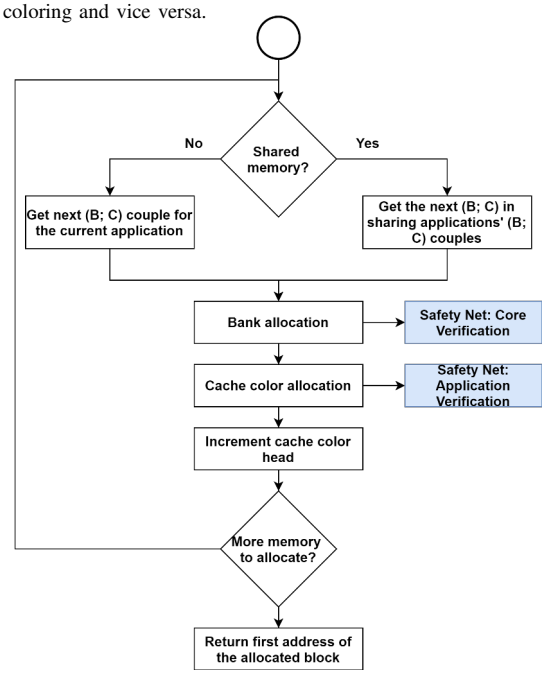


Fig. 4. Two-level memory allocator flow. We use the same process for both private and shared memory.

[8] also discuss the bank bit randomization technique used in modern processors. Such process is not present in all architecture but should be considered by the system designer when studying the memory address layout of the system. Our allocator takes into account overlapping bits in memory coloring to ensure complete isolation of the resources.

We propose a two-level memory allocator. First the allocator selects a memory bank based on the core the application is running on. In our design, we allocate banks on a per-core basis while allocating cache partition on per-application basis. Using private cache partitions for each applicative partition allows the systems to skip cache flush and invalidate when switching between partitions as recommended by the CAST-20 [42]. This method allows us to use cache more efficiently while reducing the partition switch time.

We define an allocation unit by the couple $(B; C)$ where B is the bank identifier and C the cache color. An application can have one or more assigned $(B; C)$ couples. Shared memory regions (between application on the same or different cores) use the $(B; C)$ couples of all applications sharing the region. Figure 4 depicts the execution flow of the allocation process. Our method ensures compatibility with all layouts depicted in Figure 3. It is also extensible to other future layouts. The safety net is explained in the next section of this paper.

When applications' private and shared memory have multiple allowed couples, the allocator select the $(B; C)$ couple to evenly distribute the workload on all available memory.

Once the algorithm selected the memory bank, the allocator provides one or more cache colors to the application. Each memory bank contains at least one cache color. We compute the number of cache colors per bank by subtracting the number of overlapping bits between the cache color bits and the bank color bits to the number of cache color bits.

To provide a faster allocation, we do not rely on linked list to represent free memory. We represent the memory banks with the *bank_color* structure.

```
struct bank_color {
    uint8_t  bank_id;
    uint32_t free_mem;
    ptr_t    cache_color_head[nb_cc];}
```

In this structure, the *cache_color_head* array represents the next free address in memory for a given cache color. We also use *nb_cc* to represent the number of cache color associated with the current bank. Using this structure, the allocation process is straightforward. When allocating a page to an application, the allocator sets the cache color head referred by $(B; C)$ to the next free page in $(B; C)$.

B. Bus bandwidth budgeting

We propose to extend the MemGuard [5] approach by adding the following features: (1) Per application budget allocation: each application receives a static amount of budget for the duration of its time window in the MAF; (2) An initial reclaiming [1] pool budget to accommodate with the scenario where all applications are critical; (3) Introduction of four application modes that we present later in this section;

1) Reclaiming modes: The computation of the initial reclaiming pool size as well as the budget allocation are out of the scope of this article and can be computed using the method proposed in [41].

To accommodate with the critical nature of ARINC-653 systems and render our approach more flexible, we propose to classify applications in four categories to extend the reclaiming feature:

- **Full reclaiming** mode, which allows an application to reclaim budget and provide budget to the reclaiming pool. Soft real-time partitions can use this mode. We also propose to further constrain the budget prediction to provide bounds on the budget removal of the application.
- **Greedy** mode for which an application does not provide any budget to the global budget pool but can reclaim budget from this same pool. This is suitable for any hard and soft real-time partitions, however, hard real-time applications are more prone to use this mode.
- **Altruist** mode, which allows an application to provide budget to the global pool but not reclaim budget from this pool. This mode is suitable for best-effort applications which can run in degraded mode.
- **Strict** mode, which forbids budget reclaiming and providing budget to the global pool.

[1]Reclaiming budget means that an application can request more budget than it was allocated by the means of a shared budget pool.

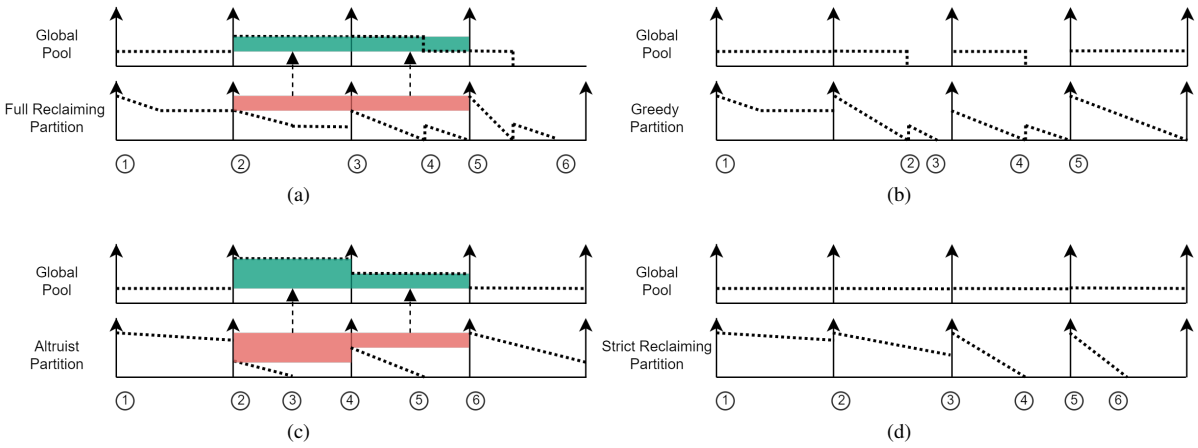


Fig. 5. Partitions' scheduling behavior when using full reclaiming mode (a), greedy mode (b), altruist mode (c) and strict reclaiming mode (d).

Figure 5 (a) depicts the execution of a partition under full reclaiming mode, which means that it can reclaim budget from the global pool and provide budget to it. At time ① the partition starts executing. At the end of the budgeting periods ② and ③, it was predicted that the partition will use less budget than initially provided. Budget is removed from the partition to provide it to the global pool. At ④ the partition exceeded its budget and reclaimed a fixed amount of free budget from the global pool. At ⑤, the does not remove budget from the partition. Finally, at ⑥ the partition exceeds its budget but the global pool is empty, the partition is removed from execution until the next timer's period.

Figure 5 (b) shows the execution of a partition under greedy mode. This time the partition's budget is exceeded at ②, ③ and ④ and is replenished at ② and ④ by reclaiming budget from the global pool and at ① and ⑤ from the bus budgeting replenish server. In greedy mode, no prediction is made and the partition does not provide any budget to the global pool.

Similarly Figure 5 (c) shows the execution of a partition under altruist mode. The budget can only be replenished by the bus budgeting server (at times ①, ②, ④ and ⑥). The partition can provide budget to the global pool and gets removed from execution when it exceeds its budget at times ③ and ⑤.

Finally, Figure 5 (d) depicts the execution for a partition using strict mode. Here, the partition does not give any budget to the global pool but also cannot reclaim any. The only replenish events occur at ①, ②, ③ and ⑤ and the partition is removed from execution at times ④ and ⑥.

The proposed reclaiming modes allow a complete flexibility on the system bus management. The system can change the mode at any moment and for any partition during the execution of the RTOS. Changing the reclaiming more is done through an API proposed by the RTOS. This API allows reconfiguring the budgeting mode while ensuring highly critical partitions are still correctly isolated. The API can be called at any moment by the partition during its execution.

Algorithm 1 shows the execution flow of the periodic server interrupt. This process is executed before restoring the application's context. Algorithm 2 depicts the process executed when an application exceeds its allocated budget. As proposed in MemGuard, when all cores become idle because of budget exceeding, the mechanism will reschedule all applications. Finally, we explain the $HealthCheck$ block in the next section and refers to the safety net feature proposed by the CAST-32A document.

Algorithm 1 Periodic server timer handler function.

```
1: function TIMERHANDLER
2:     App = ScheduledApp()
3:     nextBud = App.initBud
4:     if App is (full reclaiming or altruist) then
5:         nextBud = PredictNextBudget()
6:         AddSlackBudgetToPool(App.initBud - nextBud)
7:     end if
8:     if HealthCheck() is Failure then
9:         RaiseHMError()
10:        Return
11:    end if
12:    UpdatePMC(nextBud)
13:    UnlockCore()
14: end function
```

Algorithm 2 Interrupt handler executed when an application exceeds its budget.

```
1: function PMCHANDLER
2:     App = ScheduledApp()
3:     if HealthCheck() is Failure then
4:         RaiseHMError()
5:         Return
6:     end if
7:     if App is (full reclaiming or greedy) then
8:         nextBud = ReclaimBudget()
9:         UpdatePMC(nextBud)
10:        Schedule(App)
11:    end if
12:    if AllCoresIdle() then
13:        ForceResetTimer()
14:    else
15:        ScheduleIdle()
16:    end if
17: end function
```

In our context, we consider systems with hard real-time applications only. We cannot afford removing budget from a partition using s. Thus, we introduce an initial reclaiming pool containing free budget for any application. The global pool contains budget that the partitions can reclaim when

their budget is exceeded. This global pool is replenished at each budget server's tick. This approach allows improving the overall system's performance as presented in our result section. The benefits of this method are presented in the results section.

Both the periodic replenish server and the PMC interrupt handler are integrated from scratch in the RTOS. We have chosen to implement the different modules from scratch and not use a patched version of MemGuard because the architecture of an ARINC-653 compliant RTOS greatly differs from the Linux kernel. The periodic replenish servers consists in an interrupt service routine called every time the global timer triggers its periodic interrupt. In the same manner, when the PMC detect that the budget is exceeded, an interrupt handler is called and an idle task is scheduled to replace the running application as presented in Algorithm 2.

V. Safety Net

To ease the certification process, we define a set of rules that should always be verified during the system's execution. Apart from the CAST-32A guideline, we provide a way to ensure synchronization between cores regarding the MAF timings.

A. Core Verification

The RTOS architecture we use offers an Asymmetric Multi-Processing paradigm to manage the CPU. In this architecture, each CPU executes a separate instance of the RTOS. Small shared memory region enables the inter-core communications.

Each CPU has its own timer and uses a tick-less scheduler. This means no periodic tick is present in the system and timers interrupt occur only when needed (for instance, when the next application should be executed). To ensure synchronization, we add constraints to the MAF length. All CPUs can have different MAF length; however, we define a global hyper period based on the least common multiple Q between all CPUs MAF.

We define q_i the length of the MAF M_i defined for CPU i. M_i will be executed $\frac{Q}{q_i}$ times before reaching a synchronization point. When reaching such point, the CPU waits for the synchronization barrier release. When all CPUs reach the barrier, the kernel releases the CPU and immediately start a new MAF. This process corrects the drift between CPU clocks while ensuring real-time constraints are met on all cores. Our results show that the overhead introduced by the synchronization mechanism does not break any real-time constraint nor introduces deadline miss in the system. We provide the overhead considerations and experiments in Section 6.3. Q may be further constrained to reduce the drift between CPUs clocks. The smaller Q is, the faster the synchronization module correct the time drift.

B. Runtime Verification

To ensure safety of the proposed memory manager, we define a set of constraints that are verified at critical points during execution. Table II defines the set of variables we use to formalize the system. We further define the following rules:

TABLE II
List of variables associated with safety net constraints

Q	System's hyperperiod, defined as the least common multiple between all CPUs MAF
q_i	The length of the MAF M_i defined for CPU i.
A_i	Application i
$\mathcal{C}_{Ai}$	The memory colors set allocated to the application i.
A_i^C	The CPU identifier where A_i is mapped.
A_i^B	The bus bandwidth budget allocated to A_i.
A_i^M	The maximum number of bus access done by A_i during a single MAF.
$Acc(A_i, t)$	Accessed number done by an A_i at time t.
$\mathcal{S}_i^C$	The memory colors set allocated to the shared memory region i.
$\mathcal{S}_i^A$	Applications set sharing the memory region i.
$G(t)$	The free budget in the global pool at t.
$R(A_i, t)$	1 when A_i executes at time t, 0 otherwise.
B_{max}	The maximal bus bandwidth.

TABLE III
Formalized safety net rules

Rule 1	$A_i^C \neq A_j^C \Rightarrow \mathcal{C}_{A_i} \bigcap \mathcal{C}_{A_j} = \emptyset$
Rule 2	$\mathcal{S}_i^C \subseteq \bigcup_{\forall A_j \in \mathcal{S}_i^A} \mathcal{C}_{A_j}$
Rule 3	$\forall t \in [0; Q], G(t) + \sum_{\forall A_i} A_i^B \times R(A_i, t) \leq B_{max}$
Rule 4	$\forall A_i, \sum_{t=1}^{q_{A_i^C}} Acc(A_i, t) \leq A_i^M$

Rule 1: At any time during the system's execution, co-running applications must not share memory or cache color.

Rule 2: Shared memory regions color set must only contain colors allocated to the applications that share this region.

Rule 1 and rule 2 are verified during the system startup when allocating memory. It is further verified during a page fault, when the RTOS handles the page fault, it ensures that the address generating the fault is mapped to a physical address in the application's colors set. This rule does not apply to explicitly shared memory.

Rule 3: At any time t during execution, the sum of the bus bandwidth budget of the executing applications and the global budget pool must not exceed the maximal bandwidth permitted by the bus.

Rule 4: For safety purpose and fault detection, we define a maximum number of bus access for each application and IO in the system. The system should never reach this number to ensure the system's stability.

To verify rule 3, we add the *HealthCheck* function in Algorithm 1. Every time the system changes the budget configuration, the health monitor ensures that rule 3 is satisfied. If not, the health monitor handles the error. Rule 4 is verified in Algorithm 1 when handling applications budget overflow.

B_{max} can be provided by the processor's manufacturer or can be empirically measured with benchmarks. In this paper, we rely on the work presented in [43] to compute the value of B_{max} for our architecture.

VI. Results

In this section we present the study of our global approach and compare it to the current literature.

Our test bench comprises an NXP T2080RDB-PC board with four PowerPC e6500 cores running at 1.8GHz. We rely on a proprietary ARINC-653 RTOS (M-RTOS [44]) to

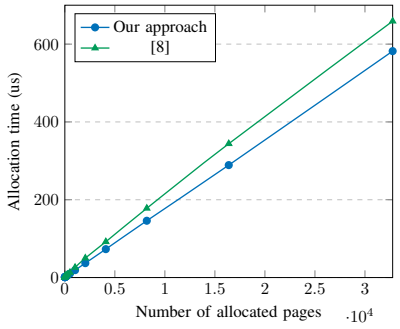


Fig. 6. Allocation time for different allocation sizes. The results show that our allocator allows faster allocation time.

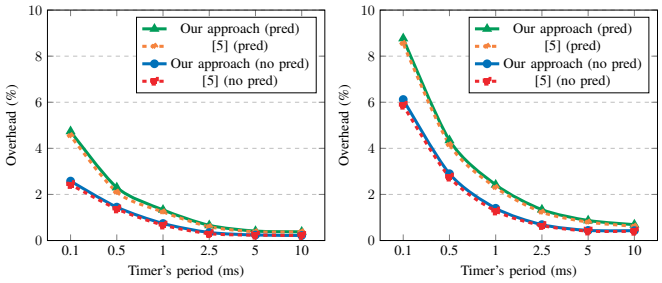


Fig. 7. Bandwidth limitation mechanism overhead depends on the timer's period.

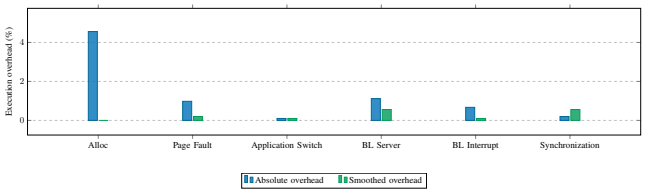


Fig. 8. Absolute overhead of the safety net compared to the smoothed overhead introduced at run-time.

retrieve our experiments results. We study the system using the TACLeBench [45] benchmark suite. We do not show results for the safety net other than its overhead. This is because the safety net is a part of the environment that is evaluated. All measures and results presented here are conducted with the safety net enabled.

A. Memory Allocation Overhead

We compare the execution time of our memory allocator for different allocation sizes to the one proposed in [8]. Our allocator initialization time is 2 us compared to 7120 us for the one proposed in [8]. This is because we do not have to create a linked list of available pages at initialization. Furthermore, we do not have to maintain this list during allocation. Figure 6 shows the allocation time reduction that benefits our method.

Memory allocation only occurs at the system's startup, which implies that the allocator's overhead only impacts the system's performance during boot time. In our test environment, memory allocation represents 9.39% (2.04ms) of the total startup time (21.74ms), which justifies the need to reduce its impact as much as possible. Our allocator reduces the startup time of the RTOS we use in our experiments by 24.8% (21.74ms) compared to the method proposed in [8] (28.91ms). After allocation, the cache partitioning technique has no overhead. Indeed, it relies on the MMU to perform a controlled translation of addresses. This process also occurs when not partitioning the cache.

B. Bandwidth Limitation Overhead

Our second experiment evaluates the overhead introduced by the bandwidth limitation module. We compare our implementation that adds reclaiming modes with the overhead introduced by MemGuard. Figure 7 present the overhead of our bus bandwidth management technique.

We based our test environment in the same context as what is proposed in [5]. To measure the overhead introduced by bandwidth limitation mechanisms, we use the following settings:

- For server's timer overhead, we disable bandwidth limitation mechanism by disabling PMCs interrupts. The rest of the mechanism is enabled. Figure 7 (a) shows our results.
- For server's timer overhead and budget exceeding mechanism, we enable bandwidth limitation mechanism by enabling PMCs interrupts. We also make sure that the application only exceeds its budget once every server's period expires. The RTOS directly reschedule the application once its budget is exceeded to measure only the limitation mechanism's overhead. Figure 7 (b) shows our results.

We compare our approach when using a predictor to add budget to the global budget pool and when not using any predictor. Results show that our extension of MemGuard to support per applications budgeting as well as reclaiming modes adds a small overhead (less than 0.5%). Our experiments show that the bus usage predictors introduce the most overhead.

Finally, we studied the impact that the mode change API has on the execution time of partitions. The mode change was implemented to only impact the calling core. Thus, this API call is no different than any system call made by the user to the RTOS (e.g. mutex acquisition, display print, etc.). The API performs a constant number of accesses to the memory. Based on the measurements of the execution time of common API and system calls in the used RTOS, changing mode is 4.6 times faster than a semaphore signaling, 8.5 times faster than getting the current partition's status and takes sensibly the same time as retrieving the current process's identifier.

C. Safety Net Overhead

We further study the impact of the safety net on the performance. We provide two metrics to express the overhead. We define the absolute overhead, which gives the absolute execution time added by the safety net. We also define the smoothed overhead, which describes the impact of the safety net on the applications execution time. It is important to note that the safety net overhead should be reduced as much as possible at its routines frequently executed in the system (e.g., TLB miss handlers, context switch, etc.).

Figure 8 presents the absolute and smoothed overhead of the safety net. *BL* stands for "Bandwidth Limitation". While we manage to maintain the overhead under 1% for most safety nets, the allocation safety net (Rule 1 and Rule 2) overhead reaches 4.56%. However, this overhead only occurs during the system startup and slows down the boot process by 1.5%. Therefore, we do not provide any allocation smoothed overhead metric, as it is nonexistent.

The time synchronization overhead presented in Figure 8 refers to the overhead added by the core MAF synchronization mechanism. Two factors cause this overhead. First, the synchronization routine itself must access shared data atomically to know the state of the synchronization barrier. The second factor comes from the potential synchronization of the cores. The more a core drifts, the more it is prone to reach the synchronization barrier late. In this test, the synchronization primitive causes the absolute overhead while the smoothed overhead accounts for the total overhead (synchronization primitive and drift of the cores clocks). The second factor is the more impacting, hence the bigger smoothed overhead. The results were gathered by measuring the number of cycles the cores stay in the synchronization routine at every MAF renewal and compared to the number of cycles the complete MAF renewal routine takes. The synchronization happens at most once every MAF start. Current timing analysis methods can take this overhead into account in the MAF renewal time. The overhead added by the synchronization was measured to be 0.56% of the MAF renewal time in the worst case. To gather the results, we measured 100,000 MAF renewal.

D. Scalability of the solution

The partitions-related safety net as well as the bus bandwidth limitation mechanism are designed to be independent for each core and are based on an AMP RTOS architecture. The cache coloring run time mechanism solely relies on the MMU and, thus, is independent of each core that has a private MMU. By construction, those mechanisms are not impacted by the number of cores in the system.

The startup memory allocation mechanism is impacted by the number of cores in the system. Memory allocation must keep global structures to ensure correct memory allocation between cores, thus synchronization primitives are used to ensure exclusive use of these structures. The worst case happens when all cores allocate memory at the same time. To access the shared data structures, a core might wait until all other cores have finished their allocation. Thus, the allocation time can be multiplied linearly by the number of cores in the system. To validate our assumption, we artificially created such a case by adding a barrier before each core allocates the partitions' memory. Each core has the same amount of memory to allocate. Our experimental measures show a maximal allocation time of 233μs on 1 core, 429μs on 2 cores, and 849μs on 4 cores. Finally, the core synchronization mechanism itself is not impacted by the number of cores. However, the time waited for all cores to reach the barrier may differ and is likely to increase with the number of cores. The equation $\sum_{i=1}^{N} D_i$ provides a way to compute the worst synchronization time where N is the number of cores and D_i is characterized by the amount of time the clock of the core i drifts between two synchronizations.

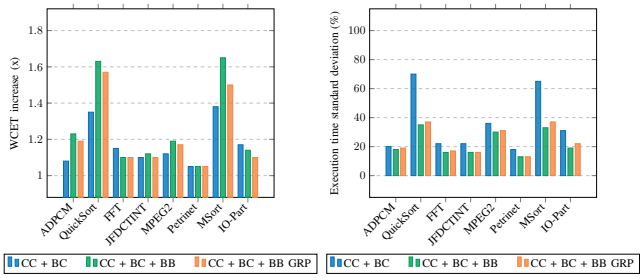


Fig. 9. WCET (a) and predictability improvements (b) provided by the memory manager. We compare the proposed metrics when using cache and bank coloring (CC + BC), bus budgeting (CC + BC + BB) and the initial reclaiming pool (CC + BC + BB GRP)

E. Effect on Predictability

We validate our approach by studying the predictability of eight concurrent applications. We define the predictability of an application based on the standard deviation of the execution time. We compare the predictability with interference management to the predictability without interference management. A lower value means the partitions' execution time is more predictable.

To approximate the measured WCET[2], we analyze each application under the same input data for all executions. We further executed each application multiple times until no higher execution time is detected after 1000 executions. In a safety-critical context, a more formal approach should be used, however, timing analysis in presence of interference in multicore system is not yet attainable and provides results that are too pessimistic [35].

This method provides a way to analyze the impact our work has on the applications' actual WCET. The WCET increase entailed by the interference mitigation mechanism is compared to the applications' WCET measured with a single-core setup, where only one application exclusively uses the system's resources. The lower the WCET increase, the better our method performs.

Finally, the bus budgets as well as the initial reclamation pool size are computed using the method proposed by [41] where an ILP formalism to the budget allocation that fits our experimental environment is defined.

Figure 9 (a) shows the relative increase of measured WCET compared to the WCET measured in single-core execution. Lower values are better and means the application suffered less slow down because of the interference mitigation means. We used 8 applications from the TacleBench suite and co-run them on 2 cores. Interferences are naturally introduced by the memory usage of each application. Table IV sump up our test

[2]We use a measured approximate WCET because current state of the art in multicore WCET analysis does not provide precise means of analysis in presence of interference. Our work is a step towards the use of conventional analysis methods, but other interferences exist and need to be addressed before being able to use regular analysis methods.

TABLE IV
TEST ENVIRONMENT APPLICATIONS

Application	Description	Core
ADPCM	Analog to digital filter Memory intensity: medium	0
QuickSort	Array sorting algorithm Memory intensity: high	0
FFR	Fast Fourrier Transform implementation Memory intensity: medium	0
JFDCTINT	Forward discrete cosine transform implementation Memory intensity: low	0
MEPG2	MPEG2 manipulation Memory intensity: medium	1
Petrinet	Petrinet simulation implementation Memory intensity: low	1
MSort	Array sorting algorithm Memory intensity: high	1
IO-Part	Application performing IOs on the serial port Memory intensity: high	1

payload. The period of each partition on the core 0 is 50 ms while periods for the core 1 are 75 ms. This ensures that with enough executions of the system, all partitions of the core 1 are co-run at least once with all partitions of the core 0. Our approach reduces the impact of isolation on the WCET while drastically increasing the predictability. In Figure 9 (b), the predictability corresponds to the standard deviation of execution time where 100% is the standard deviation in multi-core processors with the same setup but without interference mitigation. We show that in our test bench, we reduce the standard deviation of execution times by 68.1% on average.

Isolation is ensured by design. When using cache coloring, shared caches are strictly segregated into different areas and partitions using strictly different cache colors cannot share any cache area. It is the same for memory banks. Bus budgeting will also detect when a budget is exceeded and stop the partition at the very instruction where the budget is exceeded. Thus, isolation on cache and bus is ensured. However, in multicore systems, far more interference channels exist. For instance, cache coherence protocols will introduce delays when in need to update private caches [2]. Other hardware mechanisms such as the Miss Status Holding Registers (MSHR) will create interferences [46]. Thus, is it impossible to reach the same predictability in multi-core systems compared to single-core ones. Our objective is to remove the most critical one to lower the analysis pessimism and further conduct the system's analysis considering the remnant interferences.

It must be noted that the increase in predictability as well as the increase of the WCET is not the same for all applications. Indeed, in the system, some applications are more prone to interferences (usually applications that use more memory). Thus, memory-intensive applications (in our experiments MSort, QuickSort) will see their execution time increased more than other applications because of the resource restriction cache coloring and bus budgeting apply to them. For the same reason, the increase of the WCET is not proportional to the decrease of the standard deviation of execution time.

We also show that for highly critical system with only hard real-time applications, the initial reclaiming pool allows improving the overall system performance. Based on experiments with the proposed benchmark, we use an initial global reclaiming pool containing 10% of the maximal bus budget. This configuration yields to the best performance by providing additional budget for hard real-time applications using full-reclaiming and greedy modes while avoiding budget waste. Our study shows that while using an initial global reclaiming pool slightly reduces predictability, it allows to reduce the WCET and reduce the impact of bus budgeting.

Finally, it is important to discuss the tradeoff between WCET increase and predictability. An increase of less than 2 on dual-core architecture means that the system will be able to host more partition than a single-core system. Similarly, an increase of less than 4 on a 4-core architecture would mean the same. The goal of our interference mitigation means is to reduce the WCET impact as much as possible to ensure isolation but also allow more petitions to be hosted by the system. Based on our results, the method we propose increases applications' WCET by 22.3% on average on 2 cores, which is deemed acceptable to provide an increase of predictability of 68.1% on average. We also applied our approach on a 4-core setup, which yielded the same results, following the scalability analysis we provide in Section 6.5. For the sake of space, we do not present these results in this paper.

VII. CONCLUSION

Safety-critical systems rely on partitioned system to ensure complete isolation between applications. However, with the advent of multi-core processors, such isolation is impossible when using regular resource management methods. The contention on resources shared between the different cores generates interferences. Interferences are non-deterministic delays in execution time that prevent certification of safety-critical systems. The delay interferences introduce at run-time cannot be predicted. Thus, when measuring the Worst-Case Execution Time of applications, one must assume the worst case when sharing resources with other applications. This yields to overly pessimistic results that render the system unusable.

Research in the domain proposes multiple solutions to isolate different components and improve predictability. However, such solutions are not always applicable to safety-critical systems, where the configuration and implementation of interference mitigation methods must be deterministic and known during system design to allow certification.

In this paper, we presented a statically configurable memory manager that isolates the memory hierarchy between applications. Our approach enables to mitigate shared cache, DRAM and bus interferences with deterministic and certifiable methods. We further provided safety net rules to accommodate with the CAST-32A guidance document. Finally, we studied the impact of our method in terms of performance improvement and overhead. Our approach increases predictability by 68.1% while reducing the impact of mitigation methods.

As explained in Section 3, other interferences exist. Our approach restricted interference mitigation to shared caches, memory bank and bus interferences. Future work could integrate the use of channel partitioning in our mitigation framework and study its impact.

REFERENCES

[1] C. B. Watkins and R. Walter, "Transitioning from federated avionics architectures to integrated modular avionics," in *2007 IEEE/AIAA 26th Digital Avionics Systems Conference*, 2007, pp. 2.A.1–1–2.A.1–10.

[2] A. Löfwenmark and S. Nadjm-Tehrani, "Challenges in future avionic systems on multi-core platforms," in *2014 IEEE International Symposium on Software Reliability Engineering Workshops*, Nov 2014.

[3] L. Sha *et al.*, "Single core equivalent virtual machines for hard real—time computing on multicore processors," 2014.

[4] C. A. S. Team, "Cast-32, multi-core processors," USA, November 2016.

[5] H. Yun *et al.*, "Memory bandwidth management for efficient performance isolation in multi-core platforms," *IEEE Transactions on Computers*, vol. 65, no. 2, feb 2016.

[6] ARINC, "Arinc specification 653: Avionics application software standard interface," Maryland, USA, 2015.

[7] S. P. Muralidhara *et al.*, *Reducing Memory Interference in Multicore Systems via Application-Aware Memory Channel Partitioning*, 2011.

[8] N. Suzuki *et al.*, "Coordinated bank and cache coloring for temporal protection of memory accesses," in *Proceedings - 16th IEEE International Conference on Computational Science and Engineering, CSE 2013*, 2013.

[9] S. Mittal, "A survey of techniques for cache partitioning in multicore processors," *ACM Comput. Surv.*, vol. 50, no. 2, May 2017.

[10] M. K. Qureshi and Y. N. Patt, "Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches," in *2006 39th Annual IEEE/ACM International Symposium on Microarchitecture*, 2006.

[11] R. SC-205, "Do-178c, software considerations in airborne systems and equipment certification," 2011.

[12] G. Kedar *et al.*, "Space: Semi-partitioned cache for energy efficient, hard real-time systems," *IEEE Transactions on Computers*, vol. 66, no. 4, 2017.

[13] J. Brock *et al.*, "Optimal cache partition-sharing," in *2015 44th International Conference on Parallel Processing*, 2015.

[14] N. El-Sayed *et al.*, "Kpart: A hybrid cache partitioning-sharing technique for commodity multicores," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2018.

[15] G. Aupy *et al.*, "Co-scheduling amdahl applications on cache-partitioned systems," *The International Journal of High Performance Computing Applications*, vol. 32, 2017.

[16] G. Sun *et al.*, "Combining prefetch control and cache partitioning to improve multicore performance," in *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2019.

[17] J. Xiao *et al.*, "Cppf: a prefetch aware llc partitioning approach," 08 2019.

[18] M. Nejat *et al.*, "Coordinated management of processor configuration and cache partitioning to optimize energy under qos constraints," *ArXiv*, vol. abs/1911.05114, 2019.

[19] Z. Cui *et al.*, "A swap-based cache set index scheme to leverage both superpage and page coloring optimizations," in *Proceedings of the 51st Annual Design Automation Conference*, 2014.

[20] X. Jin *et al.*, "A simple cache partitioning approach in a virtualized environment," in *2009 IEEE International Symposium on Parallel and Distributed Processing with Applications*, 2009.

[21] M. Xu *et al.*, "Holistic multi-resource allocation for multicore real-time virtualization," in *Proceedings - Design Automation Conference*, jun 2019.

[22] M. Hassan, "Managing dram interference in mixed criticality embedded systems," in *2019 31st International Conference on Microelectronics (ICM)*, 2019, pp. 253–257.

[23] L. Liu *et al.*, "Going vertical in memory management: Handling multiplicity by multi-policy," in *Proceedings - International Symposium on Computer Architecture*. Institute of Electrical and Electronics Engineers Inc., 2014.

[24] M. Hassan *et al.*, "A framework for scheduling DRAM memory accesses for multi-core mixed-time critical systems," in *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS*, vol. 2015-May, may 2015.

[25] W. Ali and H. Yun, "RT-Gang: Real-time gang scheduling framework for safety-critical systems," in *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS*, vol. 2019, apr 2019.

[26] J. Fang *et al.*, "A memory scheduling strategy for eliminating memory access interference in heterogeneous system," *Journal of Supercomputing*, vol. 76, apr 2020.

[27] L. Liu *et al.*, "BPM/BPM+: Software-based dynamic memory partitioning mechanisms for mitigating dram bank-/channel-level interferences in multicore systems," in *Transactions on Architecture and Code Optimization*, vol. 11, no. 1, 2014.

[28] T. Ikeda and K. Kise, "Application aware DRAM bank partitioning in CMP," in *Proceedings of the International Conference on Parallel and Distributed Systems - ICPADS*. IEEE Computer Society, 2013.

[29] J. Zhou and J. Wang, "Archsampler: Architecture-aware memory sampling library for in-memory applications," in *2018 IEEE 36th International Conference on Computer Design (ICCD)*, 2018.

[30] M. A. Awan *et al.*, "Mixed-criticality scheduling with dynamic memory bandwidth regulation," in *Proceedings - 2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA 2018*, jan 2019.

[31] J. Kim *et al.*, "Reducing Memory Interference Latency of Safety-Critical Applications via Memory Request Throttling and Linux Cgroup," in *International System on Chip Conference*, vol. 2018-September. IEEE Computer Society, jan 2019.

[32] H. Yun *et al.*, "BWLOCK: A dynamic memory access control framework for soft real-time applications on multicore platforms," *IEEE Transactions on Computers*, vol. 66, no. 7, jul 2017.

[33] A. Agrawal *et al.*, "Contention-Aware Dynamic Memory Bandwidth Isolation with Predictability in COTS Multicores: An Avionics Case Study," in *29th Euromicro Conference on Real-Time Systems (ECRTS 2017)*, 2017.

[34] G. Durrieu *et al.*, "Predictable Flight Management System Implementation on a Multicore Processor," in *Embedded Real Time Software (ERTS'14)*, TOULOUSE, France, Feb. 2014.

[35] S. Park *et al.*, "Execution model to reduce the interference of shared memory in arinc 653 compliant multicore rtos," *Applied Sciences*, vol. 10, 04 2020.

[36] M. Hassan and H. Patel, "Criticality- and Requirement-Aware Bus Arbitration for Multi-Core Mixed Criticality Systems," in *2016 IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2016 - Proceedings*, apr 2016.

[37] A. Kostrzewa *et al.*, "Safe and dynamic traffic rate control for networks-on-chips," in *2016 Tenth IEEE/ACM International Symposium on Networks-on-Chip (NOCS)*, 2016.

[38] A. Dabaghi and H. Farbeh, "High performance and predictable memory controller for multicore mixed-criticality real-time systems," *IET Computers & Digital Techniques*, jun 2019.

[39] J. Park *et al.*, "HyPart: A hybrid technique for practical memory bandwidth partitioning on commodity servers," in *Parallel Architectures and Compilation Techniques - Conference Proceedings, PACT*, nov 2018, pp. 1–14.

[40] Y. Xiang *et al.*, "EMBA: Efficient memory bandwidth allocation to improve performance on intel commodity processor," in *ACM International Conference Proceeding Series*. Association for Computing Machinery, aug 2019.

[41] M. A. Awan *et al.*, "Uneven memory regulation for scheduling IMA applications on multi-core platforms," *Real-Time Systems*, vol. 55, apr 2019.

[42] C. A. S. Team, "Cast-20, addressing cache in airborne systems and equipment," USA, July 2003.

[43] J. D. McCalpin, "Stream: Sustainable memory bandwidth in high performance computers," University of Virginia, Tech. Rep., 1991-2007.

[44] M. S. . S. I. MSS, "M-rtos," 2020. [Online]. Available: https://www.mss.ca/m-rtos/

[45] H. Falk *et al.*, "TACLeBench: A benchmark collection to support worst-case execution time research," in *16th International Workshop on Worst-Case Execution Time Analysis (WCET 2016)*, 2016.

[46] P. K. Valsan *et al.*, "Taming non-blocking caches to improve isolation in multicore real-time systems," in *2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2016.

Whole-System Analysis for Memory Protection and Management

Felix Bräunling[1], Simon Wegener[2], Daniel Kästner[2], and Isabella Stilkerich[3]*

***Keywords*—** Memory Management, Safety, Genericity

Abstract

The automotive industry is a market that is heavily cost driven. Thus, software development without the reuse of software components is no longer economically feasible due to the increase in complexity of vehicle functionality, software, and hardware. To facilitate this reuse, software development—especially for control systems—is done on a functional level using domain-specific languages (DSLs) and model-based software development (MBSD). DSLs and models are later on translated into a general-purpose language such as C or C++. While such software components are likely to meet functional requirements, they need to be tailored to the specific application being targeted and the quality requirements posed by that application. One aspect that needs to be taken into consideration during this tailoring is the management of memory and its direct impact on quality requirements such as functional safety, security and performance. The proper management of memory is a non-trivial task and has to embrace not only the target hardware but also the infrastructure software used, e.g., the operating system.

In this paper we present cAMP (**c**onstructive and **A**daptive **M**anagement **P**artitioning), a tool to influence model-based code generation to respect functional safety and performance requirements by performing trait-based data classification. The trait-based data classification is based on the output of concurrency-aware sound static code analysis performed by Astrée.

1 Introduction

Model-based software development and the use of domain-specific languages enable control engineers to develop application software, e.g., traction or engine control systems, in a language familiar to their domain. At the same time, models and DSLs abstract away the underlying programming language and hardware model, therefore no deep knowledge about these parts is required by the control engineer. Control engineers are often domain experts on control theory, but not software engineers. As an additional advantage, MBSD focuses by design on a pure functional development of systems, which eases reusability in projects with similar functional requirements. Models and programs written in DSLs later on are translated into a lower-level language such as C/C++, or are directly compiled into machine code. An integration engineer then takes the generated artifact and integrates it into a bigger software system, along with other application software, basic software, and infrastructure software such as an operating system. A successful integration requires the fulfillment of quality requirements, especially for safety-critical embedded systems:

- **Functional Safety:** The system shall operate correctly in reaction to its inputs such that the environment is not harmed by the system. This is part of the system's functional suitability.
- **Performance / Resource Efficiency:** To save costs, execution time and memory usage of the deployed system should be economic.
- **Adaptability and Reusability:** The integrated software shall be downward adaptable, meaning it can be adapted to various microcontrollers to use their resources in an efficient manner. It also shall be upward adaptable to reflect application-specific, that is, project-specific requirements. Software components shall be reusable in various project instances (platform approach).

These requirements are not indifferent to each other, but are interwoven by complementary or competitive relations. Oftentimes quality requirements are also referred to as non-functional requirements. The ones mentioned above and more are part of the quality model described in ISO 25010 [19].

1.1 Problem Statement

In this paper, we will describe how the quality requirements mentioned above are respected in our approach. As an example, we consider memory management, which is a crucial aspect that needs to be considered during integration. Proper memory management has a big impact on achieving *resource efficiency*, by reducing access times, using the restricted amount of available memory efficiently, and exploiting the intertradeabilty of memory and computation time. Memory management is also supporting *functional safety* by limiting processes to their own memory regions, isolating these memory regions with memory protection mechanisms, and creating deterministic memory access times to support real-time system development.

On the other hand, MBSD and the goal of writing software generically for high *reusability* does not consider memory management at all, due to its tight coupling with the specific target hardware. While this lack of memory management awareness can be mitigated manually during the integration, this

*[1]Felix Bräunling is with Method Park by UL, Germany felix.braeunling@fau.de
[2]Simon Wegener and Daniel Kästner are with AbsInt Angewandte Informatik, Germany {swegener, kaestner}@absint.com
[3]Isabella Stilkerich is with Schaeffler Technologies AG, Germany isabella.stilkerich@schaeffler.com

approach does not scale with platform-based or product-line-based development approaches. The manual approach is also prone to errors and reduces reproducibility of the achieved results. The problem space grew even larger with the introduction of multi-core microcontrollers and heterogeneous, distributed memories-on-chip The question arises how reusable or generic software, developed using DSLs and MBSD, can be tailored during the integration to fit application-specific requirements with regards to memory management in a manner that is scalable and reproducible.

1.2 General Approach

The authors argue that, by analysis of the whole system using sound static analysis, and combining this information with system-architecture information, a memory management solution can be automatically derived.

To achieve this, sound static analysis has to be applied on the code of the software instantiated for a particular project. In order to derive information on data and memory-usage behavior, the software has to be free of type and memory defects. While languages such as Rust, Java, and Ada exhibit type and memory safety by design, C and C++ which are predominant in the automotive industry are not type and memory safe. By employing sound static analysis these properties can be retrofitted to C and C++ by removing detected type and memory defects in application software.

Type- and memory-safe software can then be analyzed using concurrency-aware whole-system analysis [28] to gather memory-access information. Combining this memory-access information with the application-specific system architecture, common *data traits* can be identified. Based on these data traits, each individual data item can be classified and a memory mapping can be derived for common classes of data.

1.3 Contribution

As we demonstrated in [13], a memory management tool can be integrated in a workflow that enables the tailoring of generic software systems to product specific application in an automated fashion. This paper focuses on the details of such an automated memory management tool and its tailoring towards application-specific quality requirements. We develop a system model of the problem statement and present the data traits that are derived from said model. We show the possibility of retrofitting C and C++ with type and memory safety using sound static analysis. cAMP, implementing a data-trait-based data classification algorithm, is presented and its capability to derive a memory mapping is demonstrated using an example application.

2 Aspects of Memory Management

Memory Management concerns itself with the allocation of memory, the access to that memory and the freeing (deallocation) after the use of that memory. In safety-critical embedded systems, supplying vehicle control functions, the memory is usually linearly mapped. Functions can optionally be protected by a memory-protection unit (MPU), if isolation is needed and the microcontroller is equipped with an MPU.

Table 1: High-level qualitative comparison of memory types [43, 32, 40]

Type	Read Time	Write Time	Volatile
SRAM	Very Low	Very Low	Yes
DRAM	Low	Low	Yes
EEPROM	High	Very High	No
Flash	Low	High	No

Often, heap-based memory management is prohibited, due to problems such as indeterministic (de-)allocation times, unpredictability of run-time memory requirements and general implementation bugs (e.g., use after free, double free, memory leaks).

The amount of available memory is usually restricted, due to the increased cost per unit of added memory. Therefore, it is necessary to provide an economic management strategy for the deployed application. This strategy shall include information about memory technologies (Section 2.1) and shall respect quality requirements (2.2).

2.1 Memory Technologies

Predominant technologies used are static RAM (SRAM), dynamic RAM (DRAM), flash memory, or EEPROMs. Each of these technologies offer different characteristics in terms of read t_R and write t_W time, volatility, and cost per unit of memory. Table 1 gives a qualitative overview of the different types of memories and their properties. The cost per unit decreases for each line in the table for the same volatility.

In addition to the chosen memory technology, the connection between a processing unit C and a memory M influences the total access time t_A. Modern microcontroller such as Infineon's AURIX TC2xx/TC3xx series [18] (see Figure 1), STM's SPC5 series [36] or the ARM Cortex-R52 [5] family exhibit similar patterns in connecting memories with processing units:

- **Tightly-coupled**: Memory is integrated within the processing unit.
- **Direct**: Memory and processing unit are connected via an exclusive direct connection.
- **Bus with arbitration**: Memory and processing unit are connected via a shared bus that uses arbitration to manage concurrent usage.
- **Interface with arbitration**: Memory access is performed through an interface, which manages concurrent access and offers additional functionality such as error detection or even error correction.

Tightly-coupled or direct connections only incur a deterministic cost t_D in addition to the technologies read or write time. DRAM incurs an additional time cost every few accesses, due to memory cell refresh cycles. In this paper this effect is not taken into consideration. Access over a bus and/or an interface add a similar deterministic cost t_D but also an arbitration-dependent cost t_{Arb}. This cost is depending on the amount of concurrent traffic on the bus/memory at runtime and the arbitration technique used and can introduce jitter when accessing

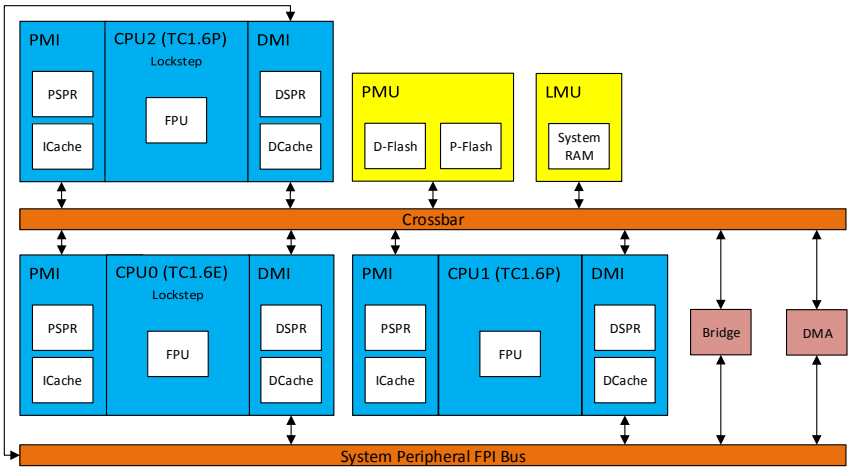


Figure 1: Hardware Diagram of the AURIX TC277

data. The total access time depending on the these factors is therefore:

$$t_A = t_{Arb} + t_D + \begin{cases} t_R \text{ for read access} \\ t_W \text{ for write access} \end{cases} \quad (1)$$

The use of caches and memory hierarchies changes this behavior as it improves the access time, if the data is present in the cache, but adds a penalty if it is not, leading to varying access times. In this paper, caching and its effects are not considered.

2.2 Quality Requirements and Memory Management

By allocating data to memories while respecting their technology and connection, the memory management can make efficient use of the available resources and support the quality requirements posed by the application. The following section will present how selected quality requirements are linked to the choice of data allocation.

2.2.1 Functional Safety: Real-Time and Isolation Requirements

In safety-critical domains, such as aerospace, medical and automotive, systems have to react in a timely manner to events. Often, *hard real-time* requirements are posed towards these systems and *time-deterministic* execution of the system's functions is therefore necessary. As Equation 1 shows non-determinism in the access time t_A stems from data access over arbitrated bus systems and memory interfaces. By allocating data to memories which exhibit deterministic, jitter-free access times, such as tightly-coupled SRAM, the allocation choices can support the requirements of hard real-time behavior and functional safety. Also, the availability and overall quality of timely functions can benefit from this approach.

In addition, functional safety requirements can add the need for fault-tolerant memory behavior. This can either be reflected in memory-technology properties such as read/write cycles in a memory lifetime and / or the resistance to environmental effects. Often, specific memories in a system offer additional measures such as error detection and error correction capabilities. By allocating data which is used in critical functions to these memories, the fault-tolerance of the system can be increased.

To achieve freedom from interference in space between different functions realized by one system, it is necessary to restrict the access to data allocated to the respected functions. This is achieved by a proper configuration of the MPU that respects proper access modes (the combination of read, write and execute authorizations) for each of the processes[1] deployed on the system and the data these processes operate on. Configuration errors which restrict the access right too much, prohibiting legal access to data, have a negative impact on the safety and security of the application. The inverse is true as well, as too permissive access authorizations enable rogue data accesses. It is thus paramount that memory protection regions configured in the MPU reflect the memory access rights intended by the application.

For a static system the MPU configuration defines how a process P running on a processing unit C can access a set of data $D = \{d_1, ..., d_n\}$. Each data set D is associated with a memory-protection region $R(D) \rightarrow A$ that prescribes the allowed access mode A, which is a combination of read r, write w and execute x modes. To differentiate between different access rights, processes can be linked to protection regions.

2.2.2 Performance Efficiency

One common trade-off in memory management is that of *computational space-time*, where memory can be exchanged for run time and vice versa in certain applications (such as using precomputed lookup tables instead of performing time consuming computations during run-time).

On the other hand run time cost is directly influenced by the choice of allocation. As the processing unit operates on data that is read from or written to memory the run time cost of the execution is dependent on the speed at which these operations are performed. In an optimal case, all necessary data must only

[1] A process is a set of threads of control executing in a common address space.

be read once and is then kept in CPU registers. But as CPU register space is limited this is not feasible. Therefore, in order to reduce run time cost caused by access times given by Equation 1, the average access time must be kept minimal:

$$min\left(\frac{\sum_{i=1}^{n} f * t_{A_i}}{n}\right) \text{ for } n = \text{Amount of Data} \quad (2)$$

Where f describes how often access to a data item is performed and t_{A_i} describes the access time for a data item. This can be achieved by allocating as much data as possible in fast-access memories such as SRAM and DRAM. As these types of memory are also often the smallest on the device and cannot store all data, priority can be given to data which is accessed often.

Avoiding concurrent access that leads to arbitration can also increase performance efficiency by reducing t_{Arb}. This can be achieved by allocating data, that is not shared between different processing units, on different memories. This reduces synchronized access by different processes to the same memory. For example on the SPC584 [36] each core has a tightly-coupled data memory that can be preferred for data not shared between the two cores to avoid access conflicts which lead to arbitration. The reduction of access time jitter caused by arbitration is also beneficial for the realization of the functional safety requirements.

3 Ensuring Well-defined Memory Access Behaviour

Our goal is the automatic derivation of a mapping of the various data objects to the memories such that the integrated software system fulfills all quality requirements (functional safety, performance, resource efficiency, adaptability, etc). We apply static whole-system analysis to derive a traits-based data classification (Section 4), paving the way for the memory mapping later on (Section 5).

However, any erroneous memory access behavior would falsify the traits-based data classification. Hence, we need to ensure that the integrated software system has a well-defined memory access behavior. In other words, we need to ensure type and memory safety of the program, since this is a necessary precondition that the semantics of a program exhibits a well-defined data access behavior. Therefore, achieving type and memory safety for a program is a crucial requirement for the automated memory-management approach presented in this paper. However, this does not exclude programming errors on the logical level. Other types of analyses and testing need to be performed to avoid these.

3.1 Memory and Type Safety in C and C++

Aiken et al. [3] defines that type safety is present when operations are only performed on types for which they are defined. Memory safety is present when only existing objects are accessed and access only happens within the object's boundaries. Type-safe applications are also memory-safe.

Languages such as Rust, Java, or Ada provide type and memory safety by design, usually enforced by their strong type system. But for various reasons such as performance, availability of programmers, availability of compilers for the intended target hardware, or the lack of a language standard, these languages are not commonly used in embedded automotive systems. C and C++ are the predominant programming languages used in embedded automotive control applications. Neither of them guarantees type and memory safety by design [35].

3.1.1 Dialects and Libraries

One approach to remedy the lack of type and memory safety are C/C++ dialects or libraries, that enhance the type system, such as Cyclon [21], CCured [33], Checked-C [38] for C or the Guideline Support Library [27] for C++. Problems with this approach can be the lack of compilers for the target hardware and the missing interoperability with legacy code. Some of them also introduce additional run-time checks, that entail a performance cost as a trade-off for the increased type and memory safety.

3.1.2 Retrofitting Type and Memory Safety Using Sound Static Analysis

While all of these problems can be remedied, the authors argue that—instead of rewriting extensive, existing code bases to use dialects or external libraries—sound static analysis based on abstract interpretation [15] can be employed to achieve type and memory safety. In C and C++ type and memory defects are exhibited when undefined or unspecified behavior is evoked. An example for a violation of type safety is the overflowing of integer types or comparing incompatible pointer values. An example for a violation of memory safety in C and C++ would be accessing an array outside its bounds or dereferencing a null pointer. To avoid these type of defects, programs need to be limited to a well defined subset of the language which is type and memory safe. Standards such as MISRA C [30] and MISRA C++ [31], CERT-C [14], or the C++ Core Guidelines [37] define such subsets. Static analysis can be employed to enforce these subsets and detect type and memory defects, allowing developers to remove those defects.

Sound static analysis based on abstract interpretation guarantees to detect all defects present in a program, for the types of defects the analysis targets. For our approach a sound system-level analysis is needed which determines the data flow between different threads (called processes in the terminology of Astrée) that may be distributed over several cores. Such an analysis requires a concurrent semantics providing a scalable sound abstraction that covers all possible thread interleavings. These requirements are met by Astrée [28, 25, 23], a sound static analyzer employing abstract interpretation for C and C++ Code. Astrée's main purpose is to report program defects caused by unspecified and undefined behaviors in C/C++ programs. Table 2 compares the type and memory safety definition given by [3] with a subset of defects that are detected by Astrée. A particularity of Astrée is its memory domain which supports an exact analysis of pointer arithmetic and union manipulations and provides a type-safe analysis of absolute memory addresses.

If all defects found by the analysis are removed from the program, the resulting program exhibits well-defined behavior and

Table 2: Mapping of Alarm Types [2] to Requirements of Type- and Memory Safety [3].

Requirement	Alarm Type
Only operations applied for instances of the type	Invalid pointer comparison
	Subtraction of invalid pointers
	Attempt to write to a constant
	Dereference of mis-aligned pointer
	Overflow of Integers or Float
	Invalid shift argument
	Use of uninitialized variables
	Division or modulo by zero
	Undefined integer modulo
	Invalid function calls
Access only existing objects	Dereference of null or invalid pointer
	Pointer to invalid or null function
	Use of dangling pointer
	Arithmetics on invalid pointers
	Possible overflow upon dereference
Access only inside object boundaries	Incorrect field dereference
	Out-of-bound array access
	Dereference of mis-aligned pointer
	Possible overflow upon dereference

is free of type and memory defects. The advantage of this approach is its applicability to legacy code. By using an iterative, per-use-based approach, legacy code can be transformed into a state of type and memory safety economically. As no additional annotations, libraries or specific compilers are necessary other parts of the tool chain are not limited or in need of being exchanged. This approach is also in line with standard recommendations, such as those posed by ISO 26262-6 [20], which recommend the use of static analysis to detect programming defects and language subsets to avoid programming defects.

The approach presented in this paper is aimed at achieving correct memory mapping at a source code level. While compiling and linking the program errors can be introduced into the resulting binary. This paper does not consider errors during compiling and linking the program.

3.2 Handling of Stack Objects

Local variables of C/C++ functions are stored on the stack, which is dynamically managed at run-time. This is usually transparent to the program and based on the translation of the program by the compiler and the targeted hardware. These stack objects need to be allocated in memory as well, and the same requirements for memory safety need to be applied to them. In safety-critical systems it is usually the case that the memory used for the stack is statically allocated before run-time with an upper limit to its size. If this upper bound is wrongly chosen a read or write out-of-bounds can occur, violating memory safety.

To determine safe upper bounds sound static analysis can be employed: in this case, soundness means, that the worst-case stack usage might be overestimated but never *under*estimated. For safe stack usage analysis, it is important to work on fully linked binary code, since the effects of code generation—inserting padding bytes, register allocation, etc.—or link-time optimizations have to be taken into account. Hence the static stack usage analysis cannot be based on the source code but must work on the executable machine code. It approximates the semantics of the machine code of the microprocessor by using an abstract model of the processor architecture. The abstract model does not need to cover the entire state of the microprocessor, only the parts affecting the stack space are needed. The hardware state relevant for worst-case stack analysis includes the processor registers and the memory cells. For a naïve analysis, only the stack pointer register is needed, but for precise results it is important to perform an elaborate value analysis on the contents of processor registers and memory cells.

The tool StackAnalyzer [22, 1] is one example of a sound static stack usage analyzer. By concentrating on the value of the stack pointer during value analysis, StackAnalyzer computes how the stack increases and decreases along the various control-flow paths. This information can be used to derive the maximum stack usage of the entire task. StackAnalyzer takes the entire application into account and interprocedurally analyzes each call site with its precise stack height. The results of StackAnalyzer are presented as annotations in a combined call graph and control-flow graph. It shows the critical call stack, i.e., the path on which the maximum stack usage is reached which gives important feedback for optimizing the stack usage of the application under analysis.

4 Traits-based Data Classification

To reason which data needs to be allocated to which memory M and be contained in which protection region $R(D)$, the data and its properties need to be known. In this section whole-system analysis using concurrency-aware static analysis is presented as a means to identify data properties. Afterwards it is shown how data can be grouped into data classes using associated data traits.

4.1 Identifying Data Traits by Whole-System Analysis

One example tool providing the necessary information about the data access behavior is Astrée, which leverages its sound abstraction of thread interleavings (cf. Section 3.1.2) for sound global data and control flow analysis.

The information about the execution model, in particular threads, their priorities, their core assignment etc. cannot be extracted from the C/C++ source code. Astrée provides intrinsics to support a manual specification of the execution model. In addition, Astrée supports a fully automatic analysis of concurrent software projects which comply to either the ARINC 653 standard, or the OSEK/AUTOSAR standards [34, 7]. For those OS norms Astrée provides stub libraries which implement an abstract model of the OS API. A particularity of OSEK/AUTOSAR is that system resources, including tasks, mutexes, or spin-locks are not created dynamically at program startup; instead they are hard-coded in the system: a specific tool reads a configuration file in OIL (OSEK Implementation Language) or ARXML (AutosaR XML) format describing these resources and generates a dedicated version of the system to be linked against the application. Astrée supports a similar work-flow [29]. In the preprocessor stage it can read

OIL/ARXML files and outputs a C file containing a table of the declared resources, with their attributes (mapping of tasks to applications and cores, task priority, alarm periodicity, etc.) which are fully taken into account during the analysis.

Astrée generates sound data and control flow reports (see Figure 3). Astrée tracks accesses to global variables, static variables, and local variables in case those accesses are made outside of the frame in which the local variables are defined (e.g., because their address is passed into a called function). All data and function pointers are automatically resolved. The soundness of the analysis ensures that all potential targets of data and function pointers are taken into account. Astrée's data and control flow reports show the number of read/write accesses for every global, static, and out-of-frame local variable, lists the location of each access and shows the function from which the access is made. All variables are classified as being globally-accessible, effectively shared between different threads, or subject to a data race. The control flow is described by listing all callers and callees for every function along with their respective call sites and the threads in which they can run. In AUTOSAR projects, additionally the application and the core to which the executing thread belongs is listed (cf. Section 5.1).

More sophisticated information about selected flows of values can be provided by a user-configurable taint analysis engine and a built-in program slicer. They enable Astrée to also compute a safe approximation of the data and control coupling between software components [24].

4.2 Data Traits and Classes

The way data is accessed can be described by traits. The traits we identified can be grouped into five groups:

- Mode of access: Data is either only read r, only written w, read and written rw or exectued x.
- "Sharedness": Data is either accessed by only one process $P = P_i$ (process-local) or by a set of processes $P = \{P_0, ...\}$ (process-global).
- Origin of access: Data is either accessed from a single processing unit $C = C_i$ (core-local) or by a set of processing units $C = \{C_0, ...\}$ (core-global)
- Special uses: Certain data is accessed for a specific purpose and behaves different than "ordinary" data. Examples for this kind of data are symbols representing memory mapped I/O as well as calibration or measurement data that can be accessed by specialized tools during development and production. Stack objects, representing the local data stack of a threads are also tagged as special uses. The symbols representing special uses are provided by the system architecture specification.
- Dead data: Data that is never accessed or only written to, but still declared in the program. However, this trait is only derived if the data is not already marked as a special uses data.

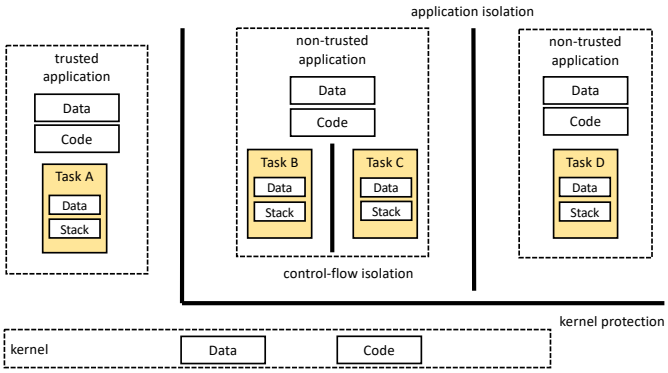


Figure 2: Isolation Boundaries in AUTOSAR [35]

Data classes, describing common access behavior, can be derived by combining the above traits. Examples for these data classes can be found in Section 5.3.

5 Memory Mapping

This section presents cAMP as an implementation of traits-based memory management using the output of a whole-system analysis as described in 4.1 and information about the system architecture. Section 5.1 presents the environment for which cAMP was developed. Section 5.2 to 5.4 describe the phases of cAMP's operation that derive the mapping. Section 5.5 presents limitations of this implementation and how these can be remedied through future work. Finally, Section 5.6 shows how example mappings have been derived for an academic control application.

5.1 Environment and Goals

cAMP targets automotive control applications that are developed using a model-based development approach and C as the intermediate programming language bridging MBSD and the target hardware. These applications are deployed alongside an implementation of the AUTOSAR operating system on an AURIX TriCore TC2xx series microcontroller. The choice of a particular microcontroller from the series depends on the performance needs of the deployed functionality.

In AUTOSAR, schedulable units (*Runnables*) are grouped into threads (*OsTasks*) that are executed according to one or more static schedules. Each *OsTask* is part of a single *OsApplication* and has its own stack object. An *OsApplication* is a collection of tasks, interrupts, alarms, data, etc. that form a functional unit in AUTOSAR with its own address space. *OsApplications* are in turn allocated to exactly one execution unit (*EcuC*). Protection regions $R(D)$ are linked to *OsApplications* which we will take as a close approximation to a process P. Protection regions can isolate one *OsApplication* from another, and even *OsTasks* executing within an *OsApplication* can be isolated from each other (indicated by the thick black lines in Figure 2). [9].

The goal of the mapping is foremost to be automated, correct and repeatable. Correct in this case means all data is bound (if possible with the available memory space) and no isolation

or safety requirements are violated. As a secondary goal the mapping intends to increase deterministic execution and performance efficiency.

5.2 Information Aggregation

To perform the mapping, cAMP gathers information about:

- **Deployment of AUTOSAR tasks**: AUTOSAR provides a system specification, that contains the mapping of *OsApplications* to *EcuCs* and *OsTasks* to *OsApplications*. It also includes information about the real-time system, such as the periodicity p_i of *OsTasks*. This specification is serialized to ARXML, which is a machine-readable format of the system specification. cAMP extracts the relevant portions from the ARXML file to identify the mapping of processes to processing units.
- **Memory Layout**: This is currently passed as a CSV file describing the boundary addresses of each hardware memory.
- **Data access behavior**: Astrée provides a report for its whole-system analysis describing the flow of data (see Figure 3). This provides information about which data is accessed by which process in which mode and from which core. Data and *OsTasks* are identified by their symbols. Astrée also produces a similar report that maps processes to functions. StackAnalyzer provides safe upper bounds for the stack objects of each *OsTask*.
- **Data classes and their traits**: cAMP uses data classes of the form described in Section 4.2. These are currently hard-coded in the implementation.
- **Mapping rule set**: The mapping rule set describes which data classes are mapped to which memories. In the current implementation this mapping rule set is a hard-coded state machine. The details of this rule set are described in Section 5.4.
- **Additional Requirements**: To identify calibration and measurement data classes cAMP reads A2L files emitted by *TargetLink* during code generation [16], which is a specification format for the CCP and XCP measurement and calibration protocols. These A2L files identify special use data by their symbols. Alternatively the AUTOSAR RTE specification and its measurement and calibration support could be leveraged [10].
- **Safety Constraints**: To respect specific binding requirements, cAMP provides a simple constraint language that enables expressions of the form $P_i(Data) \rightarrow M_i$ that enforces a mapping of data belonging to P_i to the memory M_i. This is useful if data must be mapped to specific memories due to the presence of safety features, such as error correction codes. Additionally, constraints of the form $M_i(P_i(Data)) \neq M_k(P_k(Data))$ can be formulated to enforce hardware isolation between process not only enforced by the MPU, but also by allocating to different physical memories. One possible use case are monitoring functions that must not share the same physical memory as the monitored functions.
- **Data representation**: From the model-based development tools (such as Matlab/Simulink in conjunction with TargetLink) information about data types of the data is extracted.

This data is automatically read in via adapters to the file formats (ARXML, CSV, A2L) and then used to perform the classification of the identified data.

5.3 Classification

In this phase each data item is mapped to an "owning" process and assigned a data class. Each data item starts with the classification "No Class". The data classes assigned by cAMP are:

- **Core-(i)-local, process-(k)-local**: Data only accessed by a process P_k deployed to processing unit C_i.
- **Core-(i)-local, process-(k, ...)-global**: Data that is accessed by a set of processes $P = \{P_k, ...\}$ that are all deployed to C_i.
- **Core-(i, ...)-global, process-(k, ...)-global**: Data that is accessed by processes $P = \{P_k, ...\}$ that are deployed to various processing units $C = \{C_i, ...\}$.
- **Constant**: Data that is only read and never written.
- **Calibration**: Data that is marked in A2L files as `CALIBRATION`.
- **Measurement**: Data that is marked in A2L files as `MEASUREMENT`.
- **Optimized Out**: Data that is only written, but never read. This is currently not used further, as cAMP is not capable of identifying symbols used for memory mapped I/O.

Orthogonal data is assigned an access mode $A \in \{r, w, rw, x\}$.

For technical reasons the mapping of a data item to an accessing core is performed by linking the *OsApplication* P via the *OsTask* T that accesses the data.

The concept of an "owning" process is introduced to identify a predominant core for data that is shared across processes and processing units. In its current implementation this "owning" process is the process with the most write accesses performed on the data. This is a pre-optimization for performance, as Table 1 shows that writes are usually more expensive and the "owning" process ensures that the core linked to that process is looked at first, when searching for a suitable memory.

The mapping is performed by a decision tree, fed by the information aggregated in the first phase, described in Section 5.2.

After the classification phase all data has an assigned data class which is used in the mapping phase to produce a mapping of data to memories and protection regions.

As a last step each data's access frequency f_{Di} is determined for reads, writes and calls. This is done by computing a hyperperiod H for all *OsTasks* T_i and their periods $p(T_i)$. This

Variable	Function	Access	Process	Data races	Shared variable	Location
about_x_axis__DiscreteTimeIntegrator	Sa2_controller_about__X_axis	read	Process 3: T1_Controllers	no	no	AttitudeController.c:583.42-78
about_x_axis__DiscreteTimeIntegrator	Sa2_controller_about__X_axis	read	Process 3: T1_Controllers	no	no	AttitudeController.c:595.3-39
about_x_axis__DiscreteTimeIntegrator	Sa2_controller_about__X_axis	write	Process 3: T1_Controllers	no	no	AttitudeController.c:595.3-39
about_y_axis__DiscreteTimeIntegrator	Sa3_controller_about_Y_axis	read	Process 3: T1_Controllers	no	no	AttitudeController.c:701.42-78
about_y_axis__DiscreteTimeIntegrator	Sa3_controller_about_Y_axis	read	Process 3: T1_Controllers	no	no	AttitudeController.c:713.3-39
about_y_axis__DiscreteTimeIntegrator	Sa3_controller_about_Y_axis	write	Process 3: T1_Controllers	no	no	AttitudeController.c:713.3-39
DiscreteIntegrator_wrapper	TASK_T1_Controllers	read	Process 3: T1_Controllers	no	no	copter_var2_4.c:1937.31-57
DiscreteIntegrator_wrapper	TASK_T1_Controllers	read	Process 3: T1_Controllers	no	no	copter_var2_4.c:1938.4-30
DiscreteIntegrator_wrapper	TASK_T1_Controllers	write	Process 3: T1_Controllers	no	no	copter_var2_4.c:1938.4-30
DiscreteTimeIntegrator	STEP_HeightObserver	read	Process 6: T4_HeightObserver	no	no	HeightObserver.c:777.33-55
DiscreteTimeIntegrator	STEP_HeightObserver	read	Process 6: T4_HeightObserver	no	no	HeightObserver.c:786.43-65

Figure 3: Excerpt of Astrée's data flow report

enables cAMP to compute how often each *OsTasks* T_i is executed in each hyperperiod:

$$f_{Ti} = \frac{p(T_i)}{H} \tag{3}$$

This is turn can be multiplied with the access frequencies of reads f_r and writes f_w on a data item D_i determined by Astrée's data flow analysis:

$$f_{Di} = f_r * f_{Ti} + f_w * f_{Ti} \tag{4}$$

For aperiodic and sporadic tasks the minimum interarrival time provided in the ARXML is used as the periods $p(T_i)$ value. This constitutes an over approximation of actual accesses. In future work alternative ways to include such tasks will be evaluated.

5.4 Mapping

The underlying memory model for the mapping models the whole system memory as a set of linear, continuous hardware memories M. Each memory M_i has a starting address a_{start}, an end address a_{end}, and therefore, a size of $s_i = a_{end} - a_{start} + 1$ in Bytes. Each memory is split into sections S, which in turn contain data D. Each section is 64-bit aligned, due to a constraint on the placement of memory protection units for the AURIX TriCore [18]. This is also reflected in the generated linker file. After the mapping data is mapped to a section, which is mapped to memory $D \rightarrow S \rightarrow M$. Data of primitive data types (e.g., int, char) have the size of their data type in Bytes. Data types are aligned in accordance with Tasking's TriCore alignment requirements [39]. Complex data types' sizes are computed by recursively accumulating the contained type's sizes and respecting word alignment. In the memory model other types of alignment (such as that of *packed structs*) is not reflected, but the linker can still align these as packed.

First the set of all mappable data is sorted according to its access frequency f_{Di}: $D = \{D_a, D_b, \ldots : f_{Db} \leq f_{Da}\}$. For the data item with the highest access frequency a mapping is derived until all data is mapped or there is no memory with free space for mapping available. The mapping rule set is described by Algorithm 1 for all data $D = \{D_0, D_1, \ldots\}$.

M is the set of all possible memories. In general the algorithm tries to map data to the tightly-coupled (local) memory that is closest to the core with the highest access frequency. If this fails other accessing cores are tried in decreasing order of access frequency. If this fails all other memories are tried, starting with tightly-coupled SRAM, bus-accessed SRAM and finally available Flash memories. This process ensures that as long as memories have enough space, all data is mapped and all mapping constraints are satisfied.

The mapping rules were designed based on the hardware layout of the targeted AURIX TriCore TC2xx series (see Figure 1). Because of XCP-tooling limitations, Calibration and Measurement Data is mapped to Flash Memories only. Constant data is mapped to Flash Memories, but cached by core-local caches. This is only used for read-only data, as the TriCore does not provide hardware cache coherency, and the decision was made that no software cache coherency will be implemented. An alternative would be to map constants like any other data, and use the core-local caches as an extension of the tightly-coupled memories. The order in which memories are tried for a fit is determined by the access speed described in the Aurix Reference Manual [18] and measured by [26] for each of the memory types.

The *UpdateMemory* function updates the section S_n of the memory M_n with the size of D_i. If there is not already a section S_n for M_n, it is added to the memory. A section contains all variables of a process P, belonging to the same data class, with the same access mode A.

The mapping for functions occurs in a similar, but simplified fashion. Only instruction memories are used for functions and no special function types exist, only the core and process access behavior is used for the mapping.

After a mapping has been derived for all data and functions, cAMP adapts the model-based code, (in this case using TargetLinks API) to add compiler-dependent syntax for section assignment. For example using `#pragma section type "section_name"` to the symbol definition. In conjunction a linker script is generated, mapping the defined sections to hardware memories during the linking stage. Alternatively, a *MemMap.h* file could be generated which serves a similar purpose in the AUTOSAR workflow. More on MemMap.h can be found in [8]. Each section's start and end addresses are also exported as symbols, which can be used to configure each core's MPU to enforce the isolation requirements. Therefore each section S can be used as a memory protection region $R(D)$.

5.5 Current Limitations

The current implementation is limited in some aspects:

- **Mapping of stack memory**: Currently stack objects are treated as if they were already allocated. Mapping of stack objects can be included by using the stack usage information emitted by StackAnalyzer for each *OsTask* stack object and allocating these objects first on the *EcuC's* tightly-coupled memory, to which the *OsTask* is deployed.

Algorithm 1 Mapping algorithm used by cAMP

Require: D is sorted by f_D
Require: M is sorted as local SRAM > global SRAM > global Flash

procedure MAPALL(M, D)
  for $D_i \in D$ **do**
    if All $M_i \in M$ are full **then return** Fail
    $(M_n, S_n) \leftarrow$ MAP(D_i, M)
    if $M_n = \text{None} \wedge S_n = \text{None}$ **then return** Fail
    else
      UPDATEMEMORY(M_n, S_n, D_i)
  return Success

function MAP(D_i, M)
  if D_i has constraints **then**
    $M \leftarrow M -$ Constrained Memories
  if $D_i \in$ {Calibration, Measurement, Constant} **then**
    Map D_i to Flash
  else if D_i = process-local **then**
    Get local memory M_j of accessing core C_k
    $(M_n, S_n) \leftarrow$ TRYFIT(M_j, M)
  else if D_i = process-global, core-(j)-local **then**
    $(M_n, S_n) \leftarrow$ TRYFIT(M_j, M)
  else if D_i = core-global **then**
    Sort accessing cores C by access frequency
    for $C_k \in C$ **do**
      Get local memory M_j of core C_k
      $(M_n, S_n) \leftarrow$ TRYFIT(M_j, M)
  if $M_n = \text{None} \wedge S_n = \text{None}$ **then**
    Let M' be $M -$ Memories tried
    for $M_j \in M'$ **do**
      $(M_n, S_n) \leftarrow$ TRYFIT(M_j, M)
      if $M_n \neq \text{None} \wedge S_n \neq \text{None}$ **then**
        Break
  return M_n, S_n

- **Mapping of handwritten code**: Due to the way in which the mapping is added to the code, cAMP only works for generated code. In the future this can be remedied by either adapting handwritten code through transpilation or by modifying unlinked object files' section assignments. The analysis would still need the source code, as Astrée does not perform data flow analysis on a binary level.

- **Memory Mapped I/O**: The current implementation is not capable of identifying symbols representing memory mapped I/O devices or special hardware registers. This information could be retrieved by parsing *CMSIS-System View Description* [6] files for ARM processors or similar formats for other microcontroller families.

- **Hardcoded Ruleset**: The current implementation uses a hardcoded set of data classes and mapping rules. By developing a domain specific language that can express the mapping rules and the relationship between data traits and classes this can be made more flexible.

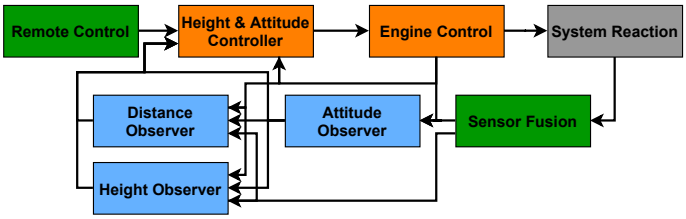


Figure 4: I4Copter control system.

- **Automated generation of the MPU configuration**: Currently only section symbols are exported to support the configuration of the MPU. In theory a configuration could be already generated automatically. But since the amount of protection regions per MPU is limited, and reprogramming at run-time causes a performance cost, the selection which sections of data need to be protected by the MPU is still done manually. cAMP still aids the configuration of the MPU by providing symbols for the start and end of each section.

- **Lack of Optimization**: The current goal of cAMP is to enable the automated, repeatable generation of memory mappings and enforce functional safety requirements. It only performs heuristic-based optimizations. The rule set could be expanded with optimizing algorithms such as [11], though this would increase the complexity of the mapping. The effect of caches is only used for constants at the moment. No consideration is given to the placement of data to achieve better cache performance.

5.6 Example

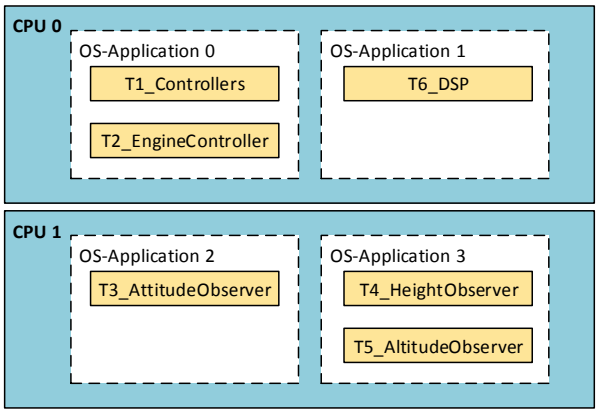
cAMP has been employed to derive memory mappings for the I4Copter [41, 42] in different configurations to demonstrate its functionality and the approach described in this paper. The I4Copter is a control system (see Figure 4) to steer a quadrocopter. It is deployed with an AUTOSAR Classic operating system to an AURIX TC277 (see Figure 1). The I4Copter system contains 7151 lines of C code which expose 206 mappable symbols.

Figure 5(a) and Figure 5(b) show two possible process deployments for the application. These were chosen to show how cAMP can be used to generate a mapping for applications where the system architecture differs. Blue boxes are execution units (CPU cores), white boxes are processes (*OsApplications*), yellow boxes are threads (*OsTasks*). Variant 2 adds additional constraints to ensure that observer processes (*OsApplications* 1, 3, 5) are exclusivly mapped to the tightly-coupled memories of the cores they are assigned to.

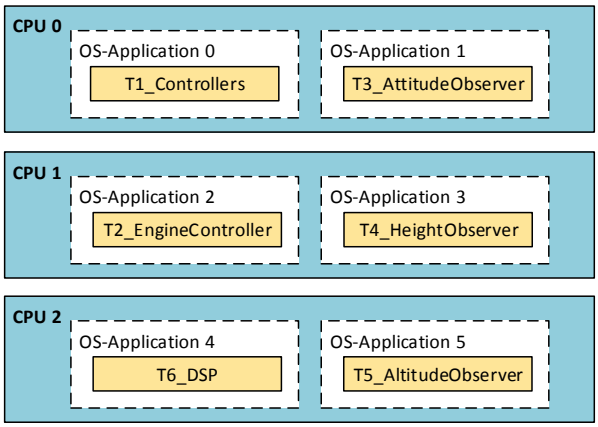
The mapping result of cAMP, generated source and linker files can be found at [12]. A summary is given in Table 3.

cAMP was able to derive a mapping for all 206 variables in both variants. The mapping process successfully returns a correct mapping as described in Algorithm 1. No data is classified as "No Class". In variant 2 all constraints are satisfied.

Due to the small size of the example application the algorithm does not map anything but data of class "Constant" to the



(a) Variant 1



(b) Variant 2

Figure 5: Deployment variants for the I4Copter

flash memory. To demonstrate the behavior for a bigger system the size of tightly-coupled memories in the input memory layout is reduced iteratively. The more it is reduced, the more data is bound to the LMU-RAM or D-Flash. This is the expected behavior and reflects the mapping rule set of cAMP, as more and more data is mapped to these memories, in ascending order of access frequency f_{Di}. This is successful until either in variant 1 everything is bound to LMU-RAM/D-Flash or until in variant 2 constraints are violated.

For either variant a mapping and the associated output files were generated in five minutes. Most of that time is spent interfacing with the code generator to adapt the configuration that adds the `#pragma section` statements. Each time cAMP runs the same mapping is derived, if given the same input.

5.7 Similar approaches

Amarnath [4] presents a very similar approach. Integer linear program (ILP) and a heuristic based greedy algorithm are compared in terms of their capability to derive a memory mapping. It concludes that a heuristic based approach for the memory mapping scales better than the ILP approach due to the dimensionality of the problem space. Giannopoulou et al. [17] primarily performs task allocating for mixed criticality systems and combines this with a heuristic based memory mapping strategy.

Table 3: Mapping determined by cAMP for variant 1, 2, and 2 without constraints

Memory	Variant 1	Variant 2	Variant 2 w/o
DSPR CPU 0	80	99	108
DSPR CPU 1	126	61	52
DSPR CPU 2	0	46	46
SRAM (LMU)	0	0	0
D-Flash (PMU)	17	17	17

Both use heuristics very similar to those used in this paper, but both are limited to the core-memory access behavior. With our approach of data classes we reflect more traits than just the core-memory access behavior and are able to reflect application specific needs with regards to the mapping through these. Neither approach is concerned with memory isolation using MPUs or mapping constraints. What is unique to our approach is the use of sound static analysis to automatically detect the access behavior using sound static analysis based on abstract interpretation. The approach to minimize task access interference presented in Giannopoulou et al. [17] could be used to enhance the heuristics of the solution presented in this paper.

6 Conclusion

The approach presented in this paper enables the automatic generation of a reproducible memory management configuration. This includes the memory mapping and the definition of memory-protection regions. The approach is less error prone than manual memory mapping and protection region definition, reproducible, and requires less effort due to automation. The manual approach for an automotive industrial sized application was estimated by engineers at Schaeffler to take around 40 person hours. cAMP was able to derive a mapping in a timespan of minutes. This is achieved by combining the information generated during a sound concurrency-aware whole-system analysis with a heuristics-based rule set that is automatically applied to generate a memory management configuration. Due to the reusability of the rule set and the application-agnostic nature of the approach it enables the tailoring of model-based generic software to the quality requirements of a specific application.

In future work the limitations in Section 5.5 will be remedied. The focus is put on enhancing the underlying memory model, supporting handwritten and third-party code, and performing optimizations. In addition we want to perform runtime benchmarks comparing cAMP to other approaches and iterations of cAMP itself.

ACKNOWLEDGMENT

This work was funded within the project ARAMiS II by the German Federal Ministry for Education and Research with the funding ID 01IS16025. The responsibility for the content remains with the authors. The authors would like to thank Peter Ulbrich for his valuable hints and ideas for this project. This work was partly supported by the German Research Foundation (Deutsche Forschungsgemeinschaft (DFG)) under grant no. SCHR 603/9-2 AORTA.

References

[1] AbsInt Angewandte Informatik GmbH. StackAnalyzer Website. http://www.absint.com/stackanalyzer.

[2] AbsInt Angewandte Informatik GmbH. *Safety Manual for aiT, Astrée, RuleChecker, StackAnalyzer*, May 2018.

[3] M. Aiken, M. Fähndrich, C. Hawblitzel, G. Hunt, and J. Larus. Deconstructing process isolation. In *MSPC '06: 2006 W'shop on Memory System Performance and Correctness*, pages 1–10, New York, NY, USA, 2006. ACM.

[4] R. Amarnath. Techniques for memory mapping on multi-core automotive embedded systems. Master's thesis, Delft University of Technology, June 2012.

[5] Arm Ltd. *Arm Cortex-R52 Processor Technical Reference Manual*, July 2020.

[6] Arm Ltd. CMSIS system view description, June 2021. https://www.keil.com/pack/doc/CMSIS/SVD/html/index.html, visited 2022-01-01.

[7] AUTOSAR (AUTomotive Open System ARchitecture). http://www.autosar.org[retrieved: Jan. 2020].

[8] AUTOSAR. Specification of memory mapping (R21-11). Technical report, Automotive Open System Architecture GbR, Nov. 2021.

[9] AUTOSAR. Specification of operating system (R21-11). Technical report, Automotive Open System Architecture GbR, Nov. 2021.

[10] AUTOSAR. Specification of rte software (R21-11). Technical report, Automotive Open System Architecture GbR, Nov. 2021.

[11] O. Avissar, R. Barua, and D. Stewart. An optimal memory allocation scheme for scratch-pad-based embedded systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 1(1):6–26, 2002.

[12] F. Bräunling, R. Hilbrich, S. Wegener, I. Stilkerich, and D. Kästner. Generic Software Tailoring Example. https://github.com/Gronner/GenericSoftwareTailoringExample.

[13] F. Bräunling, R. Hilbrich, S. Wegener, I. Stilkerich, and D. Kästner. Using generic software components for safety-critical embedded systems - an engineering framework. In *Proceedings of the 10th European Congress on Embedded Real Time Systems*, ERTS '20, Jan. 2020.

[14] *SEI CERT C Coding Standard: Rules for Developing Safe, Reliable, and Secure Systems*. June 2016.

[15] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *4th POPL*, pages 238–252, Los Angeles, CA, 1977. ACM Press.

[16] dSpace Gmbh, Rathenaustraße 26, Paderborn, Germany. *Generating TargetLink Code and ASAP2 Files for an Arbitrary Platform*, Feb. 2014.

[17] G. Giannopoulou, N. Stoimenov, P. Huang, and L. Thiele. Mapping mixed-criticality applications on multi-core architectures. In *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6. IEEE, 2014.

[18] Infineon Technologies AG, 81726 Munich, Germany. *AURIX™ – TC27x – User's Manual*, Nov. 2013.

[19] ISO 25010. *ISO/IEC 25010:2011: Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models*. ISO, Geneva, Switzerland, 2011.

[20] ISO 26262-6. *ISO 26262-6:2018: Road vehicles – Functional safety – Part 5: Product development at the software level*. ISO, Geneva, Switzerland, 2018.

[21] T. Jim, J. G. Morrisett, D. Grossman, M. W. Hicks, J. Cheney, and Y. Wang. Cyclone: A safe dialect of C. In *2002 USENIX ATC*, pages 275–288, Berkeley, CA, USA, 2002. USENIX.

[22] D. Kästner and C. Ferdinand. Proving the Absence of Stack Overflows. In *SAFECOMP '14: Proceedings of the 33th International Conference on Computer Safety, Reliability and Security*, volume 8666 of *LNCS*, pages 202–213. Springer, September 2014.

[23] D. Kästner, L. Mauborgne, S. Wilhelm, and C. Ferdinand. High-Precision Sound Analysis to Find Safety and Cybersecurity Defects. In *10th European Congress on Embedded Real Time Software and Systems (ERTS 2020)*, Toulouse, France, Jan. 2020.

[24] D. Kästner, L. Mauborgne, S. Wilhelm, and C. Ferdinand. Static Data and Control Coupling Analysis. *Submitted to the 11th European Congress on Embedded Real Time Software and Systems (ERTS 2022)*, March 2022.

[25] D. Kästner, A. Miné, L. Mauborgne, X. Rival, J. Feret, P. Cousot, A. Schmidt, H. Hille, S. Wilhelm, and C. Ferdinand. Finding All Potential Runtime Errors and Data Races in Automotive Software. In *SAE World Congress 2017*. SAE International, 2017.

[26] S. Keßler. Messbasierte analyse der zeitlichen isolationseigenschaften der aurix tc2xx familie. Master's thesis, Friedrich-Alexander-University Erlangen-Nürnberg, Mar. 2019.

[27] Microsoft et al. Gsl: Guideline support library, 2021. https://github.com/Microsoft/GSL, visited 2021-12-30.

[28] A. Miné. Static analysis of run-time errors in embedded real-time parallel C programs. *Logical Methods in Computer Science (LMCS)*, 8(26):63, Mar. 2012.

[29] A. Miné, L. Mauborgne, X. Rival, J. Feret, P. Cousot, D. Kästner, S. Wilhelm, and C. Ferdinand. Taking Static Analysis to the Next Level: Proving the Absence of Run-Time Errors and Data Races with Astrée. *Embedded Real Time Software and Systems Congress ERTS*^{2}.

[30] *MISRA C: 2021 - Guidelines for the Use of the C Language in Critical Systems*. Feb. 2019.

[31] *MISRA C++: 2008 - Guidelines for the Use of the C++ Language in Critical Systems*. June 2008.

[32] S. Mittal and J. Vetter. A survey of software techniques for using non-volatile memories for storage and main memory systems. *IEEE Transactions on Parallel and Distributed Systems*, 27, 01 2015.

[33] G. C. Necula, S. McPeak, and W. Weimer. CCured: Type-safe retrofitting of legacy code. In *POPL '02: 29th ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages*, pages 128–139, New York, NY, USA, 2002. ACM.

[34] OSEK/VDX. *OSEK/VDX Operating System. Version 2.2.3.*, 2005.

[35] M. Stilkerich. *Memory Protection at Option - Application-Tailored Memory Safety in Safety-Critical Embedded Systems*. PhD thesis, Friedrich-Alexander University Erlangen-Nuremberg, 2012.

[36] STMicroelectronics. *SPC58 C Line Data Sheet*, May 2021.

[37] B. Stroustrup et al. C++ core guidelines, 2021. https://github.com/isocpp/CppCoreGuidelines, visited 2021-12-30.

[38] D. Tarditi, A. S. Elliott, A. Ruef, and M. Hicks. Checked c: Making c safe by extension. In *2018 IEEE Cybersecurity Development (SecDev)*, pages 53–60, October 2018.

[39] Tasking Germany GmbH. *Alignment Requirements - Restrictions for the TriCore Architecture*, Feb. 2021.

[40] I. Troxel. Memory technology for space, Aug. 2009. https://nepp.nasa.gov/mapld_2009/talks/083109_Monday/06_Troxel_Ian_mapld09_pres_2.pdf, visited 2021-12-29.

[41] P. Ulbrich. The I4Copter project — Research platform for embedded and safety-critical system software. https://www4.cs.fau.de/Research/I4Copter/, visited 2012-07-20.

[42] P. Ulbrich, R. Kapitza, C. Harkort, R. Schmid, and W. Schröder-Preikschat. I4Copter: An adaptable and modular quadrotor platform. In *26th ACM Symp. on Applied Computing (SAC '11)*, pages 380–396, New York, NY, USA, 2011. ACM.

[43] X. Wu, J. Li, L. Zhang, E. Speight, R. Rajamony, and Y. Xie. Hybrid cache architecture with disparate memory technologies. In *Proceedings of the 36th Annual International Symposium on Computer Architecture*, ISCA '09, page 34–45, New York, NY, USA, 2009. Association for Computing Machinery.

Session We.3.B

Model Driven Engineering II

Wednesday 1st June

15:00

–

Room Lauragais

Automatic Test Generation - An Industrial Feedback

Mathilde Ducamp, David Lesens, Philippe Ranoarivony
first_name.surname@ariane.group
ArianeGroup

Keywords: Automatic Test Generation, Simulation, Model Based System Engineering
Abstract: This paper presents an industrial feedback about the use of automatic test generation in the development of a critical real-time embedded system (a space launcher).

1. Introduction: Why automatic test generation?

Test activities are known as some of the most costly phases in the development of a critical system. So, naturally, automating completely or partially these activities has been an objective of a lot of industries since several decades, with the classical objectives of decreasing the costs and the delays, while in the same time increasing the quality of the system.

Testing a system is achieved in several steps: (1) definition of the test cases (what has to be tested), (2) definition of the test scenarios (what inputs shall be provided to the system to cover the test cases), (3) execution of the tests (on a real or on a simulated platform), (4) analysis of the test's results and decision about their status (success or failed), (5) assessment of the test's coverage (generally requirement's coverage). The challenges to be solved when dealing with automatic test generation depend on the nature of the considered system: Complexity of the missions to be achieved, man in the loop, nominal scenarios or degraded ones, and so on…

The objective of this paper is to provide an industrial feedback (development of a space launcher) concerning all these steps of automation of tests. On one hand, a space launcher is a complex system: it involves complex subsystems (propulsion, avionics, flight control…), shall be more and more versatile (i.e. shall cover a huge number of types of missions) and shall implement a complex FDIR (Fault Detection, Isolation and Recovery) to reach fault tolerance objectives. On the other hand, it has no man in the loop (except during the ground phase before the lift-off) and has a quite predictable environment (wind, gravity) compared to other systems (such as automotive).

The section 2 recalls the obviously prerequisites for automating the tests of a system. The section 3 describes the approach used to develop space launchers. The section 4 details the synchronous approach used at both system and software levels and allowing a full representativeness of the used models with respect to the actual system. The section 5 compares this approach with off the shelf solutions. And finally, the section 6 provides a feedback on automatic test generation.

2. Prerequisites

In a classical approach, testing a system requires obviously the availability of the system (the product under tests) and the system requirements (allowing inferring the test's objectives). Automating the test activity requires furthermore the capability to simulate the system environment (man in the loop, occurrence of failure…) and the capability of automating the assessment of the quality of the service delivered by the system (e.g. the assessment of the consumption of energy of the system can be easily automated; the level of usability of the system may be more interpretable).

More and more often, the use of Model Based System Engineering (MBSE) allows testing the system design before its actual development. Several kinds of models are generally used, covering specific parts of the development (concepts, architecture or detailed design) and / or of the system (the system as a whole, a subsystem such as the propulsive subsystem, the avionics subsystem, the FDIR or the software subsystem). The availability of a system model makes easier the simulation of the system environment and so the automation of test's execution.

In a similar way, generating automatically the tests requires a level of formalisation of the system requirements allowing their automatic analysis by a computer. Ideally, the automatic generation of tests shall use the formalism put in place for the system development. Duplicating the requirements (first in an informal way for the system development and second in a more formal way for the automatic test generation) is indeed costly and especially error prone (the risk being a discrepancy between the two formalisms).

3. Case of the management of a space launcher

3.1. Problematic

ArianeGroup has put in place a method to develop the management of a space launcher (and more generally of a space system). The launcher design is based on a modular architecture (see [1]) where the launcher is composed of a set of

logical components. Each logical component deals with a kind of equipment (accelerometers, valves, and so on…) and contains a piece of software controlling all the equipment of this kind. For each logical component, a Finite State Machine represents its behaviour.

The consistency between the logical components is ensured by a centralised mission management executing a mission plan (aiming at fulfilling the mission objectives). The mission plan sends commands to the components which can in return send mission events to the mission plan. The FDIR monitors the launcher state and in case of failure raises an alarm. Depending on the alarm and on the launcher state, a recovery is decided, which can be executed either locally to a logical component (local recovery) or executed by the mission management (system recovery implying a change of mission).

We call launcher management the association mission management and FDIR. Our target is to automate as far as possible the testing of the launcher management.

Several modelling languages are used to describe the system

- Capella for the hardware architecture
- SysML ([2]) for the logical components with their Finite State Machines and the FDIR
- A Domain Specific Language (DSL) for the launcher management

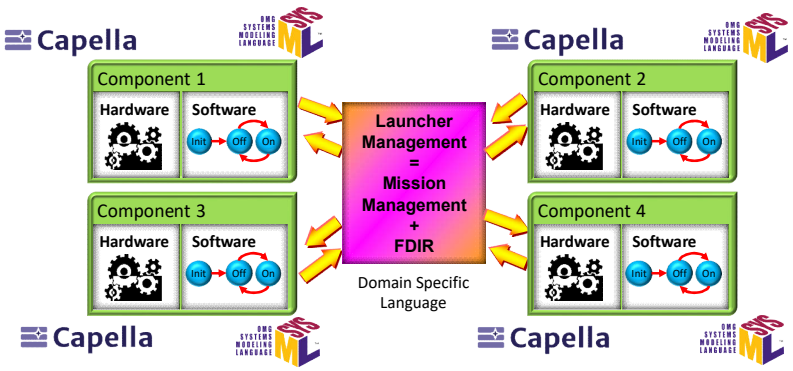


Figure 1: The modular architecture of a launcher

This approach allows developing the system in a modular way:

- Each logical component is developed independently of the other logical components depending on the maturity of its definition, for its hardware part, or for its software part.
- The launcher management is the last developed artefact allowing ensuring the consistency between all the previously developed logical components.

3.2. Semantics definition

3.2.1. Generality

Automatic testing requires first a clear unambiguous definition of the semantics of the used system artefacts. Especially in a MBSE approach, it is of prime importance to understand the links between the actual system (which development is the objective of the project) and the model representing it (aiming at developing and validating the actual system). We have thus first defined a system physical event as an event occurring at hardware level (i.e. on the actual system, such as the ignition of an engine, the release of the fairing, the occurrence of a failure, and so on). As such events are not easily handled, we have defined the notion of virtual event corresponding to the representation of the physical event by a model element, such as commands (controlling a physical events), alarms (detection of a failure), activation / deactivation of a continuous control, entry / exit of a state (being in a given state corresponding generally to a continuous action) or other events (detection of a tail-off, detection of an empty tank, and so on). The requirements and the hypotheses on the system environment are formalised on these virtual events

3.2.2. Formalisation of requirements

The requirements of the launcher management which have to be verified can then be modelled by timing constraints between the defined virtual events. For instance

- "Event1 {1} > Event2 {3} + 10s": The first occurrence of the event "Event1" shall occur more than 10 seconds after the third occurrence of the event "Event2" (the notation {i} referring to the i^{th} occurrence of the event during a test).
- "For all Event1, for some Event2, Event1 = Event2 + 5s + [0ms .. 20ms]": For each occurrence of the event "Event1", it shall exist an occurrence of the event "Event2", such that "Event1" occurs 5 seconds after "Event2" with an accuracy of 20ms.

- "State S in [5s .. 10s]": Each time the system enters the state "S", it shall stay in this state more than 5 seconds and less than 10 seconds.

To make easier the daily use of this syntax and to facilitate its acceptance, this syntax may be adapted to the tools the system engineers are accustomed: for instance an ASCII description or a tabular representation in the Excel tool (as shown in the next figure).

First event	Operation	Second event	Delta T	Jitter
Event1	=	Event2	5	[0ms .. 20ms]

Figure 2: Syntax included in Excel

The applicability of these requirements depends generally on the current mission. To ease their management, the requirements are gathered in consistent sets: e.g. such set of requirements is applicable only for a sun synchronous orbit, this other set is applicable only for a launcher configured with four boosters, and so on. The applicability of a set of requirements is formalised by SysML activity diagrams (called the requirement's architecture)

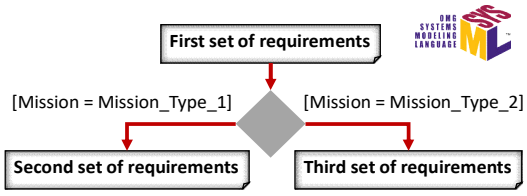


Figure 3: A SysML activity diagram describing the scheduling of set of requirements

Each node of the SysML activity diagram is either decomposed into another activity diagram, or corresponds to a set of requirements. The conditions guarding the transitions between the nodes allow controlling the launcher mission behaviour. Such guard may correspond either to a nominal behaviour or to an abnormal behaviour. So, it may involve either:

- A configuration data defining the targeted mission. E.g. "[Mission = Mission_Type_1]": In Figure 3, when the configuration data "Mission" is equal to "Mission_Type_1", the applicable sets of requirements are the sets 1 and 2. If this configuration data is equal to "Mission_Type_2", the applicable sets of requirements are the sets 1 and 3.
- Or the occurrence of an alarm. E.g. "[Alarm1 or Alarm2 in [Engine_Start .. Engine_Start+10s]]": The transition is triggered if "Alarm1" or "Alarm2" occurs in the ten seconds following the start of the engine.

These timing constraints organised by these activity diagrams play the role of oracles for the system under test.

3.2.3. Formalisation of the system environment

A system behaves as intended only in a specific environment. The formalisation of the system environment is thus of prime importance to first validate the system and second to generate automatically tests.

A specific syntax has been defined to capture such environment. E.g.

- "Event = **fix** Guidance.Date **when** Test_Level.Low_Level_Detected": The date of the "Event" is defined as the value of the parameter "Date" computed by the "Guidance" when the flag "Low_Level_Detected" is raised by the "Test_Level" function.
- "Event = **saturate** Guidance.Flag **in** [Min, Max]": The date of the "Event" is defined by
  - "Min" if the "Flag" is raised by the "Guidance" before the "Min" date.
  - The date of the raising of the "Flag" by the "Guidance" if it is raised between the "Min" date and the "Max" date.
  - "Max" if the "Flag" is not raised by the "Guidance" before the "Max" date
- "Event = **min** Flag1 **or** Flag2 **in** [Min, Max]": The date of the "Event" is defined by
  - "Min" if "Flag1" or "Flag2" is raised before the "Min" date.
  - The date of the raising of "Flag1" if it is raised between the "Min" date and the "Max" date and before "Flag2".
  - The date of the raising of "Flag2" if it is raised between the "Min" date and the "Max" date and before "Flag1".
  - "Max" if neither "Flag1" nor "Flag2" are raised before the "Max" date

- And so on

3.3. Automatic test generation

3.3.1. Approach

Generating a system test may be quite complex. For instance, to test the behaviour of the system in case of sub-propulsive engine, one has to simulate the behaviour of an accelerometer to detect too early a thrust tail-off. The MBSE approach allows greatly simplifying the test generation thanks to abstractions. As our objective is the validation of the launcher management, the environment of the launcher (such as the engine, the accelerometer or the navigation algorithm) can altogether be modelled by the raising of a single flag at the desired time (at a nominal time, too early or too late depending on the test objective).

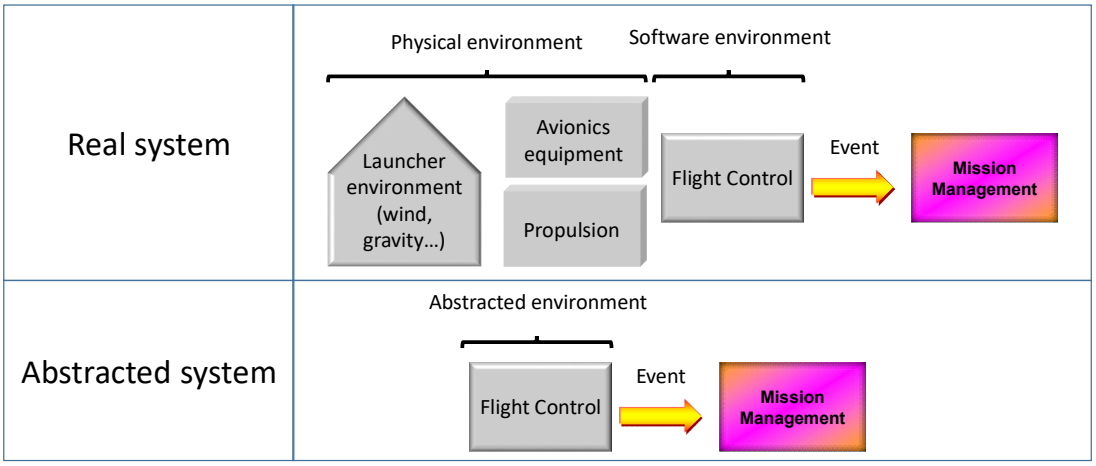


Figure 4 Abstraction of the environment of the product under test

The raising of the Event is specified by the syntax defined in section 3.2 (such as “Event = **saturate** Guidance.Flag **in** [Min, Max]”)

The coverage of requirements requires first

- The coverage of the SysML model (the requirement’s architecture) describing the architecture of the sets of requirements (shown Figure 3). The customisation data and the occurrences of alarms are selected to reach this coverage.

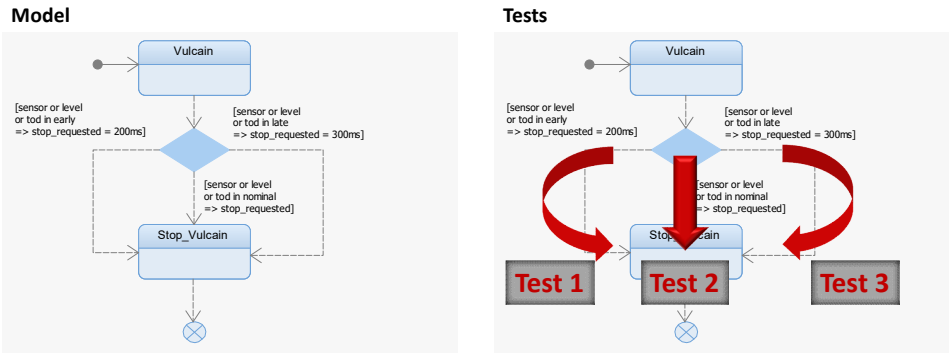


Figure 5: Automatic Test Generation: Coverage of the SysML model

- The coverage of the complex events. E.g. for a saturation, three tests may be generated:
  1. With the event detected before the minimal time,

2. Between the minimal and maximal times and
3. After the maximal time.

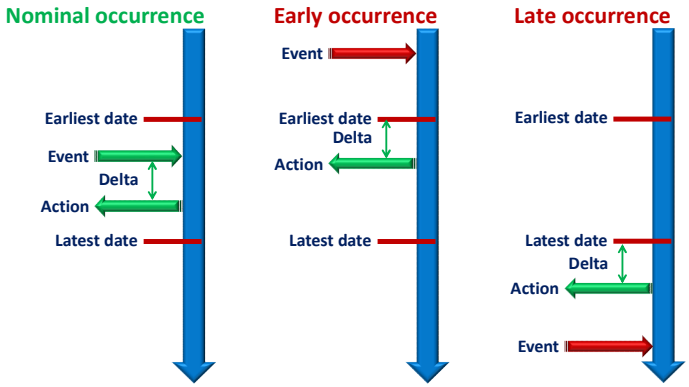


Figure 6: Automatic Test Generation: Coverage of the input events

This first coverage can be completed by robustness tests. On one hand, a recovery (local or system) may have a functional impact (modification of the mission) or a temporal one (delay of a nominal command). On the other hand, a nominal action may prevent the immediate execution of a recovery and delay it. An automatic test generation strategy may be to simulate a failure each time a nominal action is executed, just before and / or just after.

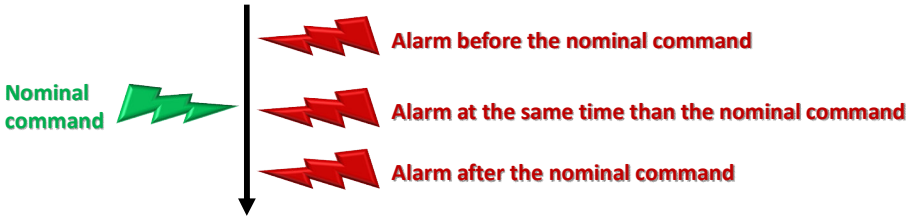


Figure 7 : Automatic Test Generation: Robustness tests

3.3.2. Strategies

As the number of automatically generated test can very quickly becomes very high, the approach shall allow the tester specifying the maximal number of failures to be considered for a test. The following strategies can be applied

- For the requirement's architecture
  - Selection of configuration data forbidding or allowing some sets of requirements
  - Coverage of the complete SysML model of the requirement's architecture
- For the complex events
  - Coverage only of the nominal cases (events generated in the middle of the [Min, Max] time windows)
  - Limit tests (events generated at the lower bound Min and at the upper bound Max of the [Min, Max] time window)
  - Degraded cases (events generated before the lower bound Min and after the upper bound Max of the [Min, Max] time window)
  - Specification of the maximal number of limit tests and of degraded cases
- For the robustness tests
  - Generation of alarms before, during or after each nominal events
  - Generation of all the alarms
  - Generation of a user specified set of alarms

- Generation of the alarm with the longest recovery

3.3.3. Implementation

As described in the previous sections, this approach requires first the development of a set of models:

- A model of the product under test through a SysML model completed by a Domain Specific Language
- A requirement's model through Excel sheets completed by architecture formalized using SysML activity diagrams. This model (Excel + SysML) is used as an oracle of success of the tests.
- A model of the input events, i.e. of the system environment, through a Domain Specific Language

Two tools allow generating, executing and analysing the tests

- A simulator of the product under test (and of its environment).
- A test supervisor (developed in Java) able to
  1. Read the test model,
  2. Provide consigns to the simulator,
  3. Analyse the simulation results and
  4. Generate a test report

In order to improve the simulator performance (i.e. to reduce the duration of the simulations), there is no direct communication between the supervisor and the simulator. The supervisor provides consigns of simulation to the simulator through an ASCII file. This consign file is made of interdependent constraints representing the requirements and the hypotheses on the environment. The simulator interprets this consign file, performs the requested simulations by solving and propagating the constraints, and generate simulation traces. The supervisor analyses the generated simulation traces.

The simulator is thus composed of two distinct parts:

1. The Ada code of the flight software (automatically generated from the SysML model of the flight software)
2. The simulation engine (also coded in Ada) composed of
   - A parser
   - A propagator of constraints

The following figure summarizes this automatic test generation approach:

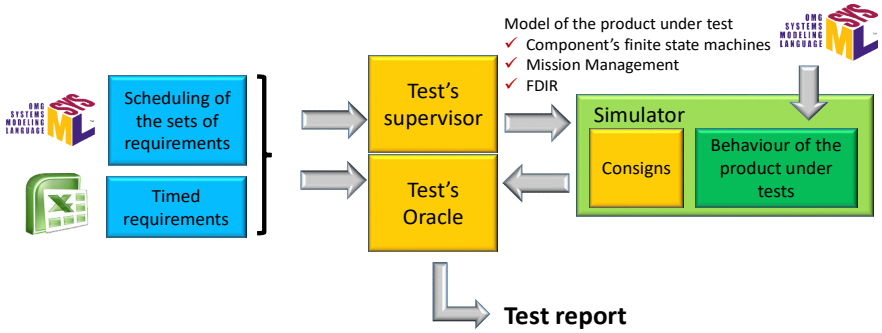


Figure 8: The automatic test generation and simulation framework

4. Representativeness of the abstraction

Performing qualification tests on a model requires an appropriate representativeness of the used model with respect to the actual system. This is insured by a synchronous approach both at system and software levels.

The wording "synchronous approach" is inspired from [3].

> *The synchronous abstraction makes reasoning about time in a synchronous program a lot easier, thanks to the notion of logical ticks: a synchronous program reacts to its environment in a sequence of ticks, and computations within a tick are assumed to be instantaneous, i.e., as if the processor executing them were infinitely fast. [...] At a more fundamental level, the synchronous abstraction eliminates the non-determinism resulting from the interleaving of concurrent behaviours.*

This notion has been extended to the avionics system thanks to the use of a Time Triggered Ethernet (or TTE, see RD [4]) and to software with multithreading architecture.

- The Time-Triggered Ethernet (SAE AS6802) (also known as TTEthernet or TTE) standard defines a fault-tolerant synchronization strategy for building and maintaining synchronized time in Ethernet networks
- In a multithreading software architecture, the threads are pre-emptive (a thread with a high priority can interrupt a thread with a low priority), with fix priority, cyclic and harmonic (the period of a thread is a multiple of the period of the thread just faster).
- All the communication are performed in a time triggered way: The communications between the avionics equipment, and between the software threads are performed at predefined moments of time.

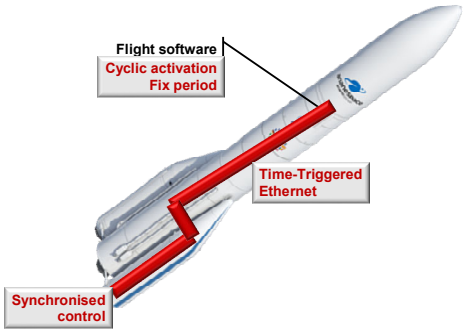


Figure 9 : Synchronous architecture at system level

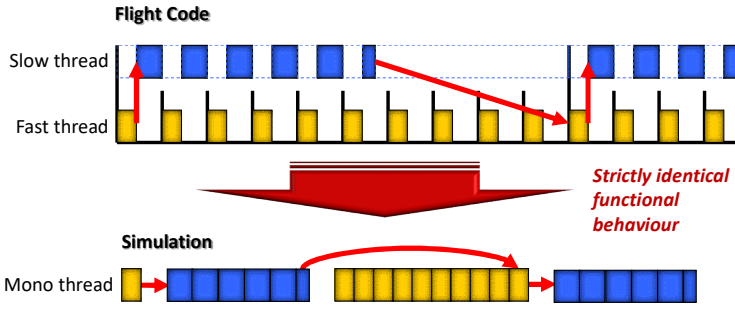


Figure 10 : Synchronous architecture at software level

In Figure 10, the slow thread (in blue) is 10 times slower that the fast tread (in yellow). The slow thread is suspended at each execution of the fast thread. It reads all its inputs at the beginning of its execution (i.e. at cycle 0). The date of end of the execution of the slow thread depends on its execution time and the execution time of each cycle of the fast thread and thus varies from one cycle to another. In this example, the slow thread ends at the cycle 5. However, in order to ensure a functional behaviour independent from the execution times, it delivers its outputs at the end of its cyclic execution (i.e. at cycle 9; in practice this exchange of data is performed during a time-triggered rendezvous initiated by the fastest thread). So, the fast thread can read the outputs of the slow thread at cycle 10. In this example, the communications between the two threads is implemented through two rendezvous symbolized by the red arrows (at cycles 0 and 9). The slow thread is said to have an offset or shift of 0 (it starts at the cycle 0) and a deadline of 10 (it ends after 10 cycles).

The slowest software thread is triggered by the TTE. During a TTE cycle, time windows are allocated to the communications with sensors and to the communication with actuators. The measurement provided by sensors at cycle n are used by the software at cycle n+1 and the corresponding commands are sent to the actuators at cycle n+2 according to a LINO approach (Last cycle In, Next cycle Out) as shown Figure 11.

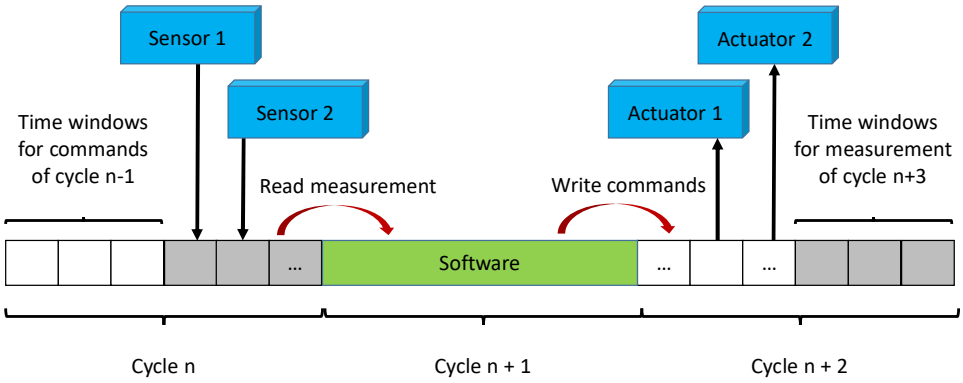


Figure 11 : Hardware / Software synchronisation

This synchronous and LINO architecture makes the system behaviour independent from the communication duration and from the computation duration. It is fully deterministic and allows a complete representativeness of the MBSE model with respect to the actual system (except for numerical accuracy, which is not needed for the validation of the launcher sequence).

5. Comparison with other approached

Several solutions of automatic generation are available on the market, either as academic products or as commercial tools. ArianeGroup has evaluated some of these solutions in the past. For instance:

- UPPAAL (see [5]): Uppaal is an integrated tool environment for modelling, validation and verification of real-time systems modelled as networks of timed automata, extended with data types (bounded integers, arrays, etc.). It uses timed automata as input formalism.
- Lutin (see [6]): from the Verimag Synchronous Reactive Toolbox, industrialised by the Stimulus tool (see [7]). Lutin is a language to program stochastic reactive systems. It has been designed to model environments and perform automated testing of reactive systems.
- MaTeLo from All4Tec (see [8]) allows generating test from a proprietary modelling language
- TGV (see [9]) is a tool for the generation of conformance test suites for protocols. TGV takes as entries a description of a protocol's behaviour and a test purpose. It applies algorithms coming from verification technology to produce automatically conformance test suites.
- Etc.

The main advantage of these tools is that they benefit from several years or decades of academic research. They may for instance include language with a high representativeness, powerful model checker and advanced analysis engines.

They all have however several drawbacks

- They relies on a formalism which is not used by the ArianeGroup system designers. The use of these tools requires thus a translator from the ArianeGroup domain specific language to this specific formalism. It has been judged that the risk of errors in this translator may cancel the benefit of the tool

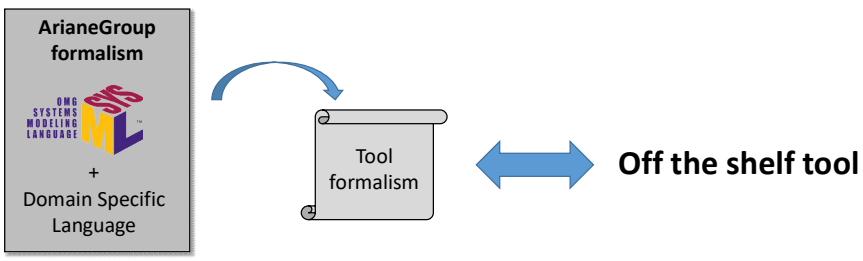


Figure 12 : Using an off the shelf solution

- Even if these tools may contain powerful engines, their goal is to be as generic as possible. We have experienced that a simple but specific tool may be more efficient than a complex but generic tool.
- These tools have to be maintained during several decades. The costs of the licences on a so long period may be quite high and there is a non-negligible risk that a commercial tool becomes unavailable

6. Feedback on automatic test generation

This section presents some feedbacks about automatic test generation.

6.1. Dashed hopes and real benefits

After several attempts to deploy generic on the shelf automatic test generation techniques, we have reach the conclusion that it was unavoidable to develop processes and tools specific to the product under tests.

As any improvement actions, the main hope of automatic test generation is an immediate decrease of costs and delays, even a complete suppression of the manual test activities. But achieving these results requires in practice first a significant investment in the process and in the tools, and then in the modelling activities for the product under tests and in the formalisation of the requirements. The development and validation teams have to be trained and opened to the change of their habits. The manual test activities are not completely suppressed, but are limited to the manual analysis of the failed tests.

But as soon as these elements are in place, the return on investment may be considerable, in terms of decrease of delays and especially improvement of the quality. We have been able to automatically generate and execute several thousands of tests in a few hours whereas it may have taken several weeks or months to generate and execute a much lower number of tests.

6.2. Unexpected benefits

The main unexpected benefit of such automatic test generation framework is its ability to be very easily extended with new features: Coverage of the product under tests and of the requirements, automated analyses, report / documentation generation, extension of the expressiveness power of the framework and so on.

We have for instance added to our framework the capability to specify the amount of resources (power, electro-valves, pyrotechnic commands, and so on) a component of the system is authorised to use. Such information allows the tools performing an automated budget analysis.

7. Conclusion

This paper has presented an example of automatic test generation framework developed specifically to a dedicated kind of product. Indeed, we had reach the conclusion that a too generic framework, even if often cheaper (thanks to its genericity) is also more difficult to deploy and not so efficient (also because of its genericity). Associated to a MBSE approach (both for the requirements and for the design of the system) and of simulation techniques, the return on investment may be considerable after an unavoidable investment.

8. References

[1] Launcher Sequential Analysis – M. Ducamp, J. Grand, D. Lesens, D. Mercier – ERTS 2018

[2] SysML - www.omgsysml.org

[3] Synchronous programming of reactive systems" – Nicolas Halbwachs, Kluwer Academic Publishers, 1993. http://www-verimag.imag.fr/~halbwach/newbook.pdf

[4] TTE - https://www.tttech.com/products/technologies/time-triggered-ethernet/ (see also SAE AS6802)

[5] UPPAAL - https://uppaal.org/

[6] Lutin - https://www-verimag.imag.fr/Lutin.html

[7] Stimulus - https://www.3ds.com/fr/produits-et-services/catia/produits/stimulus/

[8] MaTeLo - https://matelo.all4tec.com/

[9] TGV - https://www-verimag.univ-grenoble-alpes.fr/TGV.html

ROS communications profiling for bus load analysis from AADL

SENN Eric
Lab-STICC / Université de Bretagne Sud
Lorient, France
eric.senn@univ-ubs.fr

***Abstract*—This paper presents a case study which settles the basis for a modeling approach dedicated to robotic applications build upon ROS, the Robot Operating System. The Architecture Analysis and Design Language (AADL) is used with specific properties that allow to analyze bus load and CPU load of the embedded hardware. This paper focuses on bus load analysis: a profiling of ROS communication services, based on standard performance reporting tools from Linux, permits to determine the values of those properties. A library of AADL models is proposed, both for ROS services and for robots and embedded computer boards. We explain how this library is built, and show how to use components from this library to model the robotic system, including the software, the hardware, and the deployment of the software components on the hardware platform. Then we demonstrate bus load analysis using the analysis tool included in OSATE2 (Open Source AADL2 Tool Environment). Combined with CPU load analysis, exploration of deployment solutions can then be achieved.**

***Index Terms*—Robotics, Embedded System, Model Driven Engineering, Architecture Analysis Design Language AADL, Robot Operating System ROS, Profiling**

I. Introduction

In the process of developing our mobile robotics application, we are constantly confronted with situations where a robot does not behave as fast as expected, or worst, does not reach the goals of its missions. At design, however, everything should work fine since, of course, every bug has been corrected. Many times, the reason why this happens is that the robot's embedded hardware is, at some point, overloaded by the application demand. We are, in our case, using the ROS (*Robot Operating System*) middleware [1]. While easing the programming of applications, bringing standard communication and synchronization mechanisms over a wide variety of platforms, this middleware adds a significant computation overhead. The knowledge of this overhead will drive the choice of the embedded computer boards and communication devices, of the software architecture, and of its deployment.

Robotic applications getting more complex, we soon needed a method, and the support of a tool, to help us with the modeling and the analysis of our software and hardware architectures. An important point is that we needed something simple, and fast. We did not want to spend an excessive time on modeling our application, and profiling its components; and we were ready to sacrifice some precision for that. So we always tried to use out-of-the-box language and tools as long as that was all right with our need which is (i) firstly to understand why performances constraints may be or not satisfied in different situations, and that means, to find where the bottlenecks are (ii) secondly to determine a solution in the design space that meets the application requirements.

We finally came with using AADL (*Architecture Analysis and Design Language*) [2] as a modeling language, and OSATE2, with it set of tools to analyze software demand vs hardware capacity [3]. We are mainly confronted with two types of bottlenecks: CPU and communication bus overload, and OSATE2 has everything to check this. Again, for the sake of simplicity, our profiling approach is based on standard Linux performance reporting and analysis tools.

Regarding bus load, we will present here two main aspects of our works. First we need to develop a model for the different components of our ROS based applications. We are indeed building an AADL library of ROS components, beginning with the most commonly used services in robotics applications (sensors interfacing, localization, mapping, naviguation ...), and the robots, devices, and computer boards we are using. Then we have to profile the component demand over CPU and communication resources. From this profiling, specific properties are added to the model for analysis purpose. In this paper, we will only focus on communication requirements analysis. Finally, we use our library of models, both for the software and hardware parts or our application, we describe the deployment of the software components on the hardware resources, and we use OSATE2 analysis tools to report on the performance.

II. Related works

Setting up robotic applications, even with the help of ROS, is a complex task since they usually involve several services, with many communications between them, each service with a large set of parameters to adjust [4]. One first question is to determine how the behavior of the robot should be evaluated, according to the objective of the mission. This could be in term of speeds or accuracy for instance like in [5] for trajectory planning, or in [6] for an automated guided vehicle application. Then understanding why the robot is not fast or precise enough implies to investigate how the hardware is stressed by the software. A full navigation stack is studied in [7] and high processor loads are measured, as well as slow messages frequencies. The hardware is obviously not fitted to the software demand. The values of ROS parameters

impact this demand, like shown in [8], where the CPU load is measured for different nodes in a complete localization and mapping (SLAM) application. Node per node, the best parameters configuration is thus determined.

Still, the way the software is deployed on the hardware needs to be considered. In [9], an heuristic is proposed to determine together parameters values and nodes allocations for maximizing performance and minimizing hardware resources. Common monitoring tools from Linux and ROS are used to measure average CPU loads and messages frequencies, a choice that we have also made for our approach. More detailed studies of the load generated by a ROS application have been proposed, like in [10]. However, the contribution of each node is not distinguished, and a large number of tests ought to be done, a thing that we want to avoid.

To get a comprehensive view of the whole system, including its software and hardware parts, and the way software components are bound to the hardware, a model is more than needed. The *Architecture Analysis & Design Language* (AADL) [11] has been proposed as a standard for model based design of mission critical embedded systems, like a robotic application often is. The language is also commonly used for modeling multiprocessors systems [12], like computer boards found in nowadays robots. The *Open Source AADL2 Tool Environment* (OSATE2) supports the writing, checking, and the manipulation of AADL models. It also comes with several plugins, which, in association with dedicated parameters to be defined in the models, allow to perform different analysis regarding the performance of the system. Recently, [13] demonstrated the benefit from modeling and developing ROS based robotic application with AADL. The author also proposes an automatic generation of ROS code from AADL [14]. The AADL model is software centric however and hardware allocation is not addressed

AADL modeling, hardware profiling, and deployment analysis, have been combined in the work reported in [15]. The approach for measuring compute execution time is however more complex since it involves modification and rebuilding of the source code for every ROS node. This involves extra work that we want to avoid, firstly for our own programs, and moreover for entire ROS on-the-shelf packages where the difficulty of such re-writing arises significantly. This approach, beside being time consuming, is also intrusive, and its impact on measured performances should be evaluated. Nodes are also considered independently, whereas we observe a modification of performance when they are connected to others; that pleads for the necessity to profile a node in its proper usage context. In the proposed models, an AADL virtual bus component is used to describe ROS communications. That unfortunately prevents the use of the OSATE2 bus load analysis tool. Also the MIPS budget properties are not set, and MIPS demand can not be checked at the processes level.

III. Robotic Application

To illustrate our approach, we use the simple application shown on figure 1. This application involves several services deployed both on the robot with its embedded computer board, and on a remote laptop. In ROS, a service is implemented as a *node*. Nodes communicate through virtual channels called *topics*. A node may publish on a topic, subscribe to a topic, or do the both in the same time. The following nodes are found in our application:

- usb_cam captures the video stream from the camera (connected to the computer board USB socket) and publishes it on the rgb_image_raw_out topic,
- ocv_color_tracking subscribes to this topic and processes the video stream to find the position of the target according to its color,
- imgview subscribes to the same topic and displays the video on the screen of the remote laptop,
- pos_to_cmd gets the target position on the target_pos_out topic and issues velocity commands on cmd_vel_out to drive the robot towards this position,
- rosaria translates those velocity commands to low-level commands for its differential drive, i.e. the two wheels actuators,
- sonar_alert detects potential collisions and stops the robot if imminent,
- on the remote computer, joy reads the joystick controls and publishes them on the joymsg_o topic.
- joyteleop converts those controls into velocity commands, which are sent to the robot. It is thus possible to teleoperate the robot from the laptop, bringing it to the desired position where it can start its mission.

The robot is a Pioneer3DX from Adept/Mobile Robots. It is equipped with an Odroid XU4 board from Hardkernel, which carries a Samsung Exynos 5422 multi-processor SoC.

Different types of communication take place in our application. All communications are in charge of the ROS middleware, and orchestrated by the ROS master node. Every other node registers to the master when it starts. When a node is ready to publish messages on a topic, it advertises this topic to the master. Another node that wishes to read those messages subscribes to the associated topic with the master. The master informs the subscriber node that a new publisher is ready. Then the subscriber directly contacts the publisher and a connection is established between the two nodes, using a TCP/IP socket like protocol called TCPROS. When latency is more important than reliable transport, the UDPROS transport might also be used.

One of the strengths of ROS is that the same communication mechanism is used, even when nodes are located on different computers, or robots, and whatever the communication media is.

In our application indeed, different media are used:

- between two nodes located on the robot's computer board the communication is purely software and can be seen as taking place on a virtual bus. This is the case, for instance, between usbcam and ocv_color_tracking (con6).
- between usbcam on the robot's board and imgview on the remote laptop, the connection takes place on a wired Ethernet or wireless WiFi link (con12).
- the connection between the robot's computer board and its sensors and actuators is done, here, through USB ports.

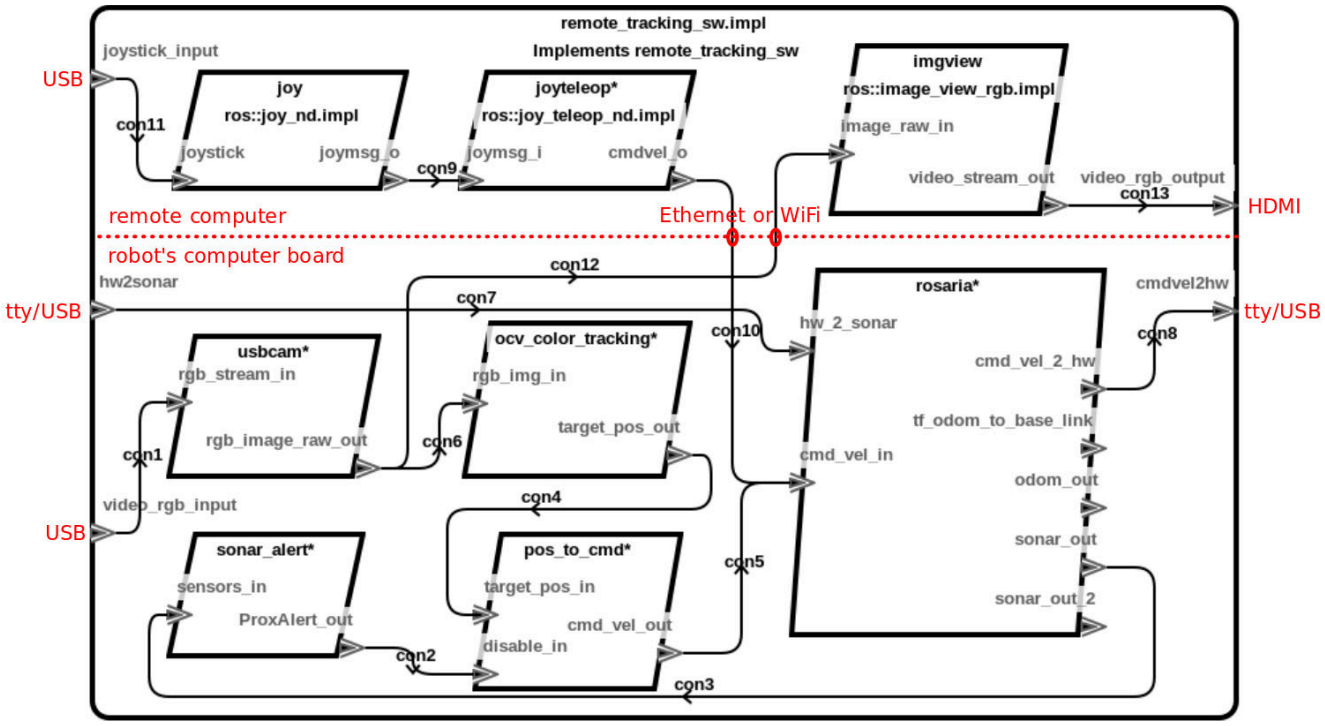


Fig. 1. Robotic application.

ROS nodes in charge of those communications can be seen as driver nodes, each one dedicated to a specific device. In our system, the USB camera is mapped to a *usb* device in OS file system, and the robot's USB/serial interface to a *ttyusb* device. On this last connection, several messages may travel if requested (velocity commands, sonar measurement, odometry feedback ...).

Depending on the application deployment scheme, connections may be bound to different media, virtual or physical. For example, the joy and the joyteleop nodes might be deployed on the robot's computer board, and the connection from joyteleop node to rosaria would be bound to the ROS virtual bus. This reflects the situation where the joystick is directly plug to the robot. In our case, tele-operation is needed and, since the joystick is plugged to the remote computer, the connection is bound to a physical WiFi or Ethernet link. Thus one message may travel on different media, but we can also have several messages traveling on one unique medium. By "message" we mean here: the stream of messages produced on a topic by a publisher node.

To determine if a message, or a set of messages, can travel on a medium, we must know:

- the message bandwidth requirement, which depends on the message size and frequency. This property is generally quite simple to get, simply multiplying the message size by the frequency. Consider the example of the video stream produced by the node usb_cam: the data size for the associated message is 640×480 (the image resolution) $\times 3$ (the number of channels: R, G, B) $\times 1$ (the number of bytes per pixel, here we are dealing with 8 bits per channel images). The result is 921600 Bytes.
- the medium bandwidth capacity. This feature is measured on the virtual or physical link to which connections may be bound.

IV. Bandwidth capacity profiling

To profile the bandwidth capacity of our connections, we use two specific nodes: a producer node that publishes messages with different sizes at different frequencies on a ROS topic, and a consumer node that subscribes to this topic.

The actual messages rate and bandwidth are measured with two standard reporting tools from the ROS toolbox: `rostopic hz` and `rostopic bw`. Measures are averaged by the tools during one second long windows.

The different data block sizes used are 32768, 65536, 131072 and 262144 Bytes, and the frequencies are: 100, 1000, 5000, and 10000 Hz. Every combination of block size and frequency is probed 6 times, during 6 seconds. Reported frequencies and bandwidths are then filtered by a median filter.

A. ROS virtual bus bandwidth capacity

We begin by showing the results for the ROS virtual bus. Figure 2 shows, with blue points, the measured bandwidth on this bus when both the producer and consumer are running on the four Cortex A15 cores of the Odroid XU4 board. Red squares show the bandwidth requested by the producer node. A saturation can be observed, and the average value of the bandwidth when the connection is saturated, noted BW sat, is 166 MBytes/s. The difference between bandwidth demand (surface in red) and bandwidth capacity (in blue) appears more clearly on figure 3.

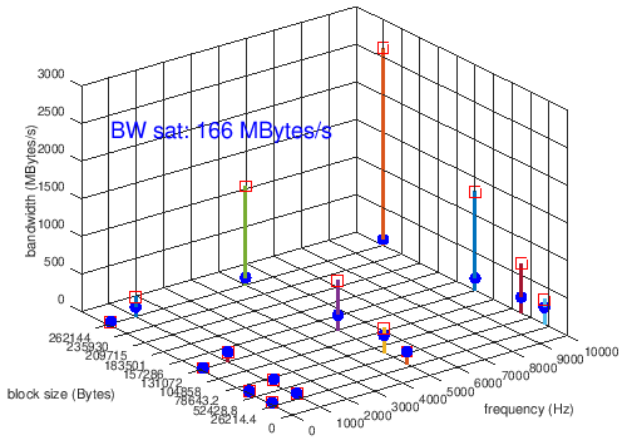


Fig. 2. ROS virtual bus profiling nodes on A15 cores.

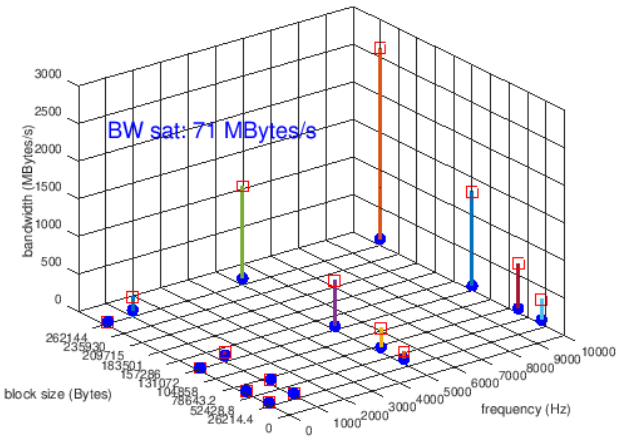


Fig. 4. ROS virtual bus profiling, nodes on A7 cores.

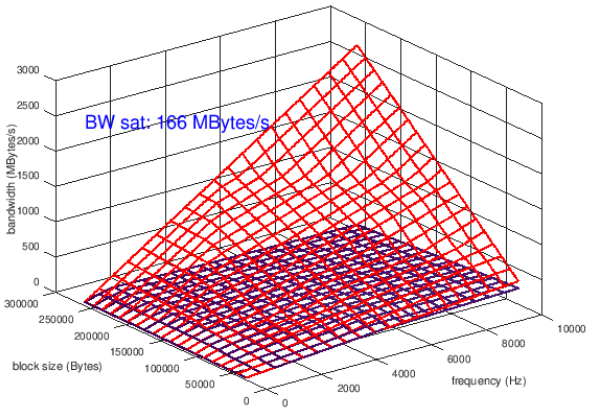


Fig. 3. ROS virtual bus capacity against demand, nodes on A15 cores.

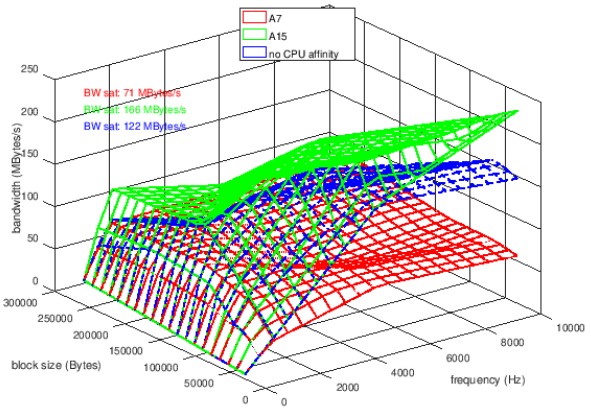


Fig. 5. ROS virtual bus capacity saturation vs CPU affinity.

The same experiment was conducted with the producer and consumer running on the four Cortex A7 cores of the Exynos SoC. Figure 4 shows the results.

It appears that the average maximum bandwidth is lower than before since BW sat is now 71 MBytes/s. That is not surprising since, the bus being virtual and the connection being established between two nodes, the speed of the processors on which those nodes are running naturally impacts the performance. Indeed, we have profiled the MIPS capacity of the Exynos 5422 SoC cores, thanks to a simple 7z LZMA benchmark, and we got 1100 MIPS for the A7 and 2000 MIPS for the A15. Thus, it seemed also interesting to profile the connection bandwidth in the standard situation where no CPU affinity is set for any of the processes in the robotic application. The maximum bandwidth is now 122 MBytes/s. Figure 5 shows how the measured bandwidth evolves with the message size and frequency in the three situations mentioned before. The situation where no CPU affinity is set appears in the middle of the two others. Since the operating system is then free to run every process on either the big or little cores, the average MIPS capacity of the whole SoC is between the capacity obtained with the four A15 cores, and the one obtained with the four A7 cores. This results seems quite normal.

B. TCPROS over Ethernet bandwidth capacity

This time, we want to profile TCPROS over wired Ethernet. The producer node is on the robot's computer board, and the consumer on the remote PC. This is the configuration we face usually when data streams from the robot's sensors have to be used, or simply echoed on a remote computer.

Figure 6 shows the bandwidth measurements for the same combinations of message sizes and frequencies as before. Bandwidth saturation appears at 103 MBytes/s, 208 MBytes/s, and 253 MBytes/s, respectively when the producer and consumer affinity is set to the Cortex A7 cores, Cortex A15 cores, or is not set. This values are significantly under the maximum 916 MBytes/s that we have benchmarked (using the standard Linux tool *iperf*) for the Ethernet link in our system (which is itself not far from the theoretical 1 GBytes/s given for fast

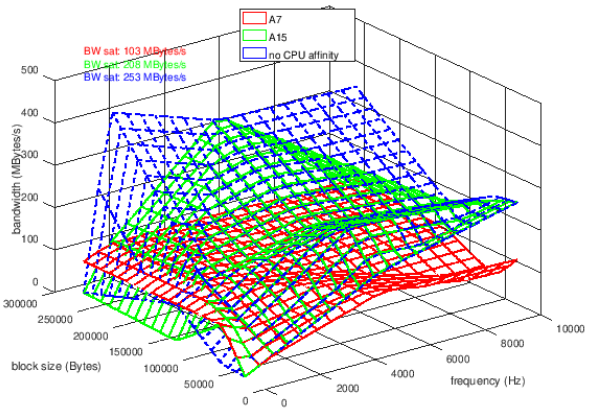


Fig. 6. TCPROS over Ethernet capacity saturation vs CPU affinity.

Ethernet over physical links). Again, the actual bandwidth is limited by the CPU speed, and, as expected, A7 cores give the lowest performances.

C. TCPROS over WiFi bandwidth capacity

When WiFi is used to transport ROS messages from a node to another, the connection speed is only slightly under the preceding one. As presented on Figure 7, it ranges from 103 to 173 MBytes/s with the CPU affinity. The theoretical speed we should observed, according to our WiFi devices specifications, should be 450 MBytes/s. The same conclusion are drawn from these new results. The actual bandwidth depends on the CPU ability to implement the TCPROS connection over the physical link.

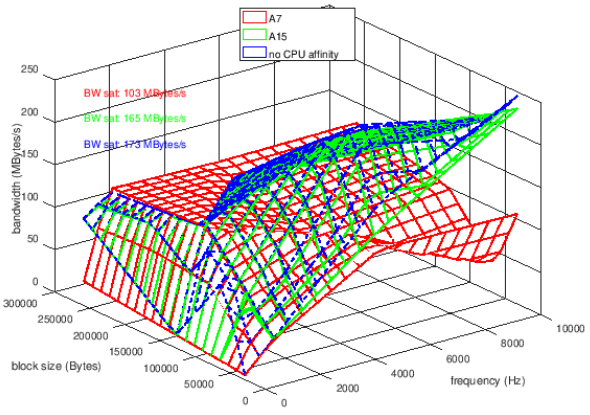


Fig. 7. TCPROS over WiFi saturation vs CPU affinity.

V. AADL Library for ROS

In this section, we explain how we organize the AADL model of our ROS applications, in order to enable bus load and bandwidth budget analysis.

A ROS application consists in a number of services running concurrently on possibly different hardware platforms. Every service is a node in the ROS terminology. A ROS distribution offers a large number of standard nodes for many of the common tasks a robot have to accomplish. To these nodes we add our own nodes, implementing the services we have to develop to fill our specific needs.

According to the component based modeling approach inherent to AADL, we have developed an AADL library of ROS components. To allow checking the validity of connections between components, data format have to be declared and associated to input or output ports of every component. Hence, our library includes a package dedicated to the declaration of every data format which may be transported as a ROS message on a ROS topic. Indeed, many different message formats are defined in the ROS middleware. This package uses the inheritance mechanism of the AADL language: from one high level data type, children are declared that extends the features and properties of the parent.

For example, the AADL model of our RGB image stream is thus:

```
data topic  end topic;
data implementation topic.impl  end topic.impl;

data sensor_msgs extends topic end sensor_msgs;
data implementation sensor_msgs.impl extends topic.impl

data Image extends sensor_msgs end Image;
data implementation Image.impl extends sensor_msgs.impl end
      Image.impl;

data implementation Image.rgb extends Image.impl
  properties
    Data_Size => 921 KByte;
  end Image.rgb;
```

As usual in AADL, the model of a component is divided into the declaration of the component type itself, and the declaration of one or several implementations of the component. Here we can observe that the size of the data is declared as a specific AADL property that will be used later by the bus load analysis tool included in OSATE2, namely Data_Size.

Now, this data implementation can be used in the declaration of input, output, or in/out ports of our components. As explained before, this, before anything, allows to check the validity of a connection between two components: obviously, a connection is only permitted between two ports that carry the same data type.

For instance, the usb_cam node will be declared as follows:

```
process usb_cam_nd extends node
  features
    rgb_stream_in: in event data port video_stream.rgb;
    rgb_image_raw_out: out event data port Image.rgb;
  end usb_cam_nd;
```

In order for the OSATE2 bus load analysis tool to work, we need to set, beside the size of data transmitted over a connection, the period at which this transmission occurs. This information is set with the period property of the thread in charge of sending the data over the connection. Let us continue with the standard ROS usb_cam node. The node implementation describes its subcomponents and inner connections:

connections between them, and from input or to output ports of the node.

```
process implementation usb_cam_nd.impl
  subcomponents
    image_broadcaster: thread imagePublisher.impl;
    usbSpinner: thread usbcam_spinner.impl;
  connections
    con1: port image_broadcaster.pub_msg ->
     rgb_image_raw_out;
    con2: port rgb_stream_in -> usbSpinner.rgb_stream_in;
end usb_cam_nd.impl;
```

Two threads are included in the usb_cam node. usbSpinner is a *spinner* thread which role is, in this node, to launch the publisher thread, named image_broadcaster, every time this is needed. For our USB camera working at 30 images/s, that is every 33 ms. This value is given to the Period property associated to the publisher thread implementation, imagePublisher.impl presented below. The connection con1 above is important since it allows to connect the node output rgb_image_raw_out to the thread output pub_msg which format is refined to data Image.rgb.

```
thread imagePublisher extends publisher
  features
    pub_msg: refined to out event data port Image.rgb;
end imagePublisher;

thread implementation imagePublisher.impl
  properties
    Period => 33333 us;—@ 30 images/s
end imagePublisher.impl;

thread implementation imagePublisher.xu4_a15 extends
    imagePublisher.impl
  properties
    Compute_execution_time => 2319 us .. 2319 us;
end imagePublisher.xu4_a15;
```

Another property is set here: Compute_execution_time. This property comes from a profiling of the MIPS demand of the running node over a specific CPU, here A15 cores in the Odroid XU4 board. We defined this property, along with processes MIPS budget and CPUs MIPS capacity, to allow for CPU load and MIPS budget analysis. This particular subject will be developed in an other paper.

Now, from the Data_Size and Period properties defined respectively in the data and in the thread components, the analysis tool is able to calculate the bandwidth demand for a publisher thread on the bus onto which its output connection is bound. In our example, that would be 921 KBytes$\times$ $^1/_{33333\mu s} = 27.63$ MBytes/s.

The next step is to compare this bandwidth demand with the bandwidth capacity of the bus. For this we need to add this specific property to the AADL model of our hardware platform, where buses are declared, and to the ROS library where a bus component will be added for every implementation of the TCPROS virtual bus. Hence, our ROS AADL package will include the following components, with the bandwidth capacities reported from the profiling work presented in section IV:

```
bus ros_bus end ros_bus;
bus implementation ros_bus.no_taskset extends ros_bus.impl
  properties
  SEI::BandWidthCapacity => 122.0 MBytesps;
  end ros_bus.no_taskset;

bus implementation ros_bus.A15 extends ros_bus.impl
  properties
    SEI::BandWidthCapacity => 166.0 MBytesps;
  end ros_bus.A15;
```

Only a part of the whole package is given here of course. In fact, because TCPROS connections capacity on Ethernet (and that is the same for WiFi), does not depends on the Ethernet device itself, but on the CPU capacities of the computer board, a software bus, called ros_ethernet (ros_wifi for WiFi) is included in the library:

```
bus ros_ethernet end ros_ethernet;
bus implementation ros_ethernet.impl end ros_ethernet.impl;

bus implementation ros_ethernet.no_taskset extends
     ros_ethernet.impl
  properties
    SEI::BandWidthCapacity => 253.0 MBytesps;
end ros_ethernet.no_taskset;
```

An AADL virtual bus component could have been used for those TCPROS buses, like in [15]. This however does not allow the use of the bus load analysis tools in OSATE2.

On the hardware side, bandwidth capacities of physical links have to be declared. Thus the AADL model of our Odroid XU4 board contains the following declarations (among other things):

```
system implementation Odroid_XU4.impl
  subcomponents
    Exynos_SOC: system Exynos_5422::Exynos_5422.impl;
    ethernet_bus: bus Ethernet::Ethernet.impl {SEI::
     BandWidthCapacity => 1000.0 MBytesps;};
    usb_bus_1: bus USB::USB.impl {SEI::BandWidthCapacity =>
      480.0 MBytesps;};
    usb_bus_2: bus USB::USB.impl {SEI::BandWidthCapacity =>
      4800.0 MBytesps;};
    usb_bus_3: bus USB::USB.impl {SEI::BandWidthCapacity =>
      4800.0 MBytesps;};
    hdmi_dev: device HDMI.impl;
  connections
    connection1: bus access ethernet -> ethernet_bus;
    connection2: bus access usb_1 -> usb_bus_1;
    connection3: bus access usb_3 -> usb_bus_3;
    connection4: bus access usb_bus_2 -> usb_2;
    connection5: port hdmi_dev.hdmi_port -> hdmi;
end Odroid_XU4.impl;
```

USB buses are used to connect the robot devices (here the RGB camera and the robot control board), to the Odroid XU4 board.

VI. Results: Bus load analysis from AADL

Using the AADL libraries that we have written for both the ROS middleware and the robot and the computer board it carries, we can build an AADL model of our complete application, including its deployment. Figure 1 is the graphical view of the application software, compliant with the *AADL diagram* meta-model, and is a direct translation of its AADL textual model, that we will not be presented here.

The application deployment is modeled as follows (instructions for binding processes and threads to CPUs are omitted):

```
system implementation remote_tracking_dep.impl
  subcomponents
    rem_trk_sw: system remote_tracking::remote_tracking_sw.
     impl;
    p3DX: system Pioneer3DX::Pioneer3DX.i;
    ROSbus: bus ros::ros_bus.impl;
    ROSEthernet: bus ros::ros_ethernet.impl;
  connections
    con1: port p3DX.video_rgb_pass -> rem_trk_sw.
     video_rgb_input;
    con2: port rem_trk_sw.cmdvel2hw -> p3DX.vel_cmd_in_pass
     ;
    con3: port p3DX.sonar_out_pass -> rem_trk_sw.hw2sonar;
  properties
```

```
    Actual_Connection_Binding => (reference (ROSbus))
     applies to rem_trk_sw.con6;-- usbcam->color_tracking
    Actual_Connection_Binding => (reference (ROSbus))
     applies to rem_trk_sw.con5;-- pos2cmd->rosaria
    Actual_Connection_Binding => (reference (ROSbus))
     applies to rem_trk_sw.con3;-- rosaria->sonaralert
    Actual_Connection_Binding => (reference (ROSbus))
     applies to rem_trk_sw.con2;-- sonaralert->pos2cmd
    Actual_Connection_Binding => (reference (ROSbus))
     applies to rem_trk_sw.con4;-- colortracking->pos2cmd
    Actual_Connection_Binding => (reference (ROSEthernet))
      applies to rem_trk_sw.con12;-- usbcam->imgview
    Actual_Connection_Binding => (reference (ROSEthernet))
      applies to rem_trk_sw.con10;-- joyteleop->rosaria
    Actual_Connection_Binding => (reference (p3DX.
     OdroidXU4.usb_bus_3)) applies to p3DX.con10;-- camera
     input on USB
    Actual_Connection_Binding => (reference (p3DX.
     OdroidXU4.usb_bus_3)) applies to rem_trk_sw.con8;--
     rosaria cmdvel on USB
    Actual_Connection_Binding => (reference (p3DX.
     OdroidXU4.usb_bus_3)) applies to p3DX.con11;--rosaria
     sonar on USB
end remote_tracking_dep.impl;

system implementation remote_tracking_dep.xu4 extends
     remote_tracking_dep.impl
  subcomponents
    rem_trk_sw: refined to system remote_tracking::
     remote_tracking_sw.xu4;
    ROSbus: refined to bus ros::ros_bus.no_taskset;
end remote_tracking_dep.xu4;
```

The binding of connections (software), to bus (hardware or pseudo hardware in case of TCPROS), is defined in the system implementation with Actual_Connection_Binding instructions.

The first five connection binding instructions, referencing bus ROSbus, are for binding connections labeled con6, 5, 3, 2, 4 in figure 1, to the ROS virtual bus. ROSbus is later refined to the ros::ros_bus.no_taskset bus component from our ROS package. The two next binding instructions are for remote connections via bus ROSEthernet. The three last instructions are for binding devices inputs or outputs connections to the physical USB bus p3DX.OdroidXU4.usb_usb_3.

With those bindings defined, an instance of our deployed system implementation is generated in OSATE. An instance is a view of the system that gathers all the components included in the different models that we have created to describe our system, software and hardware, from our library of ROS components. Figure 8 show the packages dependency graph for our complete application model.

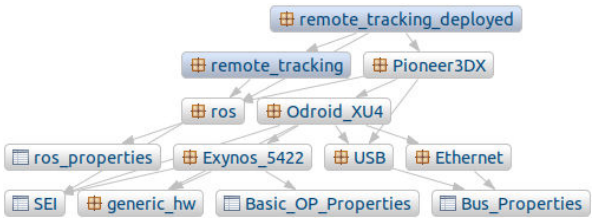


Fig. 8. Packages dependency graph.

From the instance, different analysis can be performed, depending on the properties that have been added to the models. The output of the OSATE2 bus load analysis tool is displayed table I. First, we get a list of all the buses found inside the system, with their capacity, budget, required budget, and actual load. The budget property is used whenever we want to allocate a limited part of the bandwidth capacity to a bus, virtual bus, or connection.

Then, we have the detail, for each bus, of the bandwidth demand per connection. A bus overload might be reported there. None are detected for any of the three buses we are using in our system. The USB bus comes first. Every connection bound to this bus is listed here. Those are connections to or from devices attached to the robot: the RGB camera, the robot's sonar belt, and the robot's control board. Messages from the sonar are relatively small: 220 Bytes are enough to transmit the measurements from the eight ultrasound sensors. They are transmitted every 100 ms. The generated bandwidth demand, 2.2 KBytes/s, is pretty small in front of the bandwidth requested by the RGB camera. Every velocity command sent to the robot is a 50 Bytes long message. Those commands are sent at 10 Hz so the bandwidth demand is 0.5 KBytes/s. This is again almost negligible in front of the RGB camera bandwidth demand.

In the situation where devices are at the source of messages, and to complete the preceding section, there is no process or thread to refer to, for finding the period at which the data is sent on the connection. The Period property must be placed in the device implementation declaration, like shown below for the RGB camera model, and the data size must be added to the data format declaration of the output stream of the device:

```
device camera_rgb
  features
    camera_usb: requires bus access USB::USB.impl;
    RGB_stream_out: out event data port ros::video_stream.
     rgb;
  end camera_rgb;

device implementation camera_rgb.impl
  properties
    Period => 33333 us;
end camera_rgb.impl;

.../...

data implementation video_stream.rgb
  properties
  --640x480x3=921600 Bytes #0.92MBytes
  Data_Size => 921 KByte;
end video_stream.rgb;
```

The two last sections in table I concern the ROS virtual bus, ROSbus, and the TCPROS connection over Ethernet, ROSEthernet. For each, the bound connections are listed with the bandwidth they are requesting from the bus. Again, the video stream from the camera, which travel both on the ROS virtual bus, from node usb_cam to ocv_color_tracking, and on the ROSEthernet bus, from usb_cam to image_view on the remote computer, is the major consumer.

VII. Conclusion

With the AADL model of our robotic application, built on top of our two main libraries: a library of components for ROS services and middleware, and a library of hardware platforms, including robots, devices, and computer boards, we are able to perform a complete analysis of the bandwidth demand from the application over the hardware resources. This analysis allows to check if the deployment scheme of our application meets its requirements. Overloaded buses are exhibited and

TABLE I
BUS LOAD ANALYSIS REPORT

```
"Physical Bus","Capacity (KB/s)","Budget (KB/s)","Required Budget (KB/s)","Actual (KB/s)"
"usb_bus_1","480000.0","0.0","0.0","0.0"
"usb_bus_2","4800000.0","0.0","0.0","0.0"
"usb_bus_3","4800000.0","0.0","0.0","27632.976302763025"
"ROSbus","122000.0","0.0","0.0","27633.034259960732"
"ROSEthernet","253000.0","0.0","0.0","27632.276302763024"

"Bus usb_bus_3 has data overhead of 0 bytes"
"Bound Virtual Bus/Connection","Capacity (KB/s)","Budget (KB/s)","Required Budget (KB/s)","Actual (KB/s)"
"p3DX.rgb_cam.RGB_stream_out -> rem_trk_sw.usbcam.usbSpinner.rgb_stream_in","","0.0","","27630.276302763024"
"p3DX.sonar.sonar_out -> rem_trk_sw.rosaria.hw_2_sonar","","0.0","","2.2"
"rem_trk_sw.rosaria.cmdvel_broadcaster.pub_msg -> p3DX.base_vel_cde.vel_cmd_in","","0.0","","0.5"

"Bus ROSbus has data overhead of 0 bytes"
"Bound Virtual Bus/Connection","Capacity (KB/s)","Budget (KB/s)","Required Budget (KB/s)","Actual (KB/s)"
"sonar_alert.alert_broadcaster.pub_msg -> pos_to_cmd.disable_in","","0.0","","0.0015015015015015015"
"rosaria.cloud_broadcaster.pub_msg -> sonar_alert.sensors_in","","0.0","","2.2"
"usbcam.image_broadcaster.pub_msg -> ocv_color_tracking.ctSubscriber.subs_msg","","0.0","","27630.276302763024"
"ocv_color_tracking.ctPublisher.pub_msg -> pos_to_cmd.pos_cmd_sub.subs_msg","","0.0","","0.24"
"pos_to_cmd.cmd_vel_pub.pub_msg -> rosaria.cmd_vel_in","","0.0","","0.31645569620253167"

"Bus ROSEthernet has data overhead of 0 bytes"
"Bound Virtual Bus/Connection","Capacity (KB/s)","Budget (KB/s)","Required Budget (KB/s)","Actual (KB/s)"
"joyteleop.teleopPublisher.pub_msg -> rosaria.cmd_vel_in","","0.0","","2.0"
"usbcam.image_broadcaster.pub_msg -> imgview.image_raw_in","","0.0","","27630.276302763024"
```

different options may be investigated, with no need for having the real hardware at hand.

In parallel of this work, we have also enrich our models with AADL properties that allows to check if the MIPS demand from every node in the application is inside the MIPS capacity of the CPU, or cluster of CPUs, the node is bound to. Again, a standard tool from OSATE2, namely, the "budget analyze resource allocations" tool, is used. With BUS load analysis and CPU load analysis, we can really explore many options for placing our ROS nodes on the hardware architecture. For instance, a node exhibiting high CPU load with low output bandwidth, might be bound on an additional computer board with no performance penalty. The node ocv_color_tracking from our application has this kind of profile. The main board would be relieved of this heavy task and have more time for dealing with the remaining processes to maybe reach real time constraints in some other situation.

Design space exploration for hardware and software architecturing is what we need for our robotic applications, and that is what we get with the set of libraries, models, and usage of tools we are proposing. But we also wanted something fast and easy to set up, hence the choice for a simple profiling approach, using standard on-the-shelf tools, for keeping models as simple as possible, for adding only properties and details that are necessary for the analysis we need to carry on.

We are currently pursuing the development of our libraries with more components both from ROS constellation of nodes and hardware platforms. Our libraries will be make available to the public in the near future.

REFERENCES

[1] ROS, powering the world's robots. [Online]. Available: https://www.ros.org/

[2] P. H. Feiler, B. A. Lewis, and S. Vestal, *The SAE Architecture Analysis & Design Language (AADL) a standard for engineering performance critical systems*, 2006, pp. 1206–1211.

[3] Osate 2.9.1 documentation. [Online]. Available: https://osate.org/

[4] X. Ma, F. Fang, K. Qian, and C. Liang, "Networked robot systems for indoor service enhanced via ROS middleware," in *2018 13th IEEE Conference on Industrial Electronics and Applications (ICIEA)*, 2018, pp. 852–857.

[5] F. de Assis Moura Pimentel and P. T. Aquino-Jr, "Performance evaluation of ROS local trajectory planning algorithms to social navigation," in *2019 Latin American Robotics Symposium (LARS), 2019 Brazilian Symposium on Robotics (SBR) and 2019 Workshop on Robotics in Education (WRE)*, 2019.

[6] D. Bore, A. Rana, N. Kolhare, and U. Shinde, "Automated guided vehicle using robot operating systems," in *2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI)*, 2019, pp. 819–822.

[7] S. Gatesichapakorn, J. Takamatsu, and M. Ruchanurucks, "ROS based autonomous mobile robot navigation using 2d lidar and rgb-d camera," in *2019 First International Symposium on Instrumentation, Control, Artificial Intelligence, and Robotics (ICA-SYMP)*, 2019, pp. 151–154.

[8] Y. Abdelrasoul, A. B. S. H. Saman, and P. Sebastian, "A quantitative study of tuning ROS gmapping parameters and their effect on performing indoor 2d slam," in *2016 2nd IEEE International Symposium on Robotics and Manufacturing Automation (ROMA)*, 2016, pp. 1–6.

[9] J. Cano, A. Bordallo, V. Nagarajan, S. Ramamoorthy, and S. Vijayakumar, "Automatic configuration of ROS applications for near-optimal performance," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016, pp. 2217–2223.

[10] F. Xiaorong, X. Jing, L. Wei, and Y. Panfe, "A multi-level grey performance evaluation model for robot operating system," in *2nd International Conference on Safety Produce Informatization (IICSPI)*, 2019.

[11] P. Feiler and D. Gluch, *Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis & Design Language*. Addison-Wesley, 2012.

[12] S. Rubini, P. Dissaux, and F. Singhoff, "Modeling shared-memory multiprocessor systems with AADL," in *1st Architecture Centric Virtual Integration (ACVI) Workshop. In conjonction with the MODELS international conference*, September 2014.

[13] G. Bardaro and M. Matteucci, "Using AADL to model and develop ROS-based robotic application," in *2017 First IEEE International Conference on Robotic Computing (IRC)*, 2017, pp. 204–207.

[14] G. Bardaro, A. Semprebon, A. Chiatti, and M. Matteucci, "From models to software through automatic transformations: An AADL to ROS end-to-end toolchain," in *2019 Third IEEE International Conference on Robotic Computing (IRC)*, 2019, pp. 580–585.

[15] M. Larsen, "Modelling field robot software using AADL," Electrical and Computer Engineering Technical report ECE-TR-25, april 2016.

STPA Analysis of Automotive Safety Using Arcadia and Capella

David Hetherington
President Asatte Press, Inc
david_hetherington@ieee.org

Pascal Roques
Independent Author, Trainer, and Consultant at PRFC
pascal.roques@prfc.fr

Abstract

This paper demonstrates the use of the Arcadia methodology and the open source Capella tool to implement a STPA-based analysis technique that augments the conventional HARA, HAZOP. The STPA approach extends the conventional methods to include a holistic perspective considering hardware, software, humans, and control failures in a balanced manner.

Introduction

As embedded software becomes an ever-increasing percentage of the value of an automobile, functional safety and cybersecurity are becoming dominant concerns in the design of both the automotive embedded electronics and the embedded software that runs on that hardware. However, both topics are exceptionally challenging in an automotive embedded software environment.

Current safety methodologies have evolved over the last 30-40 years from a set of practices originally intended for chemical plants.

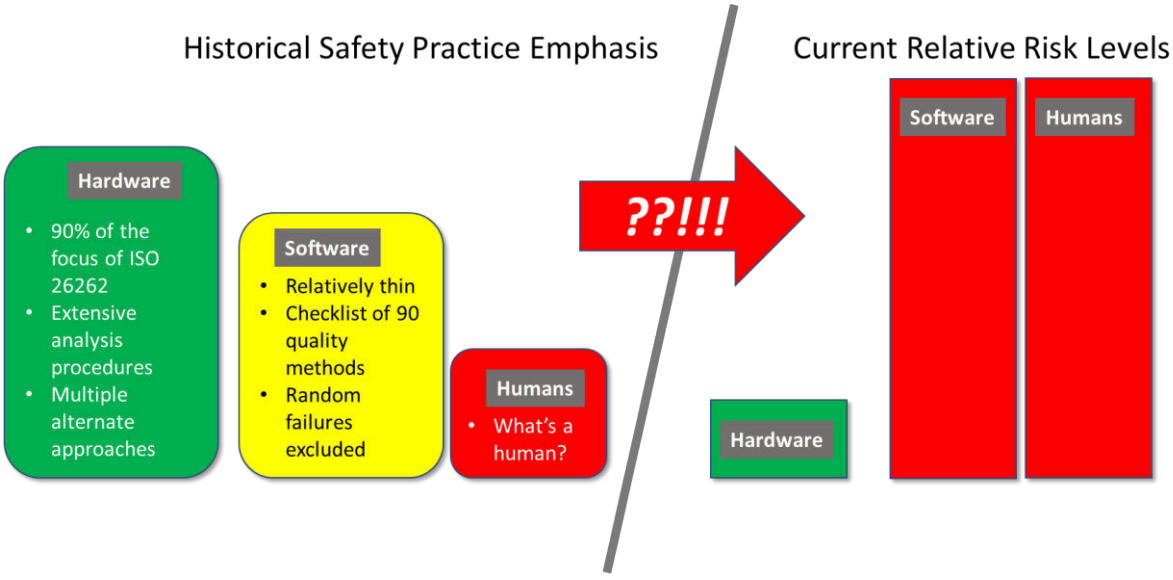


Figure 1 – Historical safety approaches not well-matched to current challenges

IEC 61508 had its origins in industrial plant safety. At the time, the primary concern with cascading hardware failures. This early focus has tended to shape the perspective of other standards such as ISO 26262 that are descended from the original versions of IEC 61508.

The 2018 version of ISO 26262 consists of 12 parts totally 808 pages. Of these, software makes up one part (Part 6) which at 66 pages comprises 8.2% of the standard. For comparison, parts 5 and 11 are completely

about hardware (288 pages). Parts 8, 9, and 10 are nominally applicable to both hardware and software, but a close examination of the detailed recommendations in these parts (For example Part 9, 7.4.4) reveals an almost complete focus on hardware as well. (Another 200 pages).

As for software, part 6 itself consists of 15 tables listing well-established quality techniques in bullet form.

Table 2 — Notations for software architectural design

Notations		ASIL			
		A	B	C	D
1a	Natural language[a]	++	++	++	++
1b	Informal notations	++	++	+	+
1c	Semi-formal notations[b]	+	+	++	++
1d	Formal notations	+	+	+	+

[a] Natural language can complement the use of notations for example where some topics are more readily expressed in natural language or providing explanation and rationale for decisions captured in the notation.

[b] Semi-formal notations can include pseudocode or modelling with UML®, SysML®, Simulink® or Stateflow®.

NOTE UML®, SysML®, Simulink® and Stateflow® are examples of suitable products available commercially. This information is given for the convenience of users of this document and does not constitute an endorsement by ISO of these products.

Figure 2 – Example table from ISO 26262 Part 6 (2018)

Figure 2 above is an example of the thin nature of the standard when it comes to software. The entire topic of model-based engineering is reduced to a single bullet: “semi-formal notations”. Model-based engineering is a huge topic, and the standard provides no depth of thinking about what one might be trying to accomplish with the use of these techniques. Although some work was done between the 2011 and 2018 versions of the standard to improve part 6, the recommendations are still weak when it comes to specific methods for dealing with emergent behavior in software system- of-systems. Interactions with humans are not considered at all. Artificial Intelligence and other types of software with behavior that is not deterministic by design is out of scope in ISO 26262.[1]

The main difficulty with classical automotive safety techniques is that they have become somewhat of a victim of their own success. The hardware reliability of automotive components has improved by orders of magnitude since the 1980s when the industry started thinking in earnest about the problem. On the other hand, the amount and complexity of the software in and around the vehicle has exploded. Even without artificial intelligence, vehicles are already streaming data into the cloud and downloading software updates from the cloud. The complexity of systems like the infotainment system one would find in a current competitive minivan are far beyond the wildest imagination of the 1980s engineers who laid the groundwork for current safety practices. Overwhelmed drivers are a real problem. Software that is so complex that it exhibits what might as well be random failures is also a problem. Our automotive functional safety processes are balanced to fit the challenges of the 1980s, not the challenges of the 2020s.

The “System-Theoretic Process Analysis” or “STPA” hazard analysis technique addresses many of the weaknesses listed above. During the hazard analysis process, STPA looks at control loops within the system. In the STPA technique, hazards are posed by unsafe control actions. This technique is quite helpful in that any of the elements in the loop can be hardware, software, or human. For example, if the controlled process is keeping a car in its lane on the highway, the “process model” that might fail could be the driver’s perception of where the lane is. With the STPA technique, we have an analysis approach that can more evenly and uniformly consider software and humans in the loop. Even better, we do *not* have to assume that the software or the humans are “deterministic” for the analysis technique to work.

The first question is what sort of tool, if any, we should use for STPA. Many instructors of STPA apparently discourage the use of modeling tools, perhaps out of fear that the modeling tool will introduce some sort of tunnel vision regarding the system.

[1] The ISO 26262 community has developed the separate SOTIF standard to address this gap. See [3]

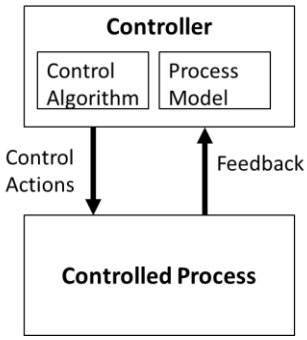


Figure 3 – Generic control loop [2]

While that concern cannot be discounted, an actual safety analysis at the scale needed for a commercial vehicle is simply not feasible without tooling.

By the time a vehicle first rolls off of the assembly line, the vehicle maker and its multiple tiers of suppliers will have worked their way through hundreds of thousands of context elements, system elements, hazards, failures, effects, and other related information. This scale of analysis simply cannot be done with paper and clipboard or even with spreadsheets. Intelligent modeling tools are mandatory. It is not feasible to design a current generation commercial automobile without them.

A number of suppliers make purpose-built safety analysis tools. Many of these purpose-built safety analysis tools include "SysML-like" features that are tailored for safety analysis work. Some tools also claim support for STPA. Unfortunately, neither author has current license for any of these tools. As such, the authors are not in a position to evaluate these sorts of purpose-built tools and they are out of scope for this paper.

That leaves model-based systems engineering (MBSE) tools as candidates for conquering the complexity that would be involved in a full-scale STPA analysis of an entire vehicle or a large subsystem within a vehicle. Within the field of MBSE tools, SysML tools are obvious candidates for performing this sort of analysis and many companies do at least some parts of their safety architecture work using SysML tools. Both authors are quite familiar with SysML tools. In fact, David Hetherington publishes beginner books for SysML tools and uses SysML tools regularly for functional safety modeling.

Recently, however, the authors have begun collaborating on a beginner book for the Capella tool and the Arcadia method. During this work, the authors noticed that Arcadia has some special characteristics that are well-suited for functional safety work and STPA in particular. This rest of this paper will demonstrate an approach to using the open source Capella tool and the Arcadia methodology[3] to perform the STPA analysis. The specific MBSE features of the tool and method that are convenient for STPA analysis will be highlighted.

Getting to the Starting Line

There are a few things we need to do to get to the starting line to use the analysis technique laid out in the STPA Handbook.[4]

System of Interest

Our system of interest is the "Bold Truck" Electric Sport Utility Vehicle.

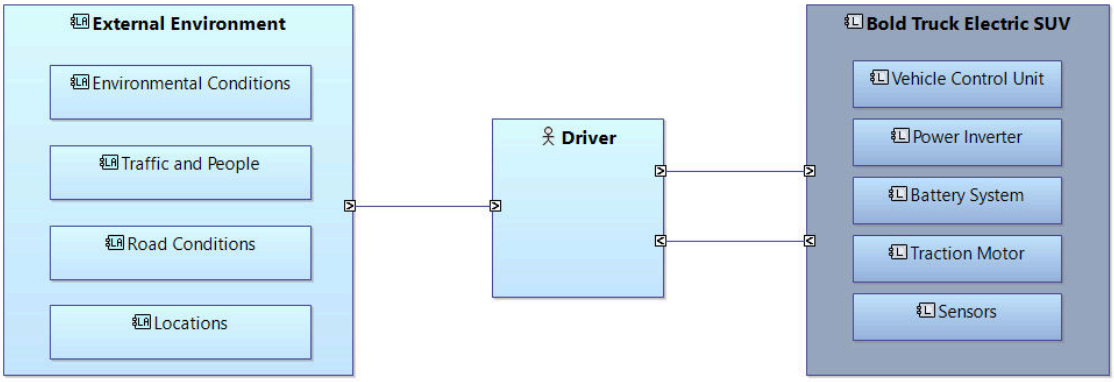


Figure 4 – The Bold Truck electric sport utility vehicle

[2] See [2] Adapted from STPA Handbook Figure 2.6 on page 23
[3] See **Erreur ! Source du renvoi introuvable.**
[4] See [2] to download the STPA Handbook.

Figure 4 shows the truck in its context. Of course, the truck would have a lot of other subsystems. For this paper, however, we are going to focus on a "narrow slice" to explore how STPA would be used to model safety hazards in the control of the electric motor.[5]

Modeling Setup

For this paper, we used Capella 5.1. We also installed the requirements add-on. [6]

Analysis Procedure

For the context analysis, we will follow the definition of "TypicalAutomotiveSituation" from the OMG Risk Analysis Modeling Language (RAML) 1.0 Beta specification[7] with the modification that we will replace the word "Typical" with the word "Valid", the word "Typical" being a little too open-ended for our purposes.

In order to keep the size of this paper manageable, we will focus on just one valid situation.

Name	Vehicle Usage	Traffic and People	Road Condition	Location	Environmental Condition
Freeway	Driving forward at >100 km/hr	Light traffic. Nearest car is 15 seconds away.	Clean, dry, asphalt	Public high-speed highway	Warm, sunny, dry, normal humidity

Table 1 – Valid automotive situation

A real-life analysis of an entire vehicle would start with a large number of such situations, perhaps 100 or more. For example, the control actions and hazards for backing out of a driveway would be quite different from those for driving on a freeway. Likewise, driving in snow or rain would present different control behaviors than driving in nice weather.

The first thing we will model is the Freeway valid automotive situation.

Below is a specific Arcadia diagram called "Contextual System Actors", modeled at "System Analysis" level. The Bold Truck Electric SUV is considered as a "black box", and all external entities are called "Actors" (as in UML and SysML). We used the "constraint" concept, still as in UML and SysML and noted with {c}, to model the qualifying scope constraints of the valid automotive situation.

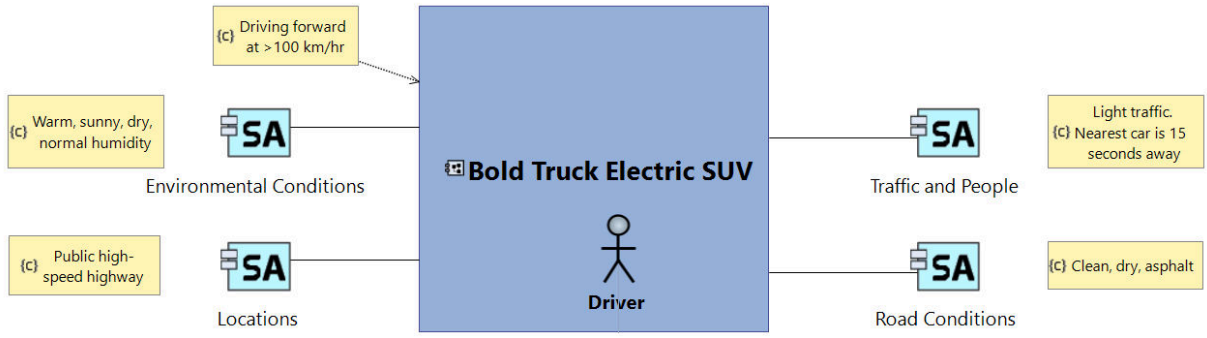


Figure 5 – The Freeway valid automotive situation

With the context defined, we can proceed with the steps laid out in the STPA Handbook.

STPA Step 1: Define the Purpose of the Analysis

Let us start now with the first STPA step: "Define Purpose of the Analysis".

[5] See [5] for an excellent discussion of the functionality and safety concerns for such a power inverter.
[6] See [7] for download of Capella and also the requirements add-on
[7] See [4] Figure 9.124 - TypicalAutomotiveSituation

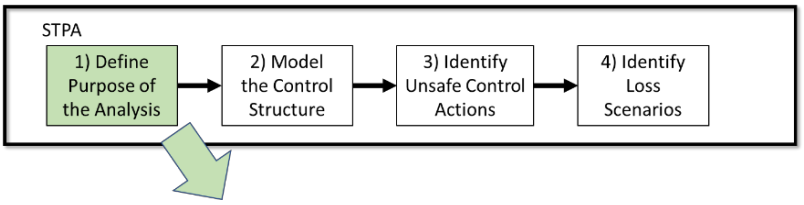


Figure 6 – STPA Step 1: Define the purpose of the analysis[8]

The first step consists of four parts:

1. Identify losses
2. Identify system-level hazards
3. Identify system-level constraints
4. Refine hazards (optional)

Identify Losses

In the first step, we need to identify the potential losses at the system level.[9]

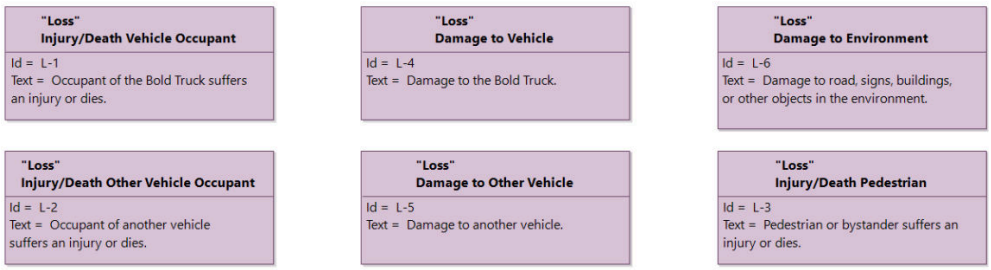


Figure 7 – Freeway: identify losses

In order to capture the losses in the model, we have created a new requirement type, using the Capella "Requirements viewpoint" add-on. For the Ids we follow the convention shown in the STPA Handbook.

Identify System-level Hazards

The next step is to identify the system-level hazards and tie them to losses. A key point here is that the methodology and the tool can help, but it is ultimately the humans who identify the hazards. The methodology and the tool merely provide a framework to stimulate productive thinking and help keep track of the hazards identified by the humans.

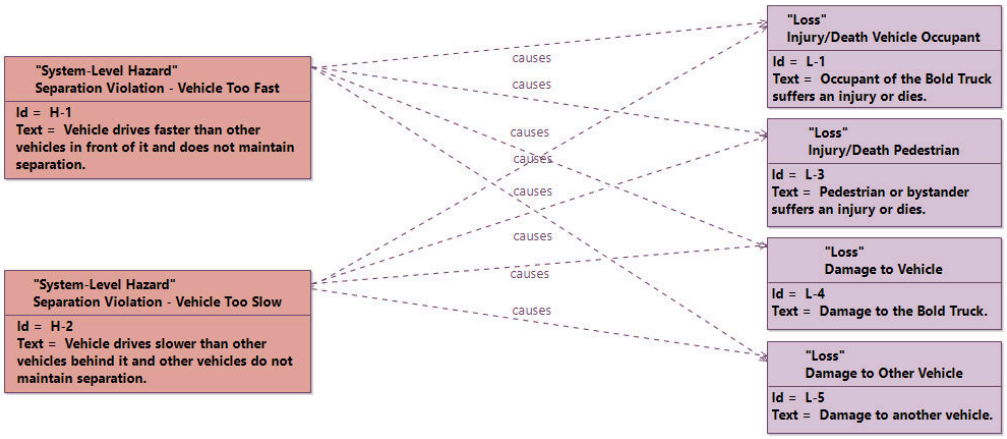


Figure 8 – Freeway: identify system-level hazards

[8] See [2] Adapted from STPA Handbook Figure 2.2 on page 15
[9] See [2] page 16 for the formal definition of a loss.

Here we create a “system-level hazard” type and the “causes” relationship. As it turns out, all of the hazards that we have identified can cause all of the losses. That is a coincidence and would not be the case in general.

Identify System-level Constraints

Here the goal is to identify constraints that will prevent or at least mitigate the hazards identified in the previous step and thereby prevent the losses from occurring.

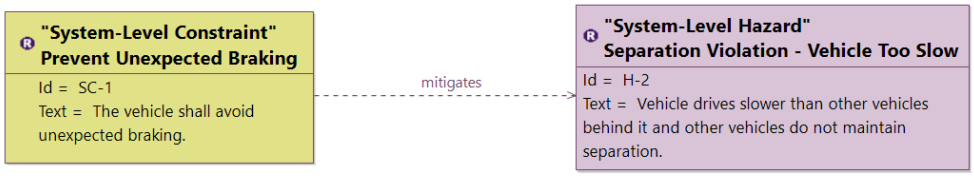


Figure 9 – Freeway: System-level constraints

In the interest of brevity, we have shown only one system-level constraint here.

STPA Step 2: Model the Control Structure

Now we are ready to proceed to the second STPA step: “Model the Control Structure”.

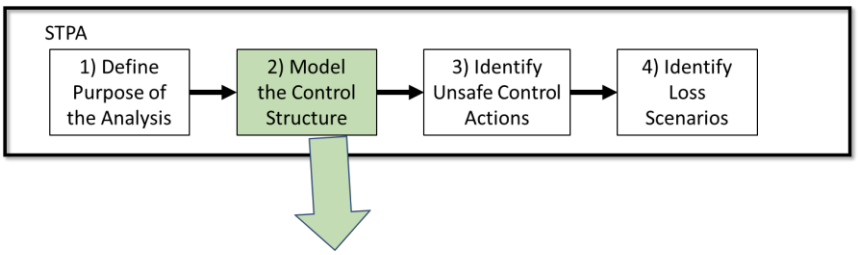


Figure 10 – STPA Step 2: Model the control structure[10]

The first level of control loop is between the driver and the vehicle with input from the weather and scenery.

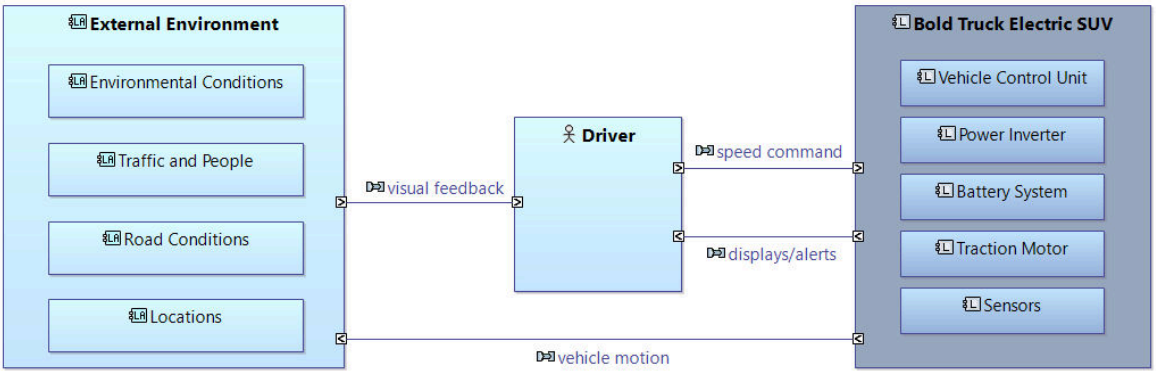


Figure 11 – Control loop: driver, vehicle, and weather/scenery

This is our first option for an Arcadia diagram to show a control loop. We used a “System Architecture Blank” diagram in Capella, which is similar to a SysML internal block diagram (*ibd*). The light blue rectangles represent external actors, as in Figure 4. Notice that small arrow icons inside the ports indicate the direction of flow between the structural blocks (system / actors). As the vehicle moves, the changing position of the vehicle in the environment causes changing visual feedback to the driver. The visual feedback from the environment as well as potential alerts from the vehicle itself cause the driver to take action to increase, decrease, or maintain speed as needed.

[10] See [2] Adapted from STPA Handbook Figure 2.5 on page 22

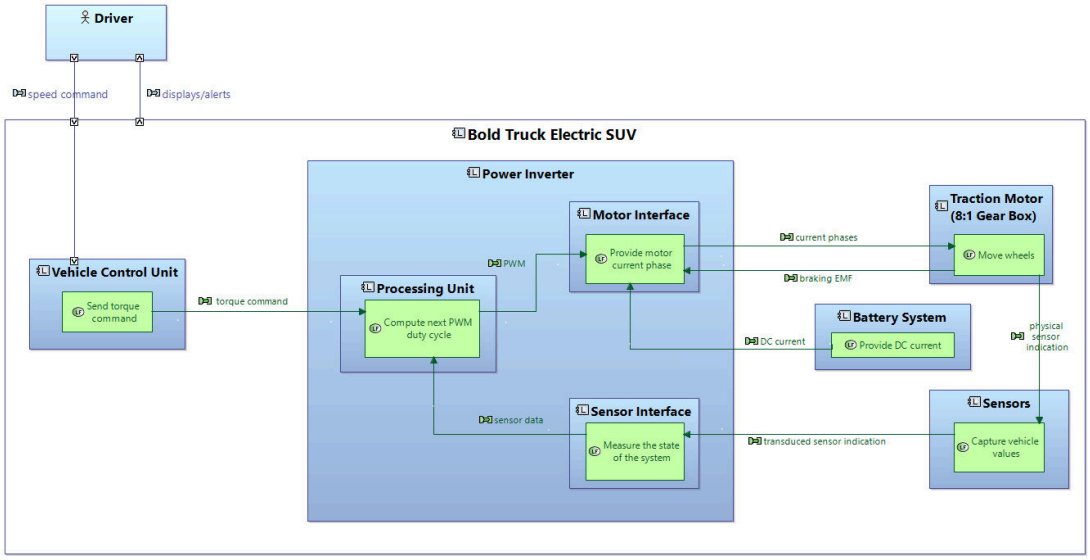


Figure 12 – Level 2: Power inverter internal control loop with Arcadia functions and functional exchanges

Within the vehicle, the power inverter controls the motors. The main subsystems involved, as shown in Figure 4, are now connected by means of functions and functional exchanges. The diagram we used is a "Logical Architecture Blank" diagram from Arcadia. This type of diagram allows us to represent not only logical components inside the system but also the functions allocated to the components. We can see the control loop just by following the sequence of functional exchanges: "torque command" going into the Processing Unit of the Power Inverter, then the outgoing pulse width modulation signal ("PWM"), which is used to create positive and negative three-phase alternating current ("current phases") to feed the motor. Coming back, current, temperature, and other physical properties ("physical sensor indication") are monitored by sensors that transduce the physical phenomena into electrical signals. Back inside the power inverter, these sensor electrical signals are transformed into meaningful digital data for use by the processing unit.

Arcadia also has a very useful concept called "Functional Chain" (which is missing from SysML). A functional chain is an ordered set of references to functions and the functional exchanges that link them, describing one possible path among all the paths forming the dataflow. Here we modeled the control loop as a specific functional chain. The functional chain is a model element itself, which means we will be able to assign non-functional properties such as requirements directly to the functional chain.

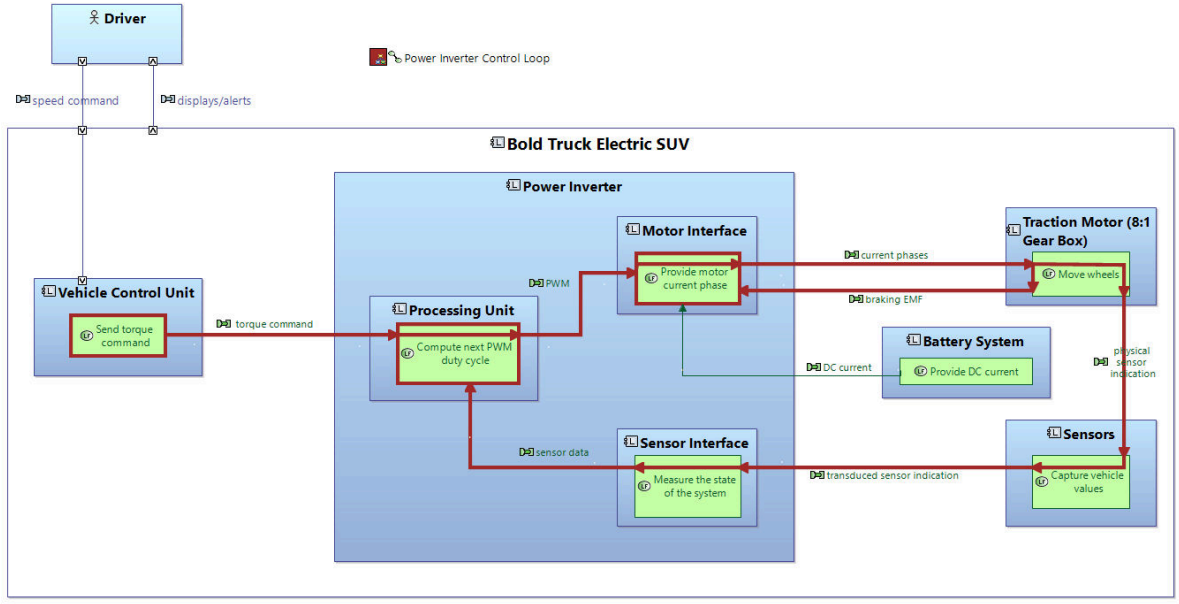


Figure 13 – Level 2: Power inverter control loop with Arcadia functional chain

For safety analysis (STPA or conventional methods like dependent failure analysis) the Arcadia functional chain is really useful. Essentially, the functional chain allows the safety engineer to depict a use case's flow through the system while showing both information moving between components/functions as well as information being processed within the components/functions. This duality is important because safety-related failures can occur both within the components/functions and also in the path between functions/components.

SysML can partially represent a similar concept, but the SysML internal block diagram does not allow the modeler to depict an item flow *through* a component, making it difficult to unambiguously show the sequential path. SysML sequence diagrams can also be used to some extent, but it is cumbersome to show a path failure inside a lifeline. In SysML it is also difficult to link a failure directly to an item flow or to a message in a sequence diagram. As we will see below, this sort of linking is quite easy in Arcadia.

STPA Step 3: Identify Unsafe Control Actions

Now we are ready to proceed to the third STPA step: "Identify Unsafe Control Actions".

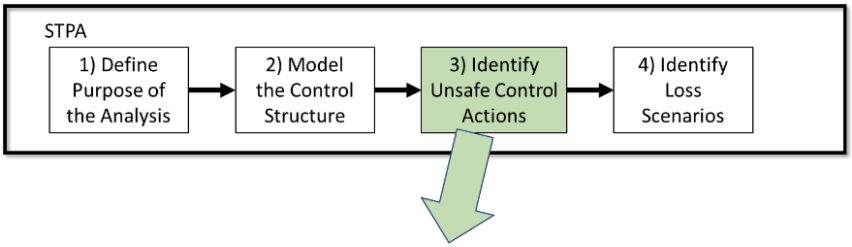


Figure 14 – STPA Step 3: Identify Unsafe Control Actions[11]

An Unsafe Control Action (UCA) is a control action that, in a particular context and worst-case environment, will lead to a hazard. Using Capella and the "Requirements Viewpoint" add-on, we will once again create a new requirement type for unsafe control actions. We can create specialized "UCA" requirements and link them with any Arcadia concept, such as a single link on a functional chain.

Looking at the functional chain shown in Figure 13 on page 7, when the processing unit sends the pulse width modulated signal (PWM) to the motor interface, this flow can be considered to be a control action "Provide PWM Signal". Using the standard STPA questions, this control action can be mirrored with four potential unsafe control actions.[12]

Control Action	Not Providing Causes Hazard	Providing Causes Hazard	Too Early, Too Late, Out of Order	Stopped Too Soon, Applied Too Long
Provide PWM Signal	UCA 1 - PWM signal not provided	UCA 2 - PWM signal provided erroneously	UCA 3 - PWM signal provided prematurely	UCA 4 - PWM signal halted prematurely UCA 5 - PWM signal provided after vehicle stopped

Table 2 - Potential unsafe control actions for Provide PWM Signal

[11] See [2] Adapted from STPA Handbook Figure 2.14 on page 35
[12] See [2] STPA Handbook Table 2.3 on page 36

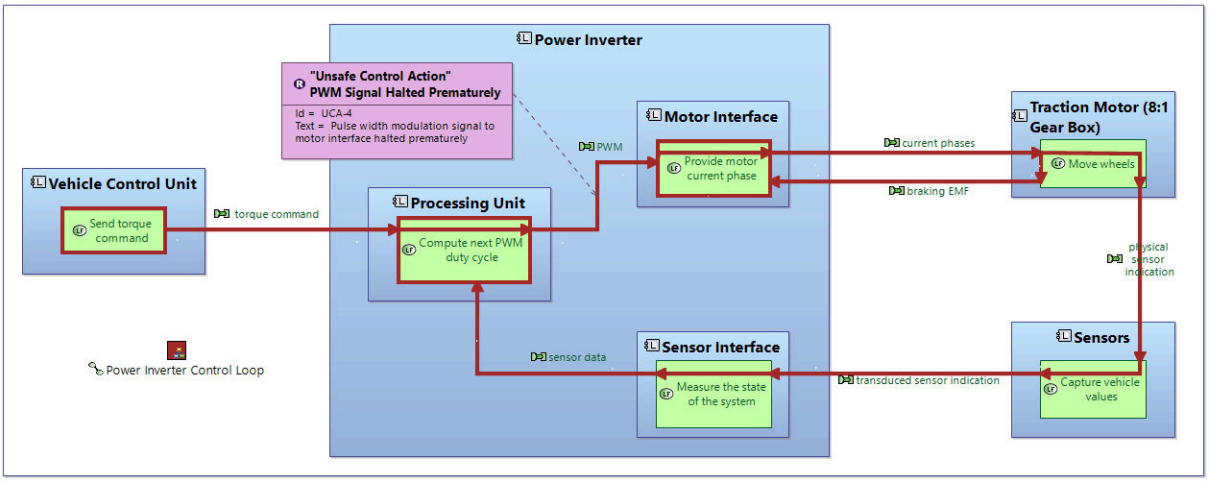


Figure 15 – Level 2: UCA linked to Arcadia functional chain

Once identified, the unsafe control action can be linked back to a specific hazard that it causes in a manner similar to the linking of the system constraints to hazards as shown in Figure 9 on page 6. In the interest of brevity, we have not provided a diagram for this step.

STPA Step 4: Identify Loss Scenarios

Now we are ready to proceed to the fourth STPA step: "Identify Loss Scenarios".

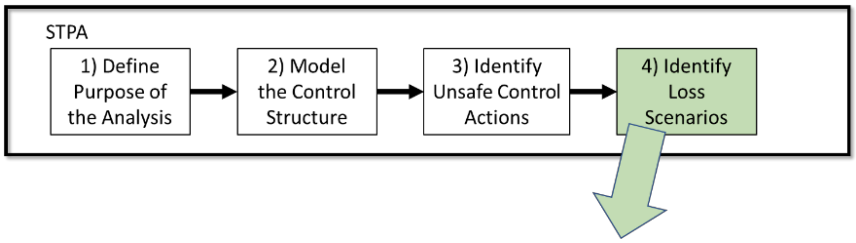


Figure 16 – STPA Step 4: Identify loss scenarios[13]

Loss scenarios are the final step in the STPA analysis technique. This step is where the cause of the hazard comes together with the resulting unsafe control action to cause the hazard. Again, we can create a specialized Arcadia requirement type for the STPA scenario.

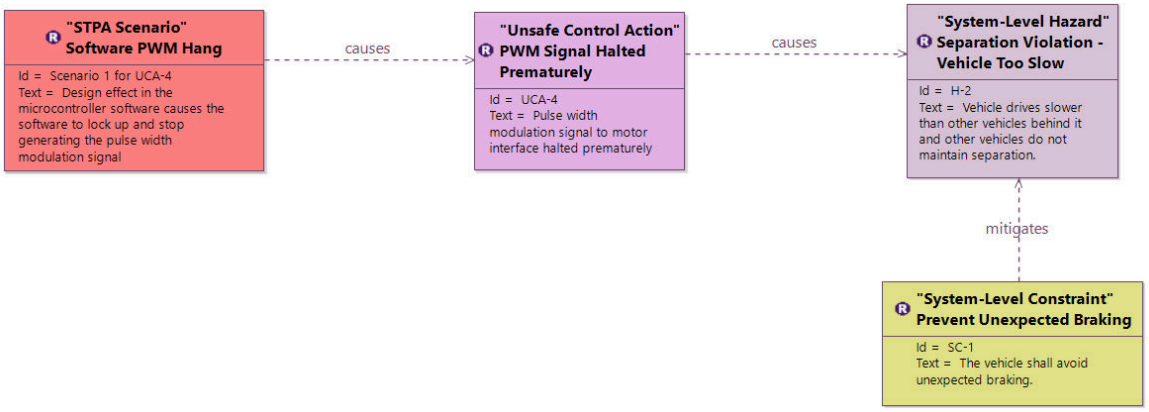


Figure 17 – Failure causes unsafe control action and hence hazard

[13] See [2] Adapted from STPA Handbook Figure 2.16 on page 42

In the example loss scenario shown in Figure 17, a bug in the processor software causes the software to hang, interrupting the PWM signal and causing the hazard that the vehicle brakes suddenly and is too slow relative to the traffic behind it.

Notice that the using the STPA analysis techniques and Arcadia functional chains, only the links between functions are candidates for unsafe control actions. The links that go through a function are the potential location of failures which become scenarios that cause the unsafe control action on the output link.

Discussion

While this example was necessarily brief, the Arcadia method, the Capella tool, and the STPA Hazard analysis technique can all contribute to a robust automotive safety program. Although STPA does not fully replace all other hazard analysis techniques, it can be used as an additional technique to cover weaknesses in the more conventional techniques regarding human factors as well as unexpected emergent behaviors caused by the complex interaction of a large number of software components sourced from different suppliers interacting with each other and less stable environments such as the internet.

STPA can be implemented using SysML tools or specialized safety analysis tools. However, the Arcadia method and Capella tool offer some convenient features that ease the analysis. In particular, functional chains are a very good fit for the fundamental control loop approach of STPA. More subtly, specialized requirement types can be created using the requirements add-on to represent the different STPA concepts like unsafe control actions and these can be placed on Arcadia diagrams and tied directly to specific links in a functional chain. While it is easy enough in SysML to create specialized requirement types, it is quite difficult or impossible in SysML to connect these directly to item flows, elements in a sequence diagram, or elements in an activity diagram.

Follow-Up

The techniques in this paper show promise, but we have not presented a full solution that would be ready for commercial deployment. As described in the introduction of [2], STPA is a hazard analysis technique. While hazard analysis is a crucial first step in a functional safety process, there is a lot more to functional safety than just identifying the hazards. A more complete implementation of the Arcadia method demonstrated in this paper would integrate the STPA method shown into a full, end-to-end automotive safety process.

Both authors will be delighted to answer follow-up questions on this paper and the Arcadia technique presented. If you would like a copy of the Arcadia model used in this paper, and created with Capella 5.1, please send an e-mail to either or both authors.

References

[1] ISO 26262 - https://en.wikipedia.org/wiki/ISO_26262

[2] STPA Handbook - https://psas.scripts.mit.edu/home/get_file.php?name=STPA_handbook.pdf

[3] SOFTIF - ISO/PAS 21448:2019 Road vehicles — Safety of the intended functionality https://www.iso.org/standard/70939.html

[4] RAAML - https://www.omg.org/spec/RAAML/

[5] Safety Concept Overview Of High-Voltage (HV) Traction Inverter
Antoine Dubois, Erik Santiago, Sandeep Kumar (NXP)
https://www.avnet.com/wps/wcm/connect/onesite/c7840908-f6bd-4df7-a07e-dd380d7f9bdc/nxp-SAFETY-CONCEPT-OVERVIEW-HV-TRACTION-INVERTER.pdf?MOD=AJPERES&CVID=nHyIEM9&CVID=nHyIEM9&CVID=nHyIEM9&CVID=nHyIEM9&CVID=nHyIEM9&CVID=nHyIEM9&CVID=nHyIEM9&CVID=nHyIEM9&CVID=nHyIEM9

[6] Arcadia Datasheet: https://www.eclipse.org/capella/resources/Datasheet_Arcadia.pdf

[7] Capella Download web site: https://www.eclipse.org/capella/download.html

Session We.3.C

Formal Methods

Wednesday 1st June

15:00

–

Room Pastel

Static Data and Control Coupling Analysis

Daniel Kästner, Laurent Mauborgne, Stephan Wilhelm, Christoph Mallon, Christian Ferdinand
AbsInt Angewandte Informatik GmbH. Science Park 1, D-66123 Saarbrücken, Germany

Abstract

All current safety norms require determining the data and control flow in the source code and making sure that it is compliant to the intended control and data flow as defined in the software architecture. In traditional static code analysis, data accesses via pointer variables and control flow by function pointer calls might be missed. Using sound static analysis based on abstract interpretation, it is possible to guarantee the absence of run-time errors that could cause memory corruption and control flow corruption. Furthermore, it is possible to guarantee that in the analysis, all data and function pointer targets are considered and that the possible data and control coupling is fully captured. In this article we propose a comprehensive methodology for statically computing a safe approximation of the data and control coupling between software components. Our approach incorporates global static data and control flow analysis, taint analysis and program slicing. It can detect critical data and control flow errors and allows to complement traditional code coverage criteria by the degree of data and control coupling covered by the testing process, helping to identify relevant previously untested scenarios. It can also demonstrate freedom of spacial interference between software components at the source code level.

Keywords: data coupling, control coupling, DO-178C, static analysis, taint analysis, program slicing, abstract interpretation, interference analysis, software architecture

1 Introduction

A failure of a safety-critical system may cause high costs or even endanger human beings. With the unbroken trend towards growing software size in embedded systems more and more safety-critical functionality is implemented in software. Preventing software-induced system failures becomes an increasingly important task. Contemporary safety norms from all industry domains – including DO-178B, DO-178C, ISO-26262, IEC- 61508, the FDA Principles of Software Validation, IEC62304, and EN-50128 – require to identify potential hazards and to demonstrate that the software does not violate the relevant safety goals.

Functional safety implies demonstrating the functional correctness: the functional software requirements have to be satisfied. Demonstrating functional correctness can be addressed by requirements-based testing, in particular automatic and model-based testing, and by formal methods such as model checking or theorem proving. In addition, safety-relevant quality requirements, the so-called non-functional requirements, have to be addressed. Examples of safety-relevant non-functional software requirements are adherence to resource bounds, especially worst-case execution time bounds and stack size, as well as freedom of run-time errors. Runtime errors are typically caused by undefined or unspecified behaviors of the programming language used. In the case of the programming language C they include faulty pointer manipulations, numerical errors such as arithmetic overflows and division by zero, as well as data races, deadlocks, and further synchronization errors in concurrent software.

The data and control flow of the software is of crucial importance for the verification of functional and non-functional correctness properties. In the DO-178C, the verification goal 6.3.3.b (Consistency) demands that "a correct relationship exists between the components of the software architecture. This relationship exists via data flow and control flow." It is complemented by the verification goal of Sec. 6.3.4.b (Compliance with the software architecture) which demands to "ensure that the source code matches the data flow and control flow defined in the software architecture". Obviously the data and control flow of the implemented software must match the intended data and control flow as specified in the software architecture, and unintended data and control flow must be avoided. This also implies demonstrating freedom of interference between software components in mixed-criticality software. According to DO-178C, Sec 2.4.1, if partitioning and independence between software components cannot be demonstrated, all of them are subject to the highest criticality level assigned to any of them. Similar requirements can also be found in ISO-26262 (cf. Sec. 7.4.9 and Annex D of [8]) and other safety norms.

Furthermore, the data and control flow also determines the required effort for functional testing. In DO-178C, Objective 8 of Annex A Table A-7 requires that "Test coverage of software structure (data and control coupling) is achieved", referencing Sec. 6.4.4.2.c which states that structural coverage analysis should "confirm that the requirements-based testing has exercised the data and control coupling between code components". These terms are defined as:

Data coupling The dependence of a software component on data not exclusively under the control of that software component.

Control coupling The manner or degree by which one software component influences the execution of another software component.

The term "software component" is not precisely defined for these requirements. As stated in the CAST19 report [2],

there is currently no established understanding of the granularity of "component", it depends on the architecture being implemented. As a consequence, [2] suggests that certification projects should define what they mean by "component" in their specific architecture for demonstrating compliance to DO-178C.

The intent of structural coverage analysis is to provide a measure of the completeness of the testing process to ensure that requirements-based testing adequately exercised the software program under test [2]. Statement coverage, decision coverage and modified condition/decision coverage (Objectives 5-7 of Table A-7 of [20]) can be addressed at the module level by reviewing test cases and executing requirements-based tests of that module in isolation from other program modules. In contrast, Objective 8 of Table A-7 is primarily intended to be a verification of the integration activity: The intent of the structural coverage analysis of data coupling and control coupling is to provide a measurement and assurance that the software modules/components affect one another in the ways in which the software designer intended and do not affect one another in unintended ways, thus resulting in unplanned, anomalous or erroneous behavior. Hence, satisfying this objective is intended to provide a measure of the completeness of integration verification. As software increases in size and complexity, the quality of data and control coupling analysis is a growing concern for certification authorities [2].

In general, the data and control flow of the software can be determined by semantical static analysis. Semantics-based methods can be further grouped into unsound and sound approaches. Abstract interpretation is a formal method for *sound* semantics-based static program analysis which provides assurance that there are no false negatives with respect to the classes of defects under consideration. For data and control flow analysis the soundness of the analysis ensures that all potential targets of data and function pointers are taken into account.

This article is structured as follows: In Sec.2 we will give a brief overview of abstract interpretation and illustrate its application to runtime error analysis with the example of the analyzer Astrée. As discussed in Sec. 3 runtime error analysis can be seen as a prerequisite for data and control flow analysis, since it aims at detecting code defects that can corrupt the intended data and control flow. Sec. 4, Sec. 5 and Sec. 6 give a general overview of the core methodologies used in our work: sound global data and control flow analysis, sound taint analysis, and semantically refined program slicing. Based on those methodologies, Sec. 7 presents a novel approach for static data and control coupling analysis and interference analysis, that builds on a specification mechanism for software components appropriate for automatic static code analysis. It augments global data and control flow analysis by the software component level and presents a scalable and automatic taint analysis to determine data and control dependences between software components. Sec. 9 concludes.

2 Sound Static Source Code Analysis

The term static analysis is used to describe a variety of program analysis techniques with the common property that the results are only based on the software structure. Purely syntactical methods can be applied to check syntactical coding rules as contained in all relevant coding guidelines, including MISRA C/C++ [19, 15], or SEI CERT C/C++ [21]. A deeper understanding of the code such as knowledge about variable values, pointer targets, etc. requires semantical static analysis. It can be applied to check semantical coding rules which are also contained in the coding guidelines mentioned above, or to identify semantical code defects.

The semantics of a programming language is a formal description of the behavior of programs. The most precise semantics is the so-called concrete semantics, describing closely the actual execution of the program on all possible inputs. Yet in general, the concrete semantics is not computable. Even under the assumption that the program terminates, it is too detailed to allow for efficient computations. *Unsound* analyzers may choose to reduce complexity by not taking certain program effects or certain execution scenarios into account. A *sound* analyzer is not allowed to do this; all potential program executions must be accounted for. Since in the concrete semantics this is too complex, the solution is to introduce a formal abstract semantics that approximates the concrete semantics of the program in a well-defined way and still is efficiently computable. This abstract semantics can be chosen as the basis for a static analysis. Compared to an analysis of the concrete semantics, the analysis result may be less precise but the computation may be significantly faster.

Abstract interpretation is a formal method for sound semantics-based static program analysis [3]. It supports formal correctness proofs: it can be proved that an analysis will terminate and that it is sound, i.e., that it computes an over-approximation of the concrete semantics. Imprecisions can occur, but it can be shown that they will always occur on the safe side. Abstract interpretation-based static analyzers provide full control and data coverage and allow conclusions to be drawn that are valid for all program runs with all inputs. Nowadays, abstract interpretation-based static analyzers that can detect stack overflows and violations of timing constraints [22] and that can prove the absence of runtime errors and data races [4][12], are widely used for developing and verifying safety-critical software [9].

Runtime Error Analysis

At the source code level, the data and control flow of a program might be accidentally affected by unintended behavior, including unspecified and undefined behaviors of the programming language. Hence, a safe analysis of data and control flow must be embedded in a runtime error analysis that captures such undefined/unspecified behaviors.

In runtime error analysis, *soundness* means that the analyzer never omits to signal an error that can appear in some execution environment. If no potential error is signaled, definitely no runtime error can occur: there are no false negatives. When a *sound* analyzer does not report a division by zero in `a/b`, this is a proof that b can never be 0. If a potential error is reported, the analyzer cannot exclude that there is a concrete program execution triggering the error. If there is no such execution, this is a false alarm (false positive).

Throughout this article, we will focus on the Astrée analyzer as an example of sound static runtime error analyzer [13][18]. Astrée's main purpose is to report program defects caused by unspecified and undefined behaviors in C/C++ programs. The reported code defects include integer/floating-point division by zero, out-of-bounds array indexing, erroneous pointer manipulation and dereferencing (e.g., buffer overflows, null pointer dereferencing, dangling pointers), accesses to uninitialized variables, and further sequential programming defects. In addition, Astrée's sound thread interleaving semantics enables it to also report concurrency defects, such as data races, lock-/unlock problems, and deadlocks. Hence, Astrée not only determines the data and control flow within one thread of control, but can also capture interferences between different threads and their effects on the data and control flow within those threads.

Practical experience on avionics and automotive industry applications are given in [13][17][14]. They show that industry-sized programs of millions of lines of code can be analyzed in acceptable time with high precision for runtime errors and data races.

3 Data and Control Flow Errors

The purpose of data and control coupling analysis is to determine the effective data and control flow between software components which might be desired or undesired, depending on the properties of the software architecture. In addition, cases where there is an actual defect in the data or control flow behavior with respect to the semantics of the programming language, must be reported as an error.

In general, Astrée reports defects related to undefined or unspecified behavior of the programming language. Since their behavior is undefined or unspecified, all of them might have an effect on data or control flow – an example is a division by 0, which can cause the program to stop with a trap, obviously causing an unexpected change in control flow. In the following we will give an overview of the alarm classes of Astrée that specifically address memory safety or control flow behavior of the program.

Data Flow Errors

- *Out-of-bounds array access:* The value of the index used to access an array can be outside the feasible index range.
- *Invalid pointer dereference and manipulation:* This defect category includes dangling pointer accesses, invalid pointer dereferences and arithmetics, null pointer dereferences, misaligned dereferences, buffer overflows, etc.
- *Invalid dynamic memory allocation:* Allocation size is negative or too large.
- *Memory leak:* Memory may not be freed after dynamic allocation.
- *Uninitialized variable access:* This category includes read accesses of uninitialized local variables and read accesses to global/static variables without explicit initializer or prior assignment.
- *Data race:* Write-write or read-write data race, i.e. access to the same variable from at least two threads without proper synchronization.
- *Spectre vulnerability:* Occurrences of Spectre V1, V1.1, or SplitSpectre vulnerabilities.
- *Writes to constant memory:* Attempts to write to a constant.
- *Pointer aliasing:* Two pointer variables may alias which have been declared as distinct by the directive `__ASTREE_check_separate`.

Control Flow Errors

- *Non-returning functions:* Functions that may never return, e.g. due to infinite loops, calling `exit`, etc.
- *Incompatible function calls:* This category includes function calls with wrong number or incompatible types of parameters, incompatible return types, etc.
- *Deadlocks:* A deadlock occurs when two threads wait for each other indefinitely, e.g. due to blocked resources.
- *Recursions:* Recursive function calls are reported.
- *Infinite loops:* This category includes alarms about loops which definitely never terminate, and alarms about loops that might never terminate (e.g. only terminate upon reading a particular value from a truly volatile variable).
- *Lock/unlock problems:* This category includes attempts to unlock a mutex variable that has not been locked, locks without unlocks, locks acquired by a wrong task, etc.
- *C++ Exception:* The alarm reports C++ statements that can raise a C++ exception.
- *Pure virtual function call:* A pure virtual function is called in a specific context (C++).

It is apparent that these defects may invalidate any assumptions about the data and control flow behavior of the program, and hence, must also be considered for data and control coupling analysis. The same may also hold for other cases of undefined/unspecific behavior which are reported by Astrée. Hence sound static runtime error analysis can be seen as prerequisite for data and control coupling analysis. As emphasized in Sec. 2 the defect classes are not limited to sequential program execution but also include program defects induced by concurrent thread execution.

4 Data and Control Flow Analysis

Classical global data and control flow analysis determines the variable accesses and function invocations throughout program execution. It is centered on concepts included in the programming language, as compared to data and control *coupling* analysis that focuses on software components which are not expressed by programming language constructs. It constitutes the required basis for data and control coupling analysis.

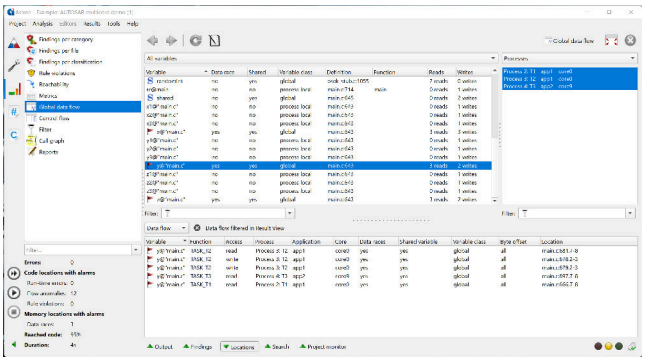


Figure 1: Astrée's Data Flow View.

In its basic data and control flow analysis module, Astrée tracks accesses to global variables, static variables, and local variables whose accesses are made outside of the frame in which the local variables are defined (e.g., because their address is passed into a called function). All data and function pointers are automatically resolved. The soundness of the analysis ensures that all potential targets of data and function pointers are taken into account. Astrée's data and control flow reports show the number of read/write accesses for every global, static, and out-of-frame local variable, lists the location of each access and shows the function from which the access is made. All variables are classified as being thread-local, effectively shared between different threads, or subject to a data race (cf. Fig. 1). Variable accesses can be interactively explored, e.g. , by selecting a variable and filtering for accesses from a particular thread or function, or for a given access type.

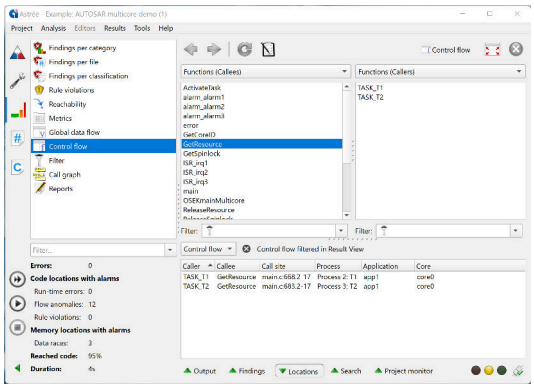


Figure 2: Astrée's Control Flow View.

The control flow is described by listing all callers and callees for every C function along with the threads in which they are called. In AUTOSAR projects, additionally the application and the core to which the executing thread belongs is listed. An example is shown in Fig. 2. The control flow can be interactively explored, e.g. , when selecting a function and filtering for its callers or callees or for threads in which it is called, all relevant call sites are displayed. There is also a call graph visualization which can be interactively explored as well (cf. Fig. 3).

In case of C++ programs, a class graph visualization shows a selected class with all its fields and methods as well as its sub- and superclasses with the relevant template instantiations in a

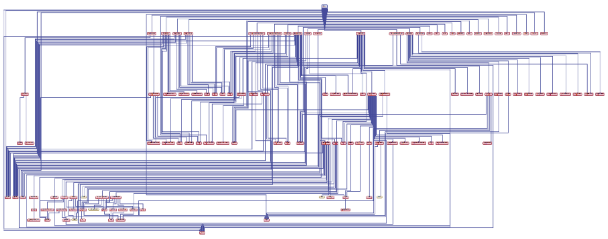


Figure 3: Astrée's Call Graph Visualization.

graph depicting the inheritance hierarchy (cf. Fig. 4).

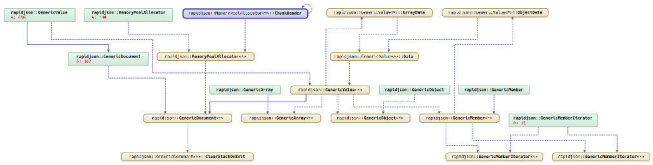


Figure 4: Astrée's Class Graph Visualization.

More sophisticated information about selected flows of values can be provided by two dedicated analysis methods: taint analysis and program slicing. *Taint analysis* is a forward analysis and can answer questions about program parts affected by reading corrupted input values. *Program slicing* is a backward analysis which can answer questions about the program parts which might influence the value of a particular variable at a particular program point.

All data and control properties described in this section are agnostic of the definition of software components; they are limited to native concepts of the programming language. Lifting this information to the level of software components is the topic of Sec. 7 which leverages the techniques of taint analysis and program slicing described in Sec. 5 and Sec. 6.

5 Taint Analysis

Taint analysis was first introduced as a dynamic analysis technique (e.g., in PERL), to try to find out which part of a code could be affected by some inputs. The original technique consisted in flipping normally unused bits, that would be copied around by operations and assignments. The same idea can be extended to static analysis by enhancing the concrete semantics of programs with tainting, the formal equivalent of the unused flipped bit in the dynamic approach. In the context of abstract interpretation, it is easy to abstract this extra information in an efficient and sound way, using dedicated abstract domains. Conceptually, taint analysis consists in discovering data dependencies using the notion of taint propagation. Taint propagation can be formalized using a non-standard semantics of programs, where an imaginary taint is associated to some input values. Considering a standard semantics using a successor relation between program states, and considering that a program state is a map from memory locations (variables, program counter, etc.) to values in $\mathscr{V}$, the *tainted* semantics relates

tainted states, which are maps from the same memory locations to $\mathscr{V} \times \{\text{taint}, \text{notaint}\}$, and such that if we project on $\mathscr{V}$ we get the same relation as with the standard semantics.

To define what happens to the *taint* part of the tainted value, one must define a *taint policy*. The taint policy specifies:

- **Taint sources** which are a subset of input values or variables such that in any state, the values associated with that input values or variables are always tainted.
- **Taint propagation** describes how the tainting gets propagated. Typical propagation is through assignment, but more complex propagation can take more control flow into account, and may not propagate the taint through all arithmetic or pointer operations.
- **Taint cleaning** is an alternative to taint propagation, describing all the operations that do not propagate the taint. In this case, all assignments not containing the taint cleaning will propagate the taint.
- **Taint sinks** is an optional set of memory locations. This has no semantical effect, except to specify conditions when an alarm should be emitted when verifying a program (an alarm must be emitted if a taint sink may become tainted for a given execution of the program).

A sound taint analyzer will compute an over-approximation of the memory locations that may be mapped to a tainted value during program execution. The soundness requirement ensures that no taint sink warning will be missed by the analyzer.

Astrée provides a generic abstract domain for taint analysis that can be freely instantiated by the users. It augments Astrée's process-interleaving interprocedural code analysis by carrying and computing taint information at the byte level. Any number of taint hues can be tracked by Astrée, and their combinations will be soundly abstracted. Tainted input is specified through directives (`__ASTREE_taint((var;hues))`) attached to program locations. Such directives can precisely describe which variables, and which part of those variables is to be tainted, with the given taint hues, each time this program location is reached. Any assignment is interpreted as propagating the join of all taint hues from its right-hand side to the targets of its left-hand side. In addition, specific directives may be introduced to explicitly modify the taint hues of some variable parts. This is particularly useful to model cleansing function effects or to emulate changes of security levels in the code.

The result of the analysis with tainting can be explored in the Astrée GUI, or explicitly dumped using dedicated directives. Finally, the taint sink directives may be used to declare that some parts of some variables must be considered as taint sinks for a given set of taint hues. When a tainted value is assigned to a taint sink, then Astrée will emit a dedicated alarm, and remove the sinked hues, so that only the first occurrence has to be examined to fix potential issues with the security data flow.

The main intended use of taint analysis in Astrée is to expose potential vulnerabilities with respect to security policies or resilience mechanisms. Thanks to the intrinsic soundness of the approach, no tainting can be forgotten, and that without any bound on the number of iterations of loops, size of data or length of the call stack. Based on its taint analysis, Astrée provides an automatic detection of Spectre-PHT vulnerabilities [10].

6 Program Slicing

The following definitions introduce the basic principles of static program slicing.

Definition 1 (Slicing Criterion) *Let P be a program. A slicing criterion in P is a tuple* (s,V) *which consists of a statement s and a set of variables V from P*

Definition 2 (Slice) *A slice S is a subprogram of P that exhibits the same behavior with respect to the slicing criterion* (s,V).

Computing a *statement-minimal* slice is an undecidable problem. However there are well-established algorithms for computing non-minimal, but still useful slices. A common approach is to compute a *System Dependence Graph* (SDG), which contains all data and control dependences of the program. Then a slice can be expressed as a reachability problem in this graph [7]. The precision of the slice directly depends on the precision of the SDG. However, computing precise system dependency graphs is a non-trivial task since it requires deriving intricate program properties. These may include points-to information for variable and function pointers, code reachability, context information or possible variable values at certain program points. As an example, over-approximating the set of possible destinations of a pointer variable blows up the size of the system dependence graph as it may add false dependences to statements which contain variables that would otherwise not be included in the slice. This may cause a drastic transitive increase in the number of dependences and vertices.

Astrée provides a novel concept of program slicing, that can be termed *sound semantically refined slicing*: its slicer can be run in the sequel of a finished Astrée analysis run, which makes it possible to leverage the invariants computed by its main fixpoint analysis. The system dependence graph computed by our approach is a sound abstraction of the data- and control dependences of a computer program. This follows from the soundness of the Astrée core analysis. As a consequence, the resulting slices are also sound. Minimizing false alarms is an important design goal of Astrée, which mandates a highly precise point-to analysis. Furthermore, Astrée detects code which is guaranteed to be unreachable for any possible program execution, and which, consequently, can be ignored when computing the slices. Hence, compared to slicing without leveraging Astrée invariants, a significant precision and efficiency gain is achieved by reducing the amount of vertices and the amount of data- and control dependences in the system dependence graph. This efficiency improvement makes it possible to compute precise slices for very large programs in feasible time.

Semantically refined slicing can be run in context-insensitive mode (considering all possible call contexts) and context-sensitive mode (considering exactly one call context). To compute context-sensitive slices we enhance the slicing algorithm of [7] with a description of call contexts (stacks). In each step of the reachability analysis we additionally check that the de-

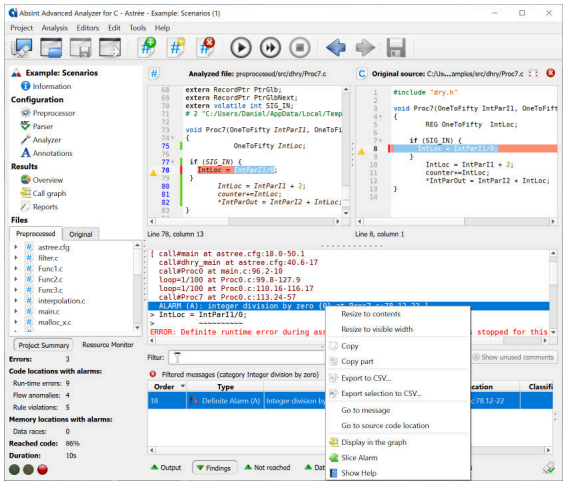


Figure 5: Alarm Slicing in Astrée

pendences under examination match the relevant stacks. Dependences which do not match are discarded. When following a dependency edge which represents a function call, the topmost function is removed from the stack. For one given program point a context-insensitive slice is identical with the union over all context-sensitive slices.

In contrast to context-insensitive slices, context-sensitive slices do not capture all possible behaviors of the original program which influence the slicing criterion. Instead, the behavior described by the slice is restricted to execution paths which are in accordance with the set of considered call contexts. Context-sensitive slices tend to be significantly smaller than context-insensitive ones.

The different slicing modes presented in this section are relevant for demonstrating safety and security properties. Sound slices can be computed by *context-insensitive* analysis-enhanced slicing. With these slices it is possible to show that certain parts of the code or certain input variables might influence or cannot influence a program section of interest. They yield a global overview of these properties for the entire program.

In contrast to that, *context-sensitive* analysis-enhanced slicing, which only considers a subset of possible contexts, is more suitable for investigating the influence of a certain code section, e.g. a function, or a module, on the program location of the slicing criterion. Hence it is perfectly suited as a basis for automatic alarm slicing, which is available in Astrée. To give an example, the critical situation for a division by zero alarm, which is reported for a given context, is precisely the context where the denominator becomes 0. Therefore the alarm slice is a partial slice for the variables used in the denominator, which only considers program paths leading to this particular context.

This concept of alarm slicing has been implemented in Astrée and is available from the GUI: from the context menu of an alarm in the Astrée graphical user interface the computation of a slice for the alarm can be automatically triggered, as shown in Fig. 5.

A detailed experimental survey of Astrée's semantically refined program slicer with programs from automotive and avionic industry is given in [11]. It demonstrates that semantically refined slicing (termed analysis-enhanced slicing in [11]) can be applied to industry-size code with high precision and with feasible memory and computation time requirements.

7 Data and Control Coupling

The CAST19 report [2] reviews the data and control coupling requirements of the DO-178C and provides clarifications about the intended use. It emphasizes that the purpose of data coupling analysis includes (among others) identifying data dependences, verifying the data interfaces between modules/components through testing and analyses, identifying inappropriate data dependencies, evaluating the need for and accurate use of global data, and evaluating input/output data buffers.

The purpose of control coupling is stated to include identifying control dependencies, identifying inappropriate control dependencies, verifying correct execution call sequence, defining and evaluating the extend of interface depth, and assisting in WCET analysis.

In addition, as outlined above, data coupling and control coupling together aim at providing a completion check of the integration testing effort. The CAST19 report further clarifies that the "Data Coupling and Control Coupling Analyses" objective of the DO-178C (Objective 8 of Annex A, Table A-7 [20]) may be satisfied as a static activity, a dynamic activity, or a combination.

In the following we propose a concept for sound static data and control coupling analysis that builds on Astrée's data and control flow analysis as described in Sec. 4, and which satisfies all requirements mentioned above. First, in Sec. 7.1, we present a flexible and extensible concept for specifying software components and critical data and control flow interactions between them. Sec. 7.2 outlines static data and control flow analysis augmented by the concept of components. Sec. 7.3 presents an automatic taint analysis that efficiently tracks the flow of values between components, and automatically reports undesired data flow and undesired control dependencies. In case of taint sink alarms indicating undesired data or control flow, alarm slicing can be used to track down the cause of the undesired behavior.

7.1 Specifying Software Components

Since there is no established understanding of the granularity of "component", a specification mechanism is needed that allows users to specify their concept of software components and the "interesting" data or control flow between them. For the purposes of data and control coupling, the full power of special-purpose architecture description languages (ADL) is not needed (cf. AADL [6], ArchiMate [1], SysML [23], or UML-based architecture specifications), since the aim is to ascertain the desired properties by automatic static analysis from the source code.

A simple starting point is to define a software component as a collection of all variables and functions defined in a set of source files. This can be provided by (conceptual) annotations of source files or functions:

```
"xfile1.c" insert :
__ASTREE_attributes((component("trusted")));
```

```
"yfile1.c" insert :
__ASTREE_attributes((component("non-trusted")));
```

This schema can be easily extended to more complex component definitions, e.g., based on individual functions, all files in a sub-directory, etc.

In addition to defining the elements of a software component, the specification also allows to declare component interactions that should be "observed" during the analysis, i.e., specifically tracked and reported. To this end, the analyzer can be instructed to report control and data flow from the component "trusted" to the component "non-trusted", and vice versa:

```
<observe>
  <item key="trusted">"non-trusted"</item>
  <item key="non-trusted">"trusted"</item>
</observe>
```

A dedicated view in the graphical user interface of Astrée allows to conveniently create the software component specification, and then export them to an XML file. Alternatively, the XML file may also be generated automatically from an existing ADL specification.

The specification of component interactions to be observed mostly aims at explicitly reporting unexpected or forbidden interactions. By placing components with expected interactions under observation, Astrée can also address intended interactions, however, in that case, the result of Astrée only contributes potential interactions due to its sound over-approximation: a reported component interaction can also be a "false alarm". A dedicated tag to support simultaneous tracking of unexpected and intended interactions is currently not available but could be easily added.

7.2 Augmented Data and Control Flow Analysis

The first step towards detecting and reporting data and control flow relations between software components is to enhance the standard data and control flow analysis by taking the component definitions into account, which we term *augmented* data and control flow analysis.

The augmented data and control flow analysis of Astrée will thus report any interaction between two software components of the following classes:

- Calling a function of a given component, including indirect calls through pointer dereferences.
- Writing to a variable defined in a given component, which can be a global variable that belongs to the component or a local variable defined in a function of the component. That includes indirect writes through pointer dereferences.
- Reading from a variable defined in a given component. That includes indirect reads through pointer dereferences.

These interactions already capture most interferences between components, including indirect ones, such as scenarios when the address of a variable of component A is stored in component C, and then read from C by component B which writes to it through dereference.

They will not capture more subtle value dependencies, such as component C copying the value of a global variable of component A, store it in one of its variables and then give the value to component B that may take different control flow based on that value. To track such interferences, we need to follow the flow of values, and we can do that using taint analysis techniques – cf. Sec. 7.3.

The data and control flow views in the Astrée GUI presented in Fig. 1 and Fig. 2 of Sec. 4, and the corresponding data and control flow reports will be augmented by additional columns which make the component assignment explicit. Each access to a variable X then is reported with the following information in the data flow view:

– variable name
– component to which the variable belongs
– location of access (may be pointer dereference)
– access type (read/write)
– function containing the access
– component to which the accessing function belongs
– task in which the accessing function is executed
– application of task
– core to which application has been assigned
– access locality (thread-local, effectively shared, data race)
– notification for components under observation

Each function call is reported with the following information in the control flow view:

– caller
– component of caller
– callee
– component of callee
– call site (may be function pointer call)
– task in which the call is executed
– application of task
– core assigned to application
– notification for components under observation

The data and control flow reports can be generated in various open formats that support post-processing, so it is easy to query for any component interaction of interest. If a flow from a component X to a component Y has been introduced with an `observe` tag in the component specification, a dedicated notification is generated about any observed flow from X to Y.

Also the visualizations of Fig. 3 and Fig. 4 will be augmented by component information so that it will be possible, e.g., to highlight the nodes for selected components, or show different components in different colors.

This mechanism gives a sound overapproximation of all variable accesses and function calls, and establishes their link to the software component definition. It makes it easy to spot flows under observation, hence call attention to flows that should be investigated.

7.3 Data and Control Coupling Analysis

As outlined in Sec. 5, taint analysis allows to track the flow of values in a project. One solution to compute data dependencies between components is to assign a distinct taint hue to each component, and use all global and local variables of a component as taint source with the component hue. To be automatically notified about all components where these values flow, all reads in each component are declared as taint sinks for all

other components hues. The required `__ASTREE_taint` and `__ASTREE_taint_sink` directives can be generated automatically. To focus on particular component interactions, the automatic generation of taint source and taint sink directives can be restricted to only those component flows placed under observation, which reduces the number of taint sink alarms about component dependences. Of course, it is also possible to manually select specific component flows to observe, or perform the taint analysis on a per-variable base for individual variables deemed critical.

This approach not only detects additional interferences compared to the augmented data and control flow analysis, it also allows to account for authorized interactions in a fine-grained way. One example is that data flow from component Y to X is forbidden, unless the access is made by specific gateway functions. Such interactions can be modeled by taint-cleaning operations which remove the taint in those gateway functions (sanitization points), reducing the number of legitimate interferences that need to be examined. To further facilitate this examination, automatic alarm slicing on taint sink alarms can be used to help identify the program regions responsible for undesired component interactions.

In addition to the data flows reported by the automatic tainting as described above, taint analysis also allows to focus on control coupling. The generation of taint sink directives can be adapted, so that only values used in guards (for conditional statements, while loops or switch statement) and in function pointer dereferences are considered as taint sinks.

Hence, the taint analysis of software components can be performed in data coupling and control coupling mode, satisfying the requirements of the CAST19 report as described above.

It should be noted that taint analysis is a complement to the augmented data and control flow analysis (cf. Sec. 7.2), but cannot replace it, since it focuses on the data flow aspect and does not report invocations of functions from other components. Its data flow results are more powerful: taint analysis can keep track of call-by-value parameters of functions and of function return values and hence report dependences with respect to constants or local variables. It not only shows that at a certain location a given variable is accessed, but also provides the corresponding call context – as an example, if a function is called with a pointer argument, and only in one call site an address of a variable from another component is passed, then the alarm context will show precisely this call site. Furthermore, it makes it possible to narrow the focus on selected component interactions and tracks the relevant data flows also through code that does not belong to one of the components under observation. Both of them together, i.e. , the combination of augmented data and control analysis and the taint analysis for software components provide a sound interference analysis. They report all interactions between software components executed in one or multiple concurrent threads, pinpoint interactions under observation, hence enable users to find undesired interactions, and browse the code locations where they happen. They also provide a basis for checking intended interactions and allow computing metrics about the data and control flow between the software components.

As an example, consider the following code sequence:

```
/* file main.c */
void main(void) {
  int x;
  x = F_a(0);
  g_b = x;
  F_c(g_b);
}

/* file A.c */
__ASTREE_attributes((component("A")));
int g_a;
int F_a(int x) {
  int y=x;
  return x + 2;
}

/* file B.c */
__ASTREE_attributes((component("B")));
int g_b;

/* file C.c */
__ASTREE_attributes((component("C")));
int g_c;
int F_c(int x) {
  g_c=g_a;
  if (g_c < 0) g_c = F_a(3);
}
```

The augmented data and control flow analysis provides the information that component C depends on A through variable read and function call, since `F_c` reads variable `g_a` and calls function `F_a`. Component tainting raises four taint sink alarms: first it reports that component C depends on A at the assignment `g_c=g_a` in `F_C`, which is information also available in the augmented data flow view. It also reports a dependence from A to C at `g_c = F_a(3);` in `F_C` because the return value of `F_A` is assigned to `g_c`. This is more precise information than provided by the augmented control flow view which just reports a call to a function from C at this location. Third, the taint analysis reports a dependence from C to A at the assignment `y=x` in `F_a`, since C passes a constant value to `F_a`, which then is assigned to a local variable of A. Finally, the component tainting reveals that component B depends on component A through variable `x`: a taint sink alarm is raised at the assignment `g_b = x` in `main.c`, which is outside of components A, B, and C. Note that in the example, all variables are directly accessed, however, the analysis results would not change if all variable accesses and function calls were made via pointers.

8 Experimental Results

The augmented data and control flow analysis is part of the sound runtime error analysis, hence, it is not associated with additional runtime or memory overhead.

To assess the performance of the taint-based components analysis we investigated four industry projects of various sizes, two from the avionics domain and two from the automotive domain. We partitioned the code in software components and determined the increase in analysis time and memory consumption caused by component tainting. Tab. 1 shows the character-

istics of the projects, the results obtained with component tainting are summarized in Tab. 2. Column S of Tab. 1 gives the size of the projects in million physical lines of code[1] after preprocessing, column N_C indicates the number of software components. The large number of software components in project AE2 results from declaring every application source file as an own component; in the other projects the components consist of larger code parts. Column T and M show the analysis time and memory consumption in their original configuration.

Project	S [$MLOC$]	N_C	T	M [GB]
AE1	0.14	11	33m	2.43
AE2	1.08	4309	8h 27m	15.53
AU1	5.46	11	7h 1m	39.81
AU2	5.03	35	10h 44m	31.52

Table 1: Project Characteristics

Column T_t and M_t of Tab. 2 show the analysis time resp. memory consumption with component tainting. The increase in analysis time and memory consumption associated with component tainting is given in column Δ_T resp. Δ_M. The results show that the overhead of component tainting is low. The increase in memory consumption is below 1% for all test cases except AE2. For AE2 an extremely high number of components has been defined, which also entails a large number of component interactions: there are 78283 alarms about variable accesses creating component dependencies (cf. Tab. 3). However, even in that scenario the memory overhead is only 4.76%.

The increase in analysis time is between 4.93% and 8.07% for the larger projects. The largest increase of 8.07% occurs in project AU2, where the analysis time increases by 45 min from a total analysis time of 10 hours 44 minutes. On the smallest project, AE1, there is no measurable difference in memory consumption.

Project	T_t	M_t [GB]	ΔT	ΔM
AE1	33m	2.43	0%	0%
AE2	8h 52m	16.27	4.93%	4.76%
AU1	7h 24m	40	5.5%	0.48%
AU2	11h 36m	31.78	8.07%	0.82%

Table 2: Results with Component Tainting

Taint sink alarms for component dependences can be limited to flows under observation. However, in the experiments, we configured Astrée to generate taint sink alarms for every cross-component flow, since the goal was to assess performance on large projects with many component interactions. The number of cross-component variable accesses derived from the augmented data flow view is listed in column N_V of Tab. 3, column N_C gives the number of cross-component function invocations, and the number of taint sink alarms denoting cross-component data flows is listed in column N_{TA}. All numbers indicate the number of *code locations* which exhibit a cross-component interaction, e.g., all accesses to a variable g_A of component A from component B are separately counted.

[1] I.e., comment lines and empty lines are not counted.

The difference between columns N_V and N_{TA}, on the one hand, is due to additional dependences discovered by tainting, e.g., due to values passed through library functions not assigned to a specific components, call-by-value function parameters and function return values. On the other hand, as discussed in Sec. 7.3, component tainting focuses on data dependences and tracks the effect of all function calls, but does not separately report the calls themselves. The effect of infrastructure code without component assignment is particularly visible in project AE2. The components do not explicitly invoke one another, the invocations are made in an infrastructure layer we did not assign to an own component. Dependencies carried through that infrastructure layer, e.g., by copying component variables via infrastructure variables, are visible with component tainting, but not in the augmented data flow view.

Project	N_V	N_C	N_{TA}
AE1	147	289	415
AE2	40929	0	78283
AU1	9569	1695	10285
AU2	8485	4242	16852

Table 3: Code locations with cross-component interactions

9 Conclusion

At the source code level, the data and control flow of a program might be affected by behavior unintended by the programmer, including unspecified and undefined behaviors of the programming language. Hence, a safe analysis of data and control flow must be embedded in a runtime error analysis that captures such undefined/unspecified behaviors. Determining the data and control flow in the source code and making sure that it is compliant to the intended control and data flow as defined in the software architecture is a common requirement in all contemporary safety norms. The aim of data coupling and control coupling analysis objective of DO-178C is to provide a measure of the completeness of integration verification by ensuring that the software components affect one another only in intended ways. Furthermore all safety norms require demonstrating the freedom of interference between software components of different criticality levels.

In this article we have presented a novel approach for data and control coupling analysis and interference analysis, that builds on a specification mechanism for software components appropriate for automatic static code analysis. It is based on the sound static analyzer Astrée that allows to prove the absence of critical runtime errors that could cause memory corruption and control flow corruption. Our approach augments sound global data and control flow analysis by the software component level, proposes a scalable and automatic taint analysis to determine data and control dependences between software components, and incorporates semantically refined program slicing to help identify the program regions responsible for undesired component interactions. It enables a sound approximation of data and control coupling and a sound interference analysis that help demonstrating the correctness of the data and control flow of the program and contribute to satisfying the data and control

coupling and freedom of interference verification goals of contemporary safety norms. The experimental results show that our approach is highly efficient and can be applied to industry-size software projects.

References

[1] The ArchiMate Enterprise Architecture Modeling Language. https://www.opengroup.org/archimate-forum/archimate-overview[retrieved: Jan. 2021].

[2] Certification Authorities Software Team (CAST). Position Paper CAST-19. Clarification of Structural Coverage Analyses of Data Coupling and Control Coupling, 2004.

[3] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proc. of POPL'77*, pages 238–252. ACM Press, 1977.

[4] D. Delmas and J. Souyris. ASTRÉE: from Research to Industry. In *Proc. 14th International Static Analysis Symposium (SAS2007)*, number 4634 in LNCS, pages 437–451, 2007.

[5] C. Faure and V. Delebarre. Automatic proof of freedom from interference with iffree. In *Proccedings of the 10th European Conference on Software Architecture Workshops, Copenhagen, Denmark, November 28 - December 2, 2016*, page 36, 2016.

[6] P. Feiler, D. Gluch, and J. Hudak. Technical Note CMU/SEI-2006-TN-011. The Architecture Analysis & Design Language (AADL): An Introduction. Technical report, Software Engineering Institute, Carnegie Mellon University, 02 2006.

[7] S. Horwitz, T. Reps, and D. Binkley. Interprocedural slicing using dependence graphs. *ACM Trans. Program. Lang. Syst.*, 12(1):26–60, Jan. 1990.

[8] ISO 26262. Road vehicles – Functional safety, 2018.

[9] D. Kästner. Applying Abstract Interpretation to Demonstrate Functional Safety. In J.-L. Boulanger, editor, *Formal Methods Applied to Industrial Complex Systems*. ISTE/Wiley, London, UK, 2014.

[10] D. Kästner, L. Mauborgne, C. Ferdinand, and H. Theiling. Detecting Spectre Vulnerabilities by Sound Static Analysis. In R. F. Anne Coull, Steve Chan, editor, *The Fourth International Conference on Cyber-Technologies and Cyber-Systems (CYBER 2019)*, volume 4 of *IARIA Conferences*, pages 29–37. IARIA XPS Press, 2019. Archived in the free access ThinkMind^{TM} Digital Library, http://www.thinkmind.org/download.php?articleid=cyber_2019_3_10_80050.

[11] D. Kästner, L. Mauborgne, N. Grafe, and C. Ferdinand. Advanced Sound Static Analysis to Detect Safety- and Security-Relevant Programming Defects. In J.-C. B. Rainer Falk, Steve Chan, editor, *8th International Journal on Advances in Security*, volume 1 & 2, pages 149–159. IARIA, 2018.

[12] D. Kästner, A. Miné, L. Mauborgne, X. Rival, J. Feret, P. Cousot, A. Schmidt, H. Hille, S. Wilhelm, and C. Ferdinand. Finding All Potential Runtime Errors and Data Races in Automotive Software. In *SAE World Congress 2017*. SAE International, 2017.

[13] D. Kästner et al. Finding All Potential Runtime Errors and Data Races in Automotive Software. In *SAE World Congress 2017*. SAE International, 2017.

[14] D. Kästner et al. Analyze This! Sound Static Analysis for Integration Verification of Large-Scale Automotive Software. In *Proceedings of the SAE World Congress 2019 (SAE Technical Paper)*. SAE International, 2019.

[15] M. Limited. MISRA C++:2008 Guidelines for the use of the C++ language in critical systems, June 2008.

[16] A. Miné. Static analysis of run-time errors in embedded real-time parallel C programs. *Logical Methods in Computer Science (LMCS)*, 8(26):63, Mar. 2012.

[17] A. Miné and D. Delmas. Towards an Industrial Use of Sound Static Analysis for the Verification of Concurrent Embedded Avionics Software. In *Proc. of the 15th International Conference on Embedded Software (EMSOFT'15)*, pages 65–74. IEEE CS Press, Oct. 2015.

[18] A. Miné et al. Taking Static Analysis to the Next Level: Proving the Absence of Run-Time Errors and Data Races with Astrée. In *8th European Congress on Embedded Real Time Software and Systems (ERTS 2016)*, Toulouse, France, Jan. 2016.

[19] MISRA (Motor Industry Software Reliability Association) Working Group. *MISRA-C:2012 Guidelines for the use of the C language in critical systems*. MISRA Limited, Mar. 2013.

[20] Radio Technical Commission for Aeronautics. RTCA DO-178C. Software Considerations in Airborne Systems and Equipment Certification, 2011.

[21] Software Engineering Institute SEI – CERT Division. *SEI CERT C Coding Standard – Rules for Developing Safe, Reliable, and Secure Systems*. Carnegie Mellon University, 2016.

[22] J. Souyris, E. Le Pavec, G. Himbert, V. Jégu, G. Borios, and R. Heckmann. Computing the worst case execution time of an avionics program by abstract interpretation. In *Proceedings of the 5th Intl Workshop on Worst-Case Execution Time (WCET) Analysis*, pages 21–24, 2005.

[23] OMG Systems Modeling Language (OMG SysML™) Version 1.6. https://www.omg.org/spec/SysML/1.6/PDF[retrieved: Jan. 2021].

[24] B. Zimmer, C. Dropmann, and J. U. Hanger. A systematic approach for software interference analysis. In *25th IEEE International Symposium on Software Reliability Engineering, ISSRE 2014, Naples, Italy, November 3-6, 2014*, pages 78–87, 2014.

Automatic Support for Requirements Validation

Assioua Yasmine[†], Rabéa Ameur-Boulifa[*], Patricia Guitton-Ouhamou[†], Renaud Pacalet[*]
[*]LTCI, Télécom Paris, Institut Polytechnique de Paris, France
firstname.lastname@telecom-paris.fr
[†]Renault Software Labs, France
firstname.lastname@renault.com

***Abstract*—The automotive industry is currently going through rapid changes from a mechanical industry to one driven by innovation in electronics and embedded software. This significant change creates also significant challenges to the industry. One of the most important is the ability to create safe vehicles, emphasizing the importance of safety by design. This paper is intended to contribute to current activities working towards an industry-wide development of reliable and secure systems. Correct by design methodology, including formal methods, have the potential to improve dependability of systems in this domain. And their use at an early stage of the development process ensures faster time to market. In this paper, we present tool support for our approach that aims at integrating the formal analysis and verification of functional requirements from early stages of the development life cycle, by using model checking technique. From informal requirement specifications the tool delivers models. They will be used to produce evidences that the requirement specifications are realizable, otherwise it can guide their revision. The approach is illustrated by a case study based on a specific function of autonomous vehicles.**

***Keywords*—Requirements analysis, Reliable systems, Model-based design, Systems engineering.**

I. Introduction

Designing complex systems is a difficult task. Conventional development approaches provide a unified process for system development, from requirements analysis to implementation [14]. Such approaches play a major role in software engineering practices. But quality of designed products in terms of correctness and robustness still remains a hot spot. The issue of building a practical but accurate methodology for designing safe and correct systems still remains unsolved. Such approaches are generally based on late-stage validation relying on testing to check that the requirements specifications are correct or to detect possible flaws, which leads to hight-cost corrective measures or even huge financial losses. According to [6] and [22] financial losses caused by failures represent more than 5% of the overall turnover of the companies. On the other hand, the study published in [12] reports that 64% of all errors are introduced during requirements specification and design, and 36% of the errors are introduced during the implementation phase.

Test coverage techniques which are commonly used to ensure the quality of developed systems cannot be used at early stages. In contrast, formal analysis grants much higher potential to discover flaws during the design phase of a system. Yet, formal techniques for verification and validation often require a strong technical background that limits their usage especially in the industrial context.

In [32] we proposed a model-based approach for early validation of requirements that relies on formal methods. To facilitate the design of automotive software from requirements, we suggest enhancing the system design process with the use of formal methods, and to offer a tool that system designers can use to assess through a rigorous and systematic process that the developed systems is compliant with the predefined requirements. This paper is an extension of this previous work. In this one, we have significantly extended and implemented the proposed method. First, we enrich and improve the requirements expression language to address more applications in the automotive domain. The work resulting from this research is an approach to analyse adaptable and extensible templates that can be used to specify requirements in the automotive domain. Second, we introduce the implementation of the resulting template in Xtext an eclispe-based tool [7]. Finally, we significantly extend the empirical study by evaluating our approach with an additional use-case. We demonstrate the applicability of our tool on an industrial context through a realistic use-case: the Automatic Park Assist system.

While this work is conducted in a context of analysis of the software requirements embedded in vehicles, we believe that our approach is not related to any specific system. This is why throughout the paper we use the term generic ”system” regardless of which system (software or physical components) it is referred to.

The rest of the paper is organised as follows: section II introduces the overall process for the formalization of requirements by showing the steps from input data to final output. Section III gives the structure of the requirements language used to specify automotive requirements. It resembles EARS requirements structure. Section IV provides the technical approach for the formalization of requirements. This includes the natural-like languages template for specifying requirements, as well as the association between templates and their semantics in the UPPAAL formalism, for the derivation of formal models. Section V validates our approach. Section VI surveys related work before concluding in section VII.

II. Approach for requirements validation

The approach we advocate for formally analysing and validating requirements is described in Fig. 1. Our tool, which

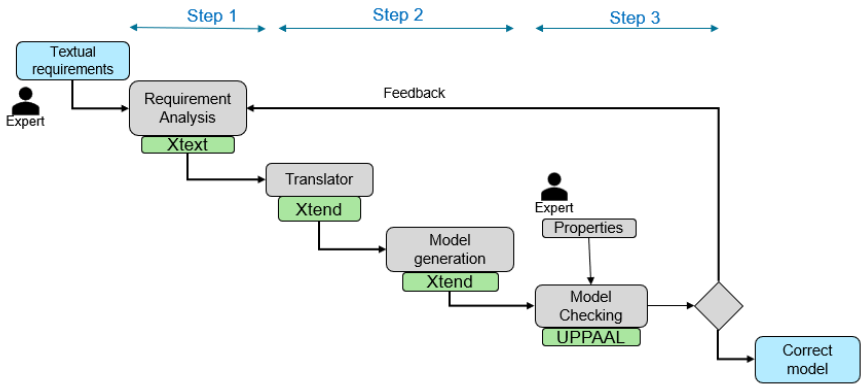


Fig. 1: An overview of the end-to-end approach for analysing formally automotive requirements

is a proof of concept, is built on the basis of open-source tools. It consists of three main parts:

- Step 1: To have a clear understanding of the automotive requirements, we conducted a deep analysis of textual requirements specifying different use-cases in our company. This work led to the identification of patterns and the establishment of their classification. From those patterns we created a grammar that gathers all the possible forms that a requirement can take. The classification is built regarding their structures and their role in the model construction. As with a programming language, the overall structure involves fixed terms (keywords), such as **while** and **shall**, that can be combined with free-form elements with no predefined scheme. The transformation of these structures into a formal notations allows to verify their completeness, consistency, and correctness by using automatic tools. The transformation task is implemented using the open-source software framework Xtext used for developing programming languages and domain-specific languages.
- Step 2: During this step we translate the requirements from their textual form to a model. We chose the UPPAAL automaton formalism as the modelling language for its ability to represent all the aspects desired and targeted by the analysis. In particular, the UPPAAL formalism supports various constructors, which gives it great power of expression. For instance, the use of expressions (for expressing guards and assignments) built over variables and parameters, and the use of synchronisation over channels. It also allows to model a global system by composition of subsystems (processes). The translation procedure is implemented using the open-souce Eclipse Xtend framework, also widely used for developing domain-specific languages.
- Step 3: The outcome of the construction of step 2 is either a valid model, or a non-valid model. In the former case, the automaton has an initial state, from the initial state there is a path to all other states, and the automaton is deterministic. A valid result provides an early evidence of requirement's consistency and correctness, it can then be used to a posteriori analysis and verification of properties. In the latter case, the malformations are shown to engineers who will correct or complete the requirements in an iterative process. To perform the analysis and verification task, we use the UPPAAL model checker: this tool allows to check automatically the consistency of the model against properties that can be expressed using specific languages (such as CTL language) or observer automata. In cases where properties are violated, the tool is able to provide a precise and useful feedback to the developer (engineer) to understand the source of the violation, and possibly how to fix it.

III. Requirements Specification

Many large firms, such as Renault, write technical specifications for the system under design or for the software application before getting started. The features and behaviour of the system or the application are described in a set of documents. It includes a variety of texts and graphics that defines the intended functionality required by the customer. At Renault, this document is called STRComp (from System Technical Requirement Component). This document includes textual templates that are requirements written in a constrained natural language, i.e, natural-like language with restricted syntax.

The analysis of the STRComp of different case studies studied during our work allowed us to classify the requirements into categories according to their role and their nature. Overall, we have identified three categories:

- *Interface requirements* specification defines interface for the system under design, in that it describes how to access the functionality provided by the system via variables or signals. This type of requirement gives the names of the variables and signals, and their domain. For example, the requirement scheme that is used to specify the signals sent by the system to the environment (subsystems) is of the following form:

```
<system> shall send from <actor> the signal <name>
[with the following values : (- <value>)+ ]
```

In the same way, requirement schemes are defined specifying the signals received by the systems:

```
<system> shall receive from <actor> the signal <name>
[ with the following values : - <value>)+ ]
```

- Functional requirements or *specification points* describe the desired behaviour of the system: what the system is intended to do and what conditions it must meet. The requirements of this category are written in a format close to the Easy Approach to Requirements Syntax (EARS) notation [24]. The structure of a specification point is made up of one or more patterns combined in the same order. A pattern is a compact and structured template; it consists of attributes and fixed syntax elements (keywords). In all the syntactical forms given below, the terms in bold are fixed syntax elements, while those between rafters are attributes. The generic pattern syntax used in our study is the following:
  1) A state-driven requirement defines the states of the system and the condition or triggering event that enables/disables actions:

     ```
     while <state> and <condition> [when <trigger>]
     <system> shall <action>
     ```

     An action describes the behaviour that the system should achieve. It is defined by a change of state or by an occurrence or consecutive occurrences of signal setting actions. It has the following format:

     ```
     <action> ::= switch to <state>
                | set <name> to <value>
                | activate <function>
                | release <function> control
     ```

     specifying a change of state, an update of a variable to a given value, an activation of a function, or a release of a control function of an actuator or sensor. The terms **activate** and **release** are introduced for readability reasons; they are used to raise requests that refer to an updating of signals, i.e. actions.
  2) An event-driven requirement defines how the system should behave in response to an effect (nominal or failure) of an action or an external stimulus that occurs:

     ```
     when <trigger>, <system> shall switch to <state>
     ```

  3) An action-driven requirement defines the action that is invoked when entering a certain state:

     ```
     when entering <state>, <system> shall (-<action>)+
     ```

  4) Some requirements define the conditions that enable a certain event to be issued. These conditions are complex and timed. A typical example of such requirements is the following:

     ```
     <system> shall detect <trigger>,
     if <name> = <value> for more than <delay>
     ```

     It specifies the conditions under which a certain event is triggered.

  where `<system>` a name of the system, `<name>` a name of signal, `<state>` a name of state, `<condition>` a condition that enables/disables actions, `<trigger>` an affect of an internal action or an external stimulus, and `<action>` a processing step, e.g. operations updating variables. These basic forms of requirements can be combined to specify complex requirements. For instance, the maximal representation of a state-driven requirement is given in Fig. 2. The identification of each pattern and its semantics allows to build the global model of the specified system.
- *Constraints* are also functional requirements that impose restrictions on the realisation of the system. They describe what constraints the realisation must satisfy to prevent various risky behaviour. They are of two types: those called *plausibility* that define rules about execution which are plausible and which are not, and those called *priorities* that define the priorities between its subsystems. As an example of a constraint:

  ```
  if <system> is in <state> and entrance conditions to
  <state> are satisfied, <system> shall switch to <state>
  ```

  this additional rule specifies the execution to be set aside and the execution to be imposed instead.

Although it is well-structured, this language, like RELAX [31] and Stimulus [20] languages, is classified as a natural language. It does not meet several assessing criteria for requirements engineering approaches, which are required by industry standards (as mentioned in ISO/IEC/IEEE 29148-2011 [19]). Among these criteria we find verifiability, which assesses whether an approach supports the possibility of formally verifying the properties of the requirements. This is inherently not the case for this language, as it is not tied to a formal semantics.

IV. From textual description to model

The main objective of our approach is to provide early evidence that a given set of requirement specifications are realizable. This objective is specifically related to the requirements formalisation challenge [26], [29]. The formalisation task refers to the transformation of requirements into formal models. In this section, we outline how formal models are derived from textual requirements.

For this purpose, we specified the requirement specifications using the state machine model. The main criteria used in selecting this formalism includes its precise formal semantics, but also its integration with automatic verification tools such as model-checkers. We have used the UPPAAL model checker [23] as it has proven to be successful and practical in various domains. This tool offers an integrated environment for analysing real-time systems based on networks of timed automata. It provides an editor, a symbolic simulator and a verifier, for modelling, enabling examination of dynamic executions, and verifying (by covering exhaustive dynamic behaviour).

A. UPPAAL Model

The model-checker UPPAAL is based on the theory of timed automata. Within this tool, a system is modelled as

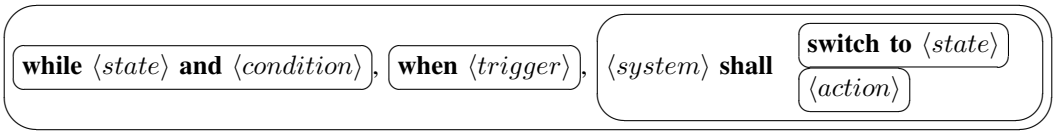


Fig. 2: The generic syntax of state-driven requirements

a network of timed automata that communicate in a synchronous fashion using so-called channels. A timed automaton is a classical finite-state machine extended with clocks. Each transition (edge) of such an automaton can be decorated with three (optional) labels:

$$\mathbf{S} \xrightarrow[update]{guard \quad sync} \mathbf{S'}$$

- *guard* expressing a condition on the values of the variables, which must be satisfied for the transition to be fired.
- *sync*, to represent synchronisation. Automata can synchronise over channels. The synchronisation mechanism in UPPAAL is a hand-shaking synchronisation: two processes take a transition at the same time on a common channel **e**, one will have a transition labelled **e!** to identify the sender and the other a transition labelled **e?** to identify receivers. UPPAAL offers urgent channels to force a synchronisation as soon as it is possible. It also supports the notion of broadcast channels that allow 1-to-many synchronisations.
- *update* a set of actions, which are expressions with a side-effect, i.e, assignment of variables or reset of clock. They may also be functions calls.

Note that S and S' are called *locations* in UPPAAL. A state of a UPPAAL model is defined by the locations of all automata being part of the model, the clock values, and the values of the variables. This other feature of UPPAAL is very useful for the applicability of our approach in an industrial context. It leads to a reduction of the state-space representation: automata with an infinite number of states can be represented by a finite set of symbolic states.

In addition to the network of automata, UPPAAL model includes a declaration part, which contains declarations of clocks, (global and local) variables, synchronisation channels, and constants manipulated by automata.

B. Model Construction

We provide an automatic and a systematic stepwise approach for transforming specification requirements into UPPAAL models. We first start with a preprocessing phase for unifying grammatical notations, e.g. unification of the letter case of attributes, and for defining a basis for modelling. Once the preprocessing is complete, all the requirements are translated systematically into automata that can model them. At the end of the translation we obtain an UPPAAL model, a network of (timed) automata, representing all the requirements.

a) Declaration part: Given a set of interface requirements, we get a set of variables. We translate each signal into a variable with the same name and definition domain. This part will be completed by the declaration of the channels as they are being created.

b) Automata generation: They are incrementally built by addressing all functional requirements (specification points and constraints). To do this, the translation procedure relies on an interpretation function denoted $[\![.]\!]$ that translates each textual item to a corresponding UPPAAL item. To build the global model we provide a systematic stepwise procedure:

1) During the first phase, the main automaton is derived from the state-driven requirements. To this end, we have associated each pattern with its semantics in UPPAAL formalism that can formally capture it. From each requirement, elements of the automaton are derived by transforming all patterns which forms the requirement, one after the other in the order of their appearance. The details of the translation of all the patterns are presented in Table I. Note that in the description, the parts already generated and which do not change from one step to another are greyed. In particular, we see each state of a requirement is translated to a location with the same name, a condition to a guard, etc. Note that an event is generated and initiated each time a condition is built, not only when a trigger event takes place. Indeed, the requirements specification is based on an eager semantics: transitions are fired as soon as they are enabled. However, UPPAAL considers all transitions as lazy: when a transition is enabled, the system can choose to react immediately or to wait. To cope with this issue, we declare all generated synchronisation channels as urgent, to enforce that the transitions will be taken immediately when the condition is satisfied. Moreover, when the translation does not generate a synchronisation event, to ensure the immediate execution of the transitions we resort to a modelling artefact with the use of the special event denoted **now** emitting over a broadcast channel. At the end of this step, all state-driven requirements are addressed and an initial automata is then built.
2) During the second phase, the model is complemented by transitions and states representing the trigger events that may occur. In particular, we see an event-driven pattern is translated into a corresponding synchronisation channel, which completes a previously generated transition.
3) The model is also complemented in a third phase through the treatment of action-driven requirements. The side effect expressions related on certain transitions are

completed (using the ⊎ operator) to produce a complete update (complete actions to be executed when these transitions are fired). At the end of these three steps, a complete automata is built.

4) The fourth phase is the pruning phase that deals with constraint-type requirements. These requirements modify the structure of the resulting automata, by pruning the non-valid transitions. Indeed, as this kind of requirements explicitly express undesirable situations. Then, based on them, the model is modified and corrected to meet their contacts. After all constraints have been applied, the final automaton is built.
5) During the final phase, other automata of the global model are generated. They are derived from the last family of requirements, those that describe when certain events are issued. These requirements are in a limited number and they have generic templates. To each template we associated a generic model which is instantiated when a matching requirement is met.

Once the translation procedure is complete, if it is possible to generate a valid result (model), this constitutes evidence that the set of requirements is consistent, correct, and feasible. This initial model may be supplemented, if necessary, by additional information that will give the engineers reference points for missing requirements. Only the properties that cannot be enforced by design need for a posteriori verification.

The different processes of our framework are automatic, except the preprocessing phase that is performed in cooperation with the domain experts in charge of system design.

V. Case Study: Automatic Park Assist

To illustrate our approach, we use as example a feature available in almost all self-driving cars. The advanced driver-assistance systems (ADAS) are a key underlying technology in emerging autonomous vehicles [28]. They include several functions for automated driving, among them the Automatic Park Assist (APA) function that assists the drivers (see Fig.3) with parking safely and accurately. APA function provides easy parking by identifying sufficient parking spaces and steering the vehicle into it. The parking system can be supervised by the driver, who can override the operation pushing the accelerator pedal or the brake pedal. It can be fully automatic by an activation of the driver on the control broad, then APA function fully takes over control of parking functions, including steering, braking, shifting, and acceleration, to assist drivers in parking. To position the vehicle for parking, it gathers information from different types of sensors, such as ultrasonic sensors, lidar, camera, evaluate the situation. It subsequently sends control signal to actuators. When these latter receive control signal, active steering or braking subsystem execute instructions effectively and efficiently. Information is exchanged between components by updating the signals shared between them.

The specification of this function is defined in a STRComp including 400 textual requirements, but also requirements in the form of use-cases, tables and graphics. We carefully

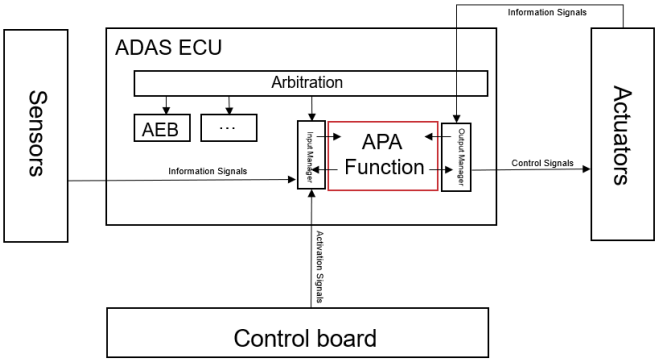


Fig. 3: Interactions between APA function and other components

studied the STRComp documents. They provide the overall description of the function, including details about interactions with other subsystems. Our study reveals that 70% of the requirements gleaned from the documents are interaction-specific while the remaining 30% are functional-specific. Their analysis helped in identifying 41 signals: 16 originate from the SONAR sensors, 17 from the Around View Monitor (AVM) system, and 18 move towards various actuators such as the steering and the braking. We will illustrate our explanation of the different phases through examples of requirements.

Declaration part: Let us consider the interface requirement `REQ_01` that defines the incoming signal `pgen_APA_Failure` and its possible values `vgen_NoFailure` and `vgen_Failure`.

```
REQ_01: APA system shall process and receive from SONAR
        the signal pgen_APA_Failure with the following values:
              - vgen_NoFailure
              - vgen_Failure
```

We define signals within UPPAAL tool as variables. For practical purposes, we encoded all symbolic values of signals by numerical values (int type). Indeed, UPPAAL does not support type String. Fig. 4 shows a small portion of the interface showing the declarations, we see the signal `pgen_APA_Failure` can have values 0 or 1 corresponding to `vgen_NoFailure` or `vgen_Failure` respectively.

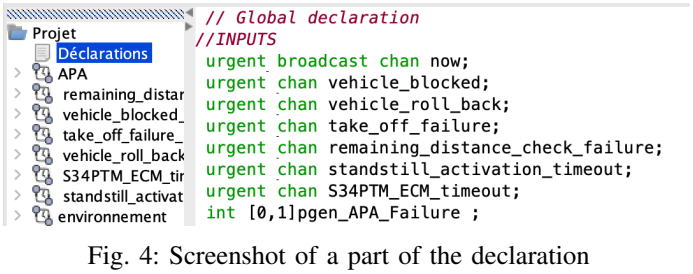


Fig. 4: Screenshot of a part of the declaration

Automata construction: The analysis of all the functional requirements reveals that the function consists of six states (positions) `in_maneuver`, `out_maneuver`, `safe_state_1`, `safe_state_2`, `safe_state_3` and `safe_state_4`

TABLE I: Patterns and their associated model excerpts

	Requirement patterns	their associated semantics
State-driven #1	(a) **while** ⟨*state*⟩ **and** ⟨*condition*⟩	$guard = [\![condition]\!]$ $\langle state \rangle \xrightarrow{guard}$
	(b) **when** ⟨*trigger*⟩	$event = [\![trigger]\!]$ $\langle state \rangle \xrightarrow{guard\ event?}$ urgent broadcast chan now $\langle state \rangle \xrightarrow{guard\ now!}$
	(c) ⟨*system*⟩ **shall** **switch to** ⟨*state*⟩ ⟨*action*⟩	$\langle state \rangle \xrightarrow{guard\ event?} \langle state \rangle$ $update = [\![action]\!]$ $\langle state \rangle \xrightarrow[update]{guard\ event?}$
Event-driven #2	**when** ⟨*trigger*⟩, ⟨*system*⟩ **shall switch to** ⟨*state*⟩	Let S be the set of all states generated in step #1 $event = [\![trigger]\!]$ For all $s \in S, s \xrightarrow{event?} \langle state \rangle$
Action-driven #3	**when entering** ⟨*state*⟩, ⟨*system*⟩ **shall** ⟨*action*⟩+	$update' = [\![action+]\!]$ for all transition $\xrightarrow[update]{guard\ event?} \langle state \rangle$ $\xrightarrow[update \uplus update']{guard\ event?} \langle state \rangle$

which are modelled in UPPAAL by six locations. These locations combined with the different signal values give several thousand of actual states.

To illustrate how the procedure operates let us apply the translation rules on a small subset of requirements.

1) Let us start with the following requirement:

```
REQ_02: while APA system is in Safe_State_2 and
        pgen_Ramp = vgen_on,
        APA system shall switch to Out_Maneuver
```

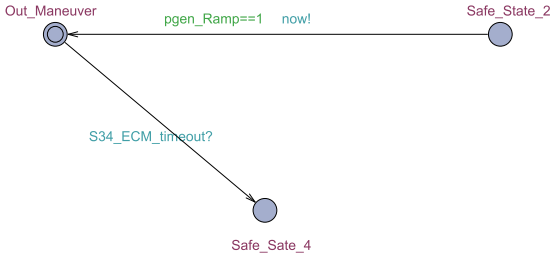
that is a state-driven requirement by applying the appropriate rule, the translation generates the following transition:

Observe the use of the urgent broadcast synchronisation channel **now** to enforce the immediate crossing of the transition when the condition is satisfied.

2) The translation of another requirement of the same type:

```
REQ_03: while Out_Maneuver, when S34_ECM_timeout,
        APA system shall switch to Safe_State_4
```

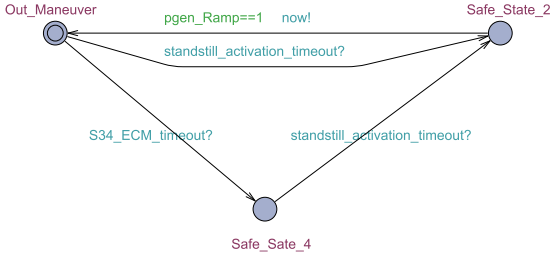
generates another transition from the state `Out_Maneuver` to `Safe_State_4`:



3) Then, let us consider the following event-driven requirement:

```
REQ_04: when standstill activation timeout,
        APA system shall switch to Safe_State_2
```

The translation of this requirement completes the model as follows:

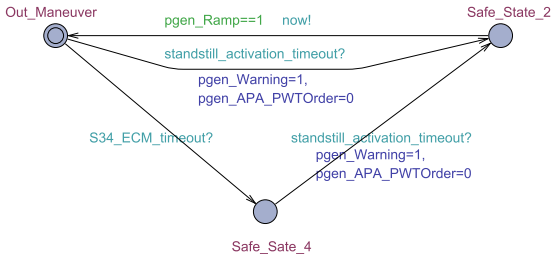


Observe that a transition going from all already generated states (`Out_Maneuver` and `Safe_State_4`) to the state `Safe_State_2` and labelled with a triggered event is generated.

4) Let us now consider an example of action-driven requirement:

```
REQ_05: when entering Safe_State_2  APA system shall:
        - set pgen_Warning to vgen_Alert
        - release powertrain control
```

its translation allows the completion of the action part as shown in the following figure:

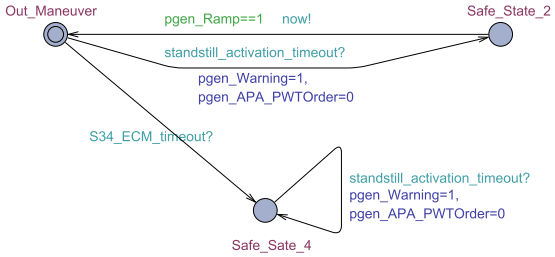


Observe that two actions are added to the two transitions entering the state `Safe_State_2`: – (`pgen_Warning`=1) is a direct translation of the first action; while the action – (`pgen_APA_PWTOrder`=0) results from the treatment of another requirement (which is not detailed here).

5) Once all requirements have been processed and the elements of the automaton created, the constraint requirements are applied. To show how they operate on the model, let us consider the following constraint:

```
REQ_06: if APA system is in Safe_State_4 and entrance
        conditions to Safe_State_2 are satisfied,
        APA system shall switch to Safe_State_4
```

The model is then transformed into the following:

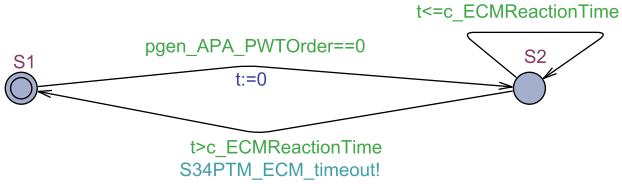


Observe that as specified by the constraint the transition going to state `Safe_State_2` is redirected to state `Safe_State_4`.

6) Finally, let us consider the following requirement that allows the triggering of the event `S34_ECM_timeout`:

```
REQ_07: APA system shall detect,
        if pgen_APA_PWTOrder = vgen_NotRequested
        for more than c_ECMReactionTime
```

the result of its translation is the following timed automaton which completes the main automaton:



The outcome of the translation of the APA system requirements is a collection of 8 automata: one main automaton which models the behaviour of the function, and additional automata modelling all the events that cause the evolution of the behaviour. The main automaton is given in Fig. 5. Although it appears to be small in size, the number of states obtained by product with the seven other automata (which are similar to the automaton corresponding to the requirement `REQ_07`) is around 37 thousand states.

Building a valid model offers an early assurance of the correctness and consistency of requirements. In addition to provide evidence that the system is realisable, the generated model from a set of requirements can be used for a posteriori verification. It may also be used as an aid to deep understanding in early phase requirements engineering. With this in mind, we used the generated model to check some properties.

First, we started by verifying (using UPPAAL model checker) usual properties such as the verification of deadlock freedom which is essential when combining concurrent components. This property is expressed in CTL language as:

- A[] NOT DEADLOCK

This property is evaluated to TRUE meaning that the model is deadlock-free.

Next, we proved a formula that checks the reachability of all system states. For example, the property:

- E <> APA.IN_MANEUVER

expressing that there exists at least one path starting from initial state along which state IN_MANEUVER will be reached. The property holds for this state and for all others, except for `Safe_State_3` and `Safe_State_4`. Not surprisingly, indeed these two states are reachable by firing condition dealing with signals updated by an external component, typically the SONAR. However, we did not include this component in our model, so the update will never be performed and therefore the condition will never be enabled.

Afterwards, we proved properties related to the system but not described in the requirements specification. For example, having the knowledge about the value of certain signals when the function is in a given state, we check whether this property is complied with. For instance, we know that the Flashing Indicator and the Braking signals have to be available only when the system is `In_maneuver` state. By using the corresponding signals in the model this can be expressed by safety properties as follows:

- A[] APA.OUT_OF_MANEUVER IMPLY PGEN_APA_FLASHINGINDICATORREQUEST_IN==0
- A[] APA.OUT_OF_MANEUVER IMPLY PGEN_APA_BRAKEWHEELTORQUEORDER_APAPARK==0

Both properties are evaluated to TRUE meaning that the specification as defined is compliant with expected behaviour, that is to say that the set of requirements covered describe the expected behaviour.

Discussion: This pilot study shows the potential of the proposed approach. Our approach is based on predefined set of templates, but it can accommodate additional templates for requirements specification, provided that they are associated with automata semantics. However, to be adopted in an industrial context some issues remain and need to be

addressed. These include the effectiveness of the approach and the use of the analysis results, that might be addressed in future work. Generated models can be used to help discussion and to explore and learn about the engineer's needs. The information provided by UPPAAL model checker to the users for feedback, including animations, simulations, and derived counter-examples, assists in this regards. However, to apply remedial measures if needed, it is necessary that the negative feedback be viewed on textual representation of the system requirements.

The scalability issues in the context of industrial verification still need to be resolved. The approach presented in the paper pursues this objective by attempting to reduce the time and cost of the verification phase. The environment or context in which a system will run is often not taken into account. Reasoning about these aspects and their integration into the tool is among the point that requires further work. It is necessary to model not only the behaviour of the target system but also the environment interacting with the system. In UPPAAL, contrary to NuSMV model checker, the system variables cannot change via external interactions with the environment. So in order to simulate the system there is a need to model the environment. The typical approach to model the environment is to build an automaton which updates the values of all signals used by the system non-deterministically. Although it allows the exhaustive control of all possible execution sequences. It tends to generate a large number of instances of a single model and consequently leads to state-explosion. There is need to other approach that enables to filter out uninterested input values from all possible values.

The applicability of our approach naturally depends on the size of the generated models and is therefore limited by the capacity of the model checker used. In this work, we have used a symbolic model checker precisely to challenge this limitation. We are aware that there are solutions that can be used to alleviate the problem of state explosion, such as the adoption of a compositional analysis approach or the use of modular model checking algorithms.

VI. Related Works

Plenty research works for the requirements analysis have been presented in the scientific literature, most of them focus on new or improved techniques for evaluating the quality of requirements. They look for ill-formedness or errors in requirements, where an error can be inconsistency, incompleteness or ambiguity [8], [27], [30]. There have been very few tools that support such analysis on real-world applications and in an industrial setting. The lack in tooling is partly due to the use of natural language. Indeed, natural language is the dominant form of expression of requirements in practical projects in industry. Such approaches face a fundamental challenge: writing requirements and designing system requires a high degree of precision and accuracy, but natural language is inherently imprecise.

Significant efforts, including both research efforts and industrial products development, have been made to improve the techniques for analysing the quality of requirements. Some approaches use partially formalized notations or semi-formal languages such as Doors [2], Reqtify [1], and SysML [16]. Doors and Reqtify are widely used in industry, they benefit from mature tools, e.g. [4] and [3] developed by major companies, IBM Rational and Dassault Systems respectively. In both cases the focus is not on analysing requirements and verification methodology, but on managing requirements.

Another approach introduces the notion of Constrained Natural-Languages (CNL) such as EARS [24], RELAX [31] and Stimulus [20] languages. These languages with a simplified syntax and restricted lexical terms help to bridge the gap between informal and formal representation of requirements, and improve their translation from informal to formal. Such research is often accompanied by proofs-of-concepts or pilot studies that show the potential of the proposed ideas. Some research, such as the approach described in the paper [17] is concerned with the translation of requirements into properties that can be used with finite-state verification tools, while our approach aims at building state machines for verification of functional properties. To the best of our knowledge, only Stimulus is supported by a tool [21], both the language and the tool have the same name. It enables engineers to generate test vectors and test objectives automatically, that can be used to check whether the developed system is compliant with its specification. Our approach is similar to to the stimulus approach, in the sense we generate state machines from CNL templates. An important difference from this approach, we aim at the transformation of system requirements into a model, that provides early evidence that the requirement specifications are realizable, as opposed to the test objectives that the system should meet.

Formal methods have proven their cost-effectiveness through their successful use in industrial context and in different areas, such as railway [13], aeronautics and aerospace [9]. There exist a number of approaches relying on mathematical theories of graphs and automata for requirements analysis. Such approaches use graphical notations, such as infinite automata, state diagrams and statecharts, to specify requirements. However, these representations make their use less practical on real-world applications, in particular in an industrial setting. Requirements can use other mathematical theories for system modelling and analysis, for example Event-B [5] which is equipped with its own methodology based on set theory and predicate logic for modelling and to formally prove system correctness. This kind of mathematically rigorous tools, while they are powerful, are intended to specialists and engineers with deep understanding of their mathematical foundations and, also, applicable domains and limits.

In recent years the early validation of requirements and the use formal methods has become a focus for industrial research such as as mentioned in [10], [25] and [29]. Such empirical research needs to be supported by tools that can be used on real-world applications and in an industrial setting. In particular, the automotive industry has expressed interest

in using such approach in developing complex automotive systems.

VII. Conclusion

This paper introduced a systematic process for building models from automotive requirements written in natural language with the aim to reduce the effort of testing and detects fixes late in software development lifecycle. The process is automated through a tool and existing verification tool.

The first effort led to categorize the entire set of requirements of the system under design in order to formalize them. We provided a proof of concept regarding our approach by designing models and verifying some properties. We verified general properties using UPPAAL such as liveness and reachability. Naturally our next goal is to find some new type of properties to be verified in order to validate safety critical aspect and also detect flaws. Another goal can be to propose a requirement specific language for the expression of requirement in the automotive domain to match standard such as AUTOSAR [18]. In [32], we have given some directions as well limited to a smaller set of requirements. This language is different from the languages that already exist for the description of automotive standards, as [15] and [11]. In the sense, it allows the specification of systems at a high abstraction level, without any prior knowledge about architectural considerations and how the functions are then allocated to the components of the physical architecture.

References

[1] Dassault system–reqtify. https://www.3ds.com/fr/produits-et-services/catia/produits/reqtify/.

[2] Ibm–rational doors. http://www-03.ibm.com/software/products/ratidoor.

[3] The reuse company–rat. https://www.reusecompany.com/rat-authoring-tools.

[4] The reuse company–rqa. https://www.reusecompany.com/rqa-quality-studio.

[5] Jean-Raymond Abrial. *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, USA, 1st edition, 2010.

[6] AFNOR. National Survey: The costs of poor quality in industry. Technical report, AFNOR, October 2017.

[7] Thomas Baar. Verification Support for a State-Transition-DSL Defined with Xtext. In *Lecture Notes in Computer Science, vol 9609*, pages 50–60. Springer, 06 2016.

[8] Daniel M. Berry and Erik Kamsties. *Ambiguity in Requirements Specification*, pages 7–44. Springer US, Boston, MA, 2004.

[9] Jean-Louis Boulanger. *Industrial Use of Formal Methods: Formal Verification*. ISTE Ltd. Wiley, 2013.

[10] Marco Bozzano, Alessandro Cimatti, Joost-Pieter Katoen, Panagiotis Katsaros, Konstantinos Mokos, Viet Yen Nguyen, Thomas Noll, Bart Postma, and Marco Roveri. Spacecraft early design validation using formal methods. *Reliability Engineering & System Safety*, 132:20–35, 2014.

[11] Stefan Bunzel. AUTOSAR the Standardized Software Architecture. *Informatik-Spektrum*, 34:79–83, 2011.

[12] Robert N Charette. *Software engineering environments : concepts and technology*. Intertext Publications, New York, NY, 1986.

[13] X. Chen, Z. Zhong, Z. Jin, M. Zhang, T. Li, X. Chen, and T. Zhou. Automating consistency verification of safety requirements for railway interlocking systems. In *2019 IEEE 27th International Requirements Engineering Conference (RE)*, pages 308–318, 2019.

[14] B. H. C. Cheng and J. M. Atlee. Research directions in requirements engineering. In *Future of Software Engineering (FOSE '07)*, pages 285–303, 2007.

[15] Philippe Cuenot, Patrick Frey, Rolf Johansson, Henrik Lönn, Yiannis Papadopoulos, Mark-Oliver Reiser, Anders Sandberg, David Servat, Ramin Tavakoli Kolagari, Martin Törngren, and Matthias Weber. *The EAST-ADL Architecture Description Language for Automotive Embedded Software*, pages 297–307. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[16] Michel dos Santos Soares and Jos L. M. Vrancken. Requirements specification and modeling through sysml. *2007 IEEE International Conference on Systems, Man and Cybernetics*, pages 1735–1740, 2007.

[17] Matthew B. Dwyer, George S. Avrunin, and James C. Corbett. Patterns in Property Specifications for Finite-State Verification. In Barry W. Boehm, David Garlan, and Jeff Kramer, editors, *Proceedings of the 1999 International Conference on Software Engineering, ICSE' 99, Los Angeles, CA, USA, May 16-22, 1999*, pages 411–420. ACM, 1999.

[18] Simon Fürst and Markus Bechter. AUTOSAR for connected and autonomous vehicles: The AUTOSAR adaptive platform. In *DSN Workshops*, pages 215–217. IEEE Computer Society, 2016.

[19] IEEE. Systems and software engineering – Life cycle processes – Requirements engineering. *ISO/IEC/IEEE 29148:2011(E)*.

[20] Bertrand Jeannet and Fabien Gaucher. Debugging Real-Time Systems Requirements: Simulate The "What" Before The "How". In *Embedded World Conference, Nürnberg, Germany*, 2015.

[21] Bertrand Jeannet and Fabien Gaucher. Debugging Embedded Systems Requirements with STIMULUS: an Automotive Case-Study. In *8th European Congress on Embedded Real Time Software and Systems (ERTS 2016)*, Toulouse,France, 2016.

[22] Herb Krasner. The Cost of Poor Quality Software in the US: A 2018 Report. Technical report, CISQ Consortium for IT Software Quality, September 2018.

[23] Kim G. Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a Nutshell. *International Journal on Software Tools for Technology Transfer*, 1(1-2):134–152, December 1997.

[24] A. Mavin, P. Wilkinson, A. Harwood, and M. Novak. Easy Approach to Requirements Syntax (EARS). In *2009 17th IEEE International Requirements Engineering Conference*, pages 317–322, 2009.

[25] Steven P. Miller, Alan C. Tribble, Michael W. Whalen, and Mats P. E. Heimdahl. Proving the shalls: Early validation of requirements through formal methods. *International Journal on Software Tools for Technology Transfer*, 8(4):303–319, 2006.

[26] Steven P. Miller, Alan C. Tribble, Michael W. Whalen, and Mats Per Erik Heimdahl. Early validation of requirements through formal methods. *International Journal on Software Tools for Technology Transfer*, 8(4-5):303–319, 2006.

[27] Paul Rayson, Ieee Computer Society, Ken Cosh, and Ieee Computer Society. K.: Shallow Knowledge as an Aid to Deep Understanding in Early Phase Requirements Engineering. *IEEE Transactions on Software Engineering*, pages 969–981, 2005.

[28] Y. Song and C. Liao. Analysis and review of state-of-the-art automatic parking assist system. In *2016 IEEE International Conference on Vehicular Electronics and Safety (ICVES)*, pages 1–6, 2016.

[29] Emmanouela Stachtiari, Anastasia Mavridou, Panagiotis Katsaros, Simon Bliudze, and Joseph Sifakis. Early validation of system requirements and design through correctness-by-construction. *Journal of Systems and Software*, 145:52–78, 2018.

[30] Kimberly S. Wasson. A Case Study in Systematic Improvement of Language for Requirements. In *Proceedings of the 14th International Requirements Engineering Conference*, pages 6–15. IEEE Computer Society, 2006.

[31] Jon Whittle, Peter Sawyer, Nelly Bencomo, Betty H. C. Cheng, and Jean-Michel Bruel. RELAX: Incorporating Uncertainty into the Specification of Self-Adaptive Systems. *2009 17th IEEE International Requirements Engineering Conference*, pages 79–88, 2009.

[32] Assioua Yasmine, Ameur-Boulifa Rabea, and Guitton-Ouhamou Patricia. Towards Formal Verification of Autonomous Driving Supervisor Functions. In *10th European Congress on Embedded Real Time Software and Systems (ERTS 2020)*, Toulouse, France, January 2020.

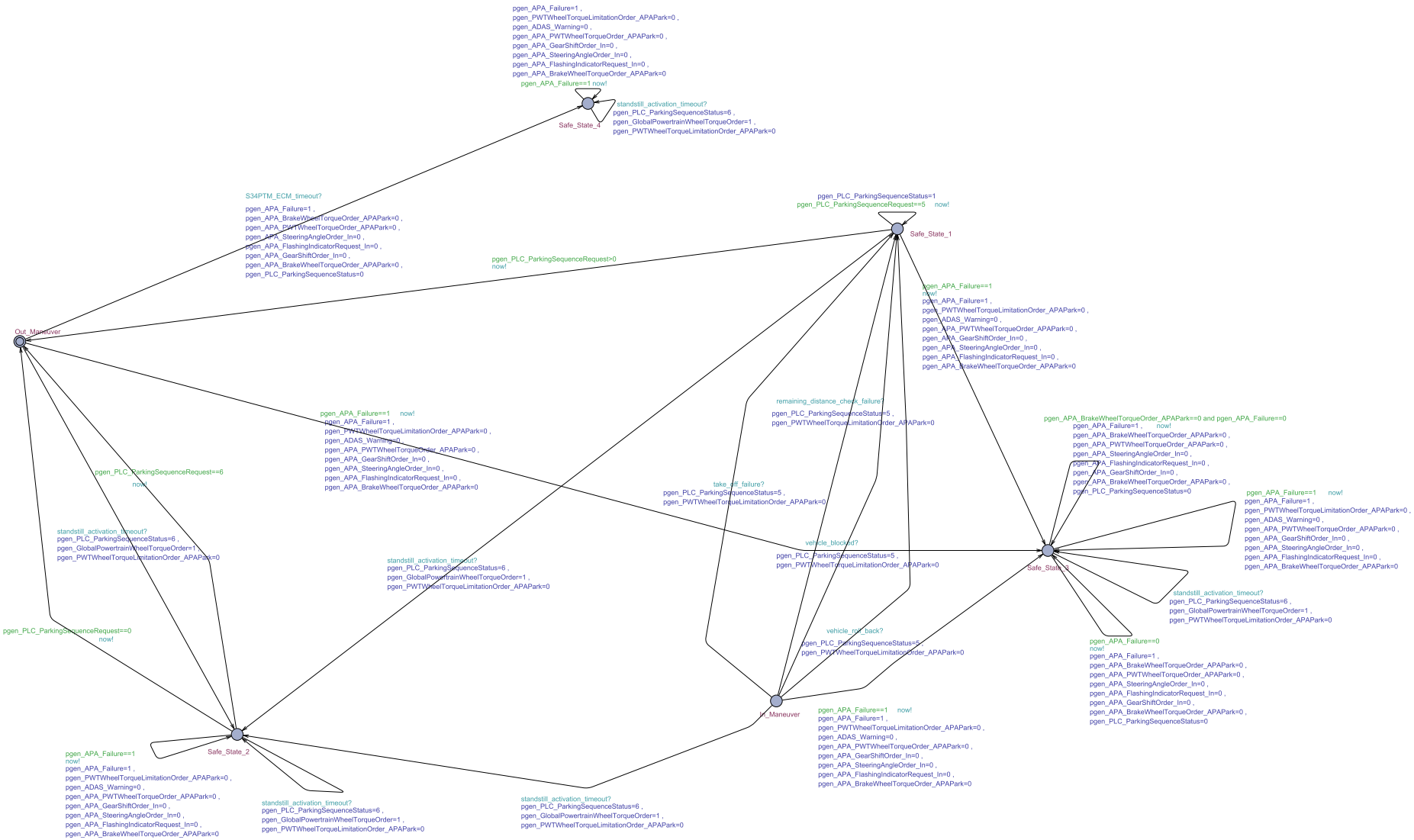


Fig. 5: Main automata for Automatic Park Assist function

Property Expression and Verification in an Incremental Model Development Framework: a Case Study

Thomas Lambolais and Anne-Lise Courbis,
EuroMov DHM, IMT Mines Ales, Univ Montpellier,
thomas.lambolais@mines-ales.fr, anne-lise.courbis@mines-ales.fr

***Abstract*—The IDCM framework (Incremental Development of Conforming Models) supports incremental constructions and evaluations of UML behavioral models (architecture of components and state machines). This framework evaluates models with respect to *implicit* temporal safety and liveness properties. The specifiers and designers only describe models, they don't need to write down explicit temporal logic properties. In this paper, we show how explicit safety and liveness properties can also be described using semi-formal boilerplates, translated into classical temporal logic formulas which are themselves translated into testing transition systems.**

Adding evaluation means to model-based development (relation checking) is a way to develop models incrementally, following a spiral-based development cycle.

I. Introduction

Our goal is to assist designers during modeling tasks of reactive systems. We consider two aspects: describing the history of an architectural model construction by a sequence of modeling steps; offering assistance during this history, by providing evaluation and construction techniques based on formal relations.

In [26], we proposed a set of architectural construction techniques which (i) contribute to architectural model qualities through well-known design principles in software engineering (separation of concerns, hierarchy and information hiding); (ii) include formal verifications for early detection of behavioral issues, i.e. safety and liveness problems.

In this paper, we extend this approach and show how typical informal temporal properties, described by boiler plates, can be translated into Labelled Transition Systems and verified by our framework.

The article is structured as follows. Section 2 presents the main incremental paradigms, safety and liveness concerns and incremental relations. This section also presents a semantics of UML architectures. Section 3 presents the way we can define and verify explicit safety and liveness properties. A case study is given in Section 4, on which two properties are specified. Comparison with existing work is given in Section 5. We conclude in Section 6.

II. Fundamentals of incremental modeling and UML composite component semantics

By *incremental* modeling, we mean that models are progressively developed. They may be refined or abstracted, and extended or restricted [26], [27]. At each step, the new model is compared to the previous one through a suitable relation, which focuses on behavioral and temporal aspects.

As shown by Alpern and Schneider [35], all temporal properties can be seen as a conjunction of safety and liveness properties. The relation used to compare models is chosen among a set of relations which can be interpreted with respect to the way they *preserve* safety and/or liveness properties. We say that a property ϕ is preserved by a relation $\mathcal{R}$ if, for any two models P and Q such that $P\ \mathcal{R}\ Q$, $P \models \phi \Rightarrow Q \models \phi$.

We observe safety and liveness properties by means of the *interactions* of the system with its environment. These interactions consist in accepting an *event* (signal or operation reception), or performing an *action* requiring a signal send or operation call. A trace is a partial sequence of observable interactions starting from the initial state.

The LTS (Labeled Transition Systems) semantics we give to UML primitive components behavior state machines is not recalled here, see [16], [27]. Let us briefly recall that an LTS P is a tuple $\langle \mathcal{P}, Act, \rightarrow, P \rangle$ where:

- $\mathcal{P}$ is a set of state names,
- Act is a set of action names,
- $\rightarrow \subseteq \mathcal{P} \times Act \times \mathcal{P}$ is a set of labeled transitions between states,
- P is the initial state.

The tool we have developed includes UML State Machine transformation into LTS [15]. Here follows an intuitive presentation of incremental relations, and a proposal for the UML composite component semantics.

A. Incremental relations between UML components.

Based on classical trace inclusion $\sqsubseteq_{\mathsf{MAY}}$ and conformance relation conf [28], the IDCM framework implements several *incremental* relations. We only give an intuitive presentation here of conf, $\sqsubseteq_{\mathsf{INC}}$, $\sqsubseteq_{\mathsf{REF}}$ and $=_{\mathsf{REF}}$, which will be used in sections III and IV. Refer to [16], [27], [33] to find formal definitions and an extensive presentation of other incremental relations. Given two models M_1 and M_2, $M_2 \sqsubseteq_{\mathsf{MAY}} M_1$ means that M_2 traces are included into M_1 traces. It ensures that M_2 satisfies any safety property of M_1: indeed, M_2 must refuse all what M_1 must refuse.

M_2 conf M_1, or M_2 *conforms to* M_1, if after any trace of M_1, M_2 must accept every action that M_1 must accept.

It ensures that M_2 is more deterministic than M_1. This relation guarantees that any liveness property of M_1 is satisfied by M_2. The conformance relation is seen as an implementation relation. However, this relation is not transitive and cannot be used as such for incremental developments.

$M_1 \sqsubseteq_{\mathsf{INC}} M_2$, or M_2 *increments* M_1, if any model which conforms to M_2 also conforms to M_1. In particular, $M_1 \sqsubseteq_{\mathsf{INC}} M_2 \Rightarrow M_2$ conf M_1, and $\sqsubseteq_{\mathsf{INC}}$ is a transitive relation.

$M_1 \sqsubseteq_{\mathsf{REF}} M_2$, or M_2 *refines* M_1, if $M_1 \sqsubseteq_{\mathsf{INC}} M_2$ and $M_2 \sqsubseteq_{\mathsf{MAY}} M_1$. Hence, $M_1 \sqsubseteq_{\mathsf{REF}} M_2$ guarantees that liveness and safety properties of M_1 are also satisfied by M_2. $=_{\mathsf{REF}}$ is the equivalence relation associated to $\sqsubseteq_{\mathsf{REF}}$.

The verification algorithms to check these relations have been implemented within the IDCM tool.

B. UML composite components semantics.

UML composite components describe architectural systems in terms of UML component instances, linked between themselves by assembly connectors and connected to the outside environment by delegation connectors. We give a semantics of UML composite component behaviors on parallel compositions of processes in the EXPOPEN process algebra [21]. EXPOPEN shares the same concepts as basic LOTOS process algebra [28]. Secondly, EXPOPEN models are translated into LTS by the CADP tool [21].

For instance, fig. 1 presents a UML architecture (A_0) which models an automotive front-light system (see details in section IV). In the architecture A_0, there are three component instances linked by assembly connectors c3 and c4. There are three delegation connectors c1, c2 and c5 that link ports of the architecture to ports of its components. Ports linked by a connector share the same UML interface.

The external interfaces of A_0 correspond to actions exchanged on delegation connectors c1, c2 and c5. These actions must be observable to verify the properties of the system, while the set of synchronized actions on assembly connectors c3 and c4 are not on the focus of verification and will be hidden.

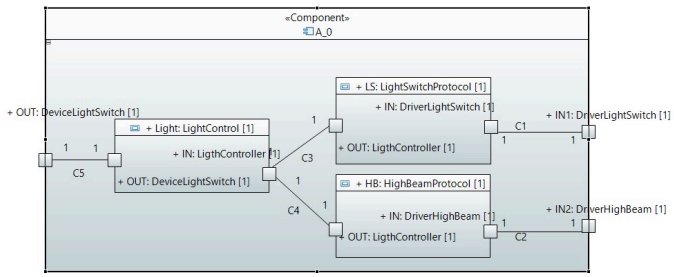


Fig. 1. A UML composite component (A_0)

III. PROPERTY DEFINITION AND VERIFICATION

Let a and b be two actions ($a, b \in Act$). We illustrate here two patterns of temporal properties to be satisfied by a model M, for safety (*S*) and liveness (*L*) properties:

(*S*) *"In any $\langle a \rangle$-circumstances, action b is never possible."*:

- an $\langle a \rangle$-circumstance means that M offers action a, what is formally written, in Hennessy Milner Logic (HML): $M \models \langle a \rangle tt$
- b is not possible is written, in HML: $M \models [b]ff$
- Hence, using usual logical implication $\Rightarrow$ and modal operator $\Box$ (always) for convenience, property (*S*) corresponds to:

$$M \models \Box(\langle a \rangle tt \Rightarrow [b]ff)$$

- This is a general expression of a safety property. Derived expressions of this kind are simply "*b is never offered*" or "*b is always possible*", since action b may also be a positive statement: $M \models \Box\langle b \rangle tt$
- For example, "*In any circumstances, the flash action is always offered*".

(*L*) *"In any circumstances, every a-action leads eventually to b."*:

- Using $\Box$ and $\Diamond$ (eventually) operators, such properties can be formally written:

$$M \models \Box([a]\Diamond(\langle b \rangle tt \wedge [Act - b]ff)$$

– A property $[a]F$ states in Hennessy-Milner logics that, for all processes after a, F is true. If there is no a-successor, this property is true whatever the value of F.

– $\langle b \rangle tt \wedge [Act - b]ff$ is true when there exists a b-successor, and no other successor.

- This is a general expression of liveness property: after any a-action, b will be done.
- For example, "*When the system is in headlamp mode (low or high beam), switching back to side lights switches off high beams and low beams*".

A. Verification of safety properties (S)

1) General case: Properties of kind (*S*) lead to:

$$M \models \Box([a]ff \vee [b]ff).$$

Such safety properties define unwanted action. We define the LTS T (using '+', '.' operators and recursion textual notations for convenience), which accepts at any time actions a or b:

$$T = a.T + b.T$$

Then, we observe the set of systems $ObsSet$ after any trace of M:

$$ObsSet = \{M' \mid M \stackrel{\sigma}{\Longrightarrow} M', \forall \sigma \in Tr(M)\}$$

Let us recall that $Tr(M)$, the set of traces of M, are all the sequences of observable actions starting from the initial state, and that $M \stackrel{a_1 \cdots a_n}{\Longrightarrow} M' = M \xrightarrow{\tau^*} \xrightarrow{a_1} \xrightarrow{\tau^*} \cdots \xrightarrow{\tau^*} \xrightarrow{a_n} \xrightarrow{\tau^*} M'$.

If the safety property (*S*) is satisfied by M, no set of $ObsSet$ should simulate T:

$$\forall M' \in ObsSet.\ T \not\sqsubseteq M'$$

The $\sqsubseteq$ relation is the preorder associated to Milner's congruence equivalence. This verification corresponds to a simulation, which is convenient to verify safety properties and the most efficient for our purposes.

2) Particular case for positive statements when there is no premise: We may choose a simpler verification means, when safety properties (*S*) are of the kind:

$$M \models \langle b \rangle tt \wedge \Box \bigcirc \langle b \rangle tt$$

which means that M must always accept b every two actions ($\bigcirc F$ means that F is true in the next state).

We define process

$$T = b.\sum_{a \in Act} a.T$$

T is a process that "always" does b, at least every two following actions, and possibly several times in sequence ($b \in Act$).

In order to check the safety property, we verify that the process T synchronized with M on every actions (operator '||') :

$$T||M$$

is deadlock free. This means that M can always do T actions. Absence of deadlock can be verified in IDCM.

B. Verification of liveness properties (L)

We consider properties of the kind (*L*):

$$M \models \Box([a]\Diamond\langle b \rangle tt \wedge [Act - b]ff).$$

Let $success \notin Act$. We define LTS T and T_b as follows:

$$\begin{aligned} T &= \langle Act - a \rangle.success.T + a.T_b \\ T_b &= b.success.T + \langle Act - b \rangle.T_b \end{aligned}$$

Then, we observe the system M synchronized with T on every actions (operator '||'), where every actions except $success$ are hidden:

$$Obs = \textbf{hide} \;\; Act - success \;\textbf{in}\; (M||T)$$

If the liveness property (*L*) is satisfied by M, Obs should be refinement equivalent to a process $Ok = success.Ok$:

$$Obs =_{\mathsf{REF}} Ok$$

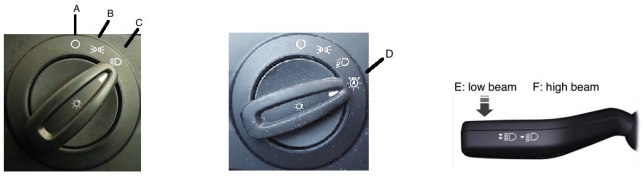
which means that, when all actions are hidden except success, Obs should perform $success$ infinitely often.

IV. Illustration: Adaptive Front-lighting System

We consider a car Adaptive Front-lighting System (AFLS) implemented by several car manufacturers [36]. We have developed an incremental construction which consists in five models (S_0, A_0, ..., A_3), as presented in [26]. In this incremental construction [26], verification relations used between models ensure us that safety and liveness properties are preserved: A_0 has the same safety and liveness properties as S_0, A_1 has the same properties as A_0, and so on. But the question of the safety and liveness properties satisfied by the initial models S_0 still remain. Here, we present S_0 and A_0 which have to fit the following requirements, and show how we can verify such safety and liveness properties:

(*Informal user req.*): the front-lighting system comprises side lamps, low and high beams that the driver chooses according to a precise protocol. There are two driver commands: a manual lighting control position switch (Fig. 2.1) and a low and high beam lever (Fig. 2.3). The lighting control switch offers "off" (A), "side lights" (B) and "headlamps" (C) positions. It is only when this switch is in the C position that the driver can change between the low and high beams with the lever. In any position, the low and high beam lever also offers a flash command.

A_1, not presented here, consider an automatic mode (Fig. 2.2, position D), which switches headlamps on and off, depending on the ambient light. High beams are still manually activated. A_2 and A_3 consider a further improvement with automatic high beams.



(1) (2) (3)

Fig. 2. Driver commands: (1) manual lighting control, (2) lighting control with auto mode, (3) low and high beam lever.

A. S_0 and A_0 models

S_0 is intended to be a primitive component, representing the initial specification, whose behaviour is described by a single State Machine. A_0, representing a possible realization of S_0, is a composite component describing an architecture. Both S_0 and A_0 have the same outside provided and required interfaces (Fig. 3): Driver Light Switch and Driver High Beam correspond to Fig. 2.1 and Fig. 2.3, Device Light Switch is the required interface which commands the lighting device through Driver Light Switch and Driver High Beam interfaces correspond to driver commands of Fig.2.1 and Fig.2.3.

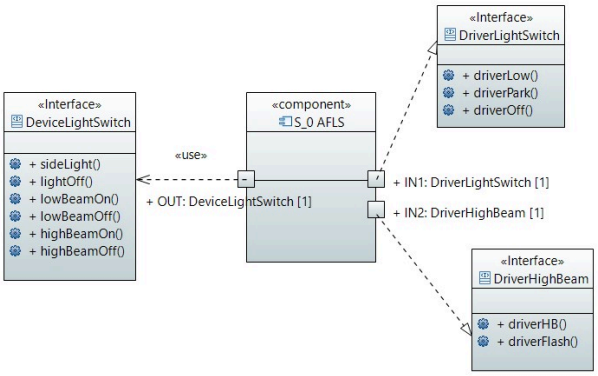


Fig. 3. UML provided and required interfaces for S_0 component

In complement to further models, we developed a Java prototype application, which simulates the AFLS behaviour (Fig. 4). The two driver commands are the two button lines

at the bottom (the low and high beam lever is grey, while the beige one can only go from one position to its successive or preceding position).

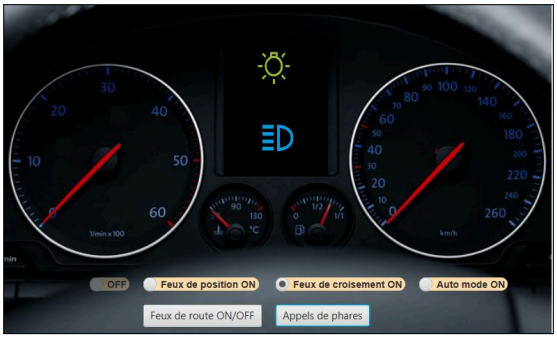


Fig. 4. Example of Graphical User Interface associated to a Java simulation of the AFLS.

The behavioral specification of S_0 (Fig. 5) has two roles: (i) it defines when operations are provided to the driver: in particular, driverHBon and driverHBoff are only possible when the switch is in LowBeam mode; (ii) it translates the driver commands into the lamp device operations: for instance driverLow switches low beams on, but keeps side lights on, driverPark switches side lights on or switches low beams off, and driverFlash effect is described by an activity of two sequenced operations: highBeamOn followed by highBeamOff.

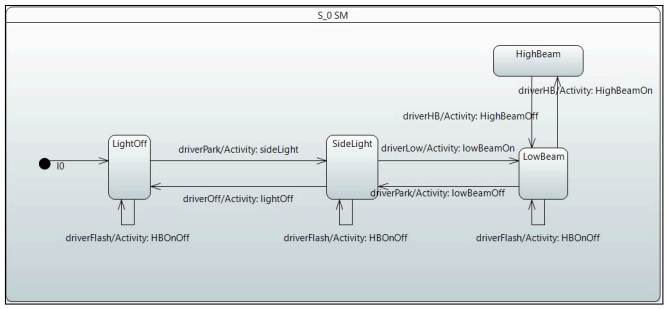


Fig. 5. S_0 state machine

In [26], we described a way to build an architecture A_0 which is a correct refinement of S_0:

$$S_0 =_{\mathsf{REF}} A_0. \quad (1)$$

It leads to A_0 (described in Fig. 1) which connects three primitive components. We give here the state machines of HighBeamProtocol (Fig. 6) and LightControl (Fig. 7).

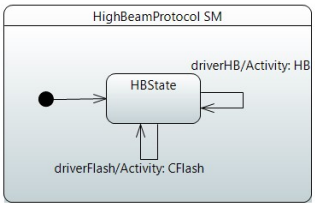


Fig. 6. HighBeamProtocol state machine

B. Specification and verification of typical properties

While equation (1) guaranties that A_0 and S_0 share the same safety and liveness properties, obviously, it does not guarantee that S_0 satisfies the informal requirements (*User informal req.*).

1) Example of safety property: "*The system can always accept* driverFlash *commands every two steps, except in highbeam mode.*" This is the following ϕ safety property:

$$\phi = \Box(\langle \mathsf{driverFlash} \rangle . \langle Act - \mathsf{driverHB} \rangle . \langle \mathsf{driverFlash} \rangle . tt)$$

In order to check this property, we build the LTS presented in Fig. 8, which first tests a driver flash action, and after any other action (driver low, driver park, driver off), tries to perform a driver flash action again.

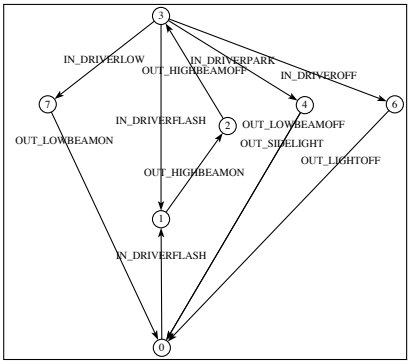


Fig. 8. LTS T to check if S_0 or A_0 can always accept the DRIVERFLASH action, except in high beam mode.

Using model transformation and IDCM, we can check that property ϕ is satisfied:

$$\textbf{hide } \mathsf{driverHB} \textbf{ in } (S_0 \,||\, T)$$

is deadlock-free.

2) Example of liveness property: "*In low/highBeam mode,* driverPark *command always switches off low beams.*" This is the following ψ property:

$$\psi = \Box([\mathsf{driverLow}][\mathsf{driverPark}]\Diamond(\langle \mathsf{lowBeamOff} \rangle tt \wedge [Act - \mathsf{lowBeamOff}]ff))$$

Let $G = Act - \mathsf{driverLow} - \mathsf{driverPark}$. The testing LTS T is:

$$T = \langle G \rangle . success . T + \mathsf{driverLow.driverPark}.T_2$$
$$T_2 = \mathsf{lowBeamOff}.success.T + \langle G - \mathsf{lowBeamOff} \rangle . T_2$$

It appears that

$$\textbf{hide } Act - success \textbf{ in } (S_0 || T) \neq_{\mathsf{REF}} Ok.$$

with $Ok = success.Ok$

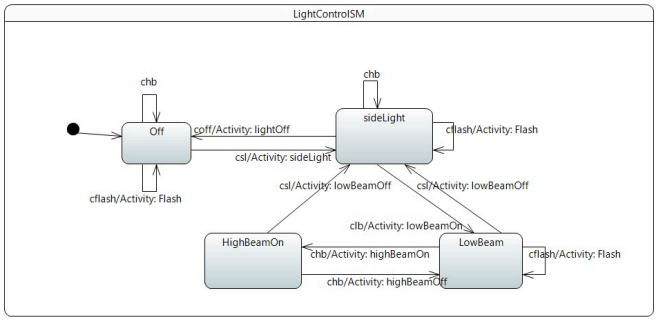


Fig. 7. LightControl state machine

Indeed, when in HighBeam state, S_0 can not perform driverPark followed by lowBeamOff: a transition is missing from HighBeam state to SideLight state, triggered by driverPark.

V. Related Work

Most of the works dealing with verification (model-checking and theorem proving) take formal specifications as inputs. We used CADP toolbox [21], as well as CCS tools such as CAAL [1], [38]. Model checking tools use temporal logics (for instance Hennessy-Milner Logics in CAAL) do describe and verify properties. Compared to such tools, our objective in the IDCM framework is threefold: (i) taking as input UML architectures and state machines, i.e. semi-formal descriptions (ii) describing properties within the same language (iii) providing incremental relations of refinement and extension. In this paper, safety and liveness properties are verified by LTS comparisons and deadlock detection, without having to use temporal logics.

Moreover, the increasing number of works dealing with formal model based analysis [9] do not ensure extension, refinement or substitutability of models [27]. To the best of our knowledge, no work has defined relations for incremental development of architectural models, defined in UML. Table I gives the synthesis of the analyzed approaches along liveness, safety, substitution, extension and refinement aspects.

	Liv.	Saf.	Sub.	Ext.	Ref.
UML/Wright [23]	✓	✓			✓
UML/B [34]	∼	✓	✓		✓
SysML/Interface automata [14]	∼				
UML/omega2 [30]	∼	✓			
AADL/FIACRE [7]	✓	✓			
AADL/BIP[13]		✓			
Archware (LOTOS) [31]	✓	✓			
PADL-Æmilia [2]	✓	✓			
SafArchie [3]		✓	∼		
FIESTA [37]					∼

✓: supported; ∼: partially supported; ' ': not supported;

TABLE I
Evaluation of architectural and verification tools.

[23] proposes a UML profile and translates UML models into Wright for using the model checker FDR. FDR focuses on safety and liveness analyses without fairness assumption. It does not analyze any extension nor substitution relation. Some work such as [34] focus on translating UML into B or Z. They include refinement techniques but do not address extension techniques. [14] considers SysML models in order to verify components assemblies. They perform behavioral compatibility verifications, but do not analyze any liveness property other than dead-lock detection and do not address extension and refinement problems. [30] has extended the analysis techniques proposed by [18] which defined OMEGA2, a UML profile. Architectures are translated into IF/IFx models [10], [11] in order to be analyzed by the CADP toolbox [21] for safety property analysis. However, model substitutability, extension and refinement are not supported.

[7] considers AADL descriptions and transforms them into FIACRE in order to apply the model checker TINA [8]. TINA analyzes safety, liveness and deadlocks under the fairness hypothesis, but it does not address extension, refinement and substitutability. [13] has a similar approach by translating AADL into the BIP language [4]. [17] transforms UML architecture into BIP. However, BIP focuses on safety properties and does not address liveness, extension, refinement, nor substitutability. Archware [32], [31] is a framework based on the LOTOS language allowing the use of the CADP model checker [21]. Safety and liveness properties are analyzed under fairness assumption. Compatibility between components is verified, but no extension nor substitution relations is considered. PADL and Æmilia [6], [2] are languages based on a stochastic process algebra. They are associated with the model checker TwoTowers [5]. Analyses can be conducted according to several bisimulation relations. It appears that these relations are too strong for incremental developments.

SafArchie and TranSAT framework [3] deal with the evolution of architectures using safe patterns. The compatibility between components is addressed from different points of view: structural, functional and behavioral. Substitutability of components is studied from a syntactical point of view by considering interfaces. This does not guarantee the behavioral conformance of the architecture in which the component is substituted. FIESTA [37] defines a generic framework where new components are introduced into architectural models. It is based on a pattern approach and focus on adding or modifying connections in order to ensure the compatibility between components. This work addresses a part of the incremental development in so far as the structural compatibility does not guarantee the behavioral one.

In [20], the authors propose a transformation of UML state charts and communication diagrams in LOTOS and use FOCOVE verification environment where properties expressed by CTL formulas are verified. [24] proposes UML statecharts and their synchronization transformation in LOTOS. No verification is proposed, nor extension and refinement.

[22] transforms UML protocol state machines into Alloy. No temporal properties are taken into account. Protocol state machines are convenient to express predicates on states, which depend on terms and values. We do not support such data verifications. On the opposite, standard Alloy models do not allow temporal logic verifications.

[25] presents a transformation of UML activity diagrams into Alloy. Such work has the same limits as [22] concerning temporal aspects, hence they do not verify liveness properties.

UML activity diagrams are also considered in [19], using model checkers such as: UPPAAL, SPIN, NuSMV and PES. Hence, safety and liveness properties are described in specific temporal logics. Nevertheless, the automated aspect of the Eclipse-plugin implementation of the tool allows users without a background in formal methods to verify the safety and liveness of a system.

[29] presents an interesting transformation of UML components diagram and state machines into timed automata that are checked with UPPAAL tool. [12] also proposes a transformation of UML state machines into timed automata. These work support timed properties, whereas we only consider temporal properties. However, user must provide explicit descriptions of properties using timed temporal logics.

VI. Conclusion

IDCM framework proposes architectural modeling techniques for reactive systems which cover refinement and extension approaches, as well as evaluation means, based on conformance and refinement relations. Such relations verify implicit safety and liveness properties. In this paper, we present patterns of explicit safety and liveness properties and a mechanism to check them on the desired models, using the refinement equivalence relation. This relation has the advantage of being weaker than the traditional observational Milner's relation.

Describing and verifying explicit properties is a complementary means to check: (i) first abstract models; (ii) extension points: in the incremental approach, we check that extension preserves liveness properties, but we were not able to check that a specific safety property is not violated by new behaviours.

This work has several limits. The designer does not need to express safety and liveness properties in a specific temporal logics, but he has to translate such properties into specific LTS. Even if we provide templates, this can be tricky task. Secondly, the UML State Machine translation into LTS does not consider data and timing aspect. We focus on 'pure' actions, without data parameters. Hence, guards, change event and time event in UML state machines are always translated by non deterministic LTS.

Further works consist in improving the IDCM tool on two points: offering a way to describe semi-formally such properties (formal translations being automatically generated); improving verdicts and counter-examples in case a relation or property is not satisfied.

References

[1] L. Aceto, A. Ingólfsdóttir, K. Larsen, and J. Srba. *Reactive Systems: Modelling, Specification and Verification*. Cambridge University Press, 2007.

[2] A. Aldini and M. Bernardo. On the usability of process algebra: An architectural view. *Theoretical Computer Science*, 335(2-3):281–329, May 2005.

[3] O. Barais, E. Cariou, L. Duchien, N. Pessemier, and L. Seinturier. Transat: A framework for the specification of software architecture evolution. *Issues on Coordination and Adaptation Techniques*, pages 31–38, 2004.

[4] A. Basu, M. Bozga, and J. Sifakis. Modeling Heterogeneous Real-time Components in BIP. In *Proceedings of the Fourth IEEE International Conference on Software Engineering and Formal Methods (SEFM 2006)*, pages 3–12, Washington, DC, USA, 2006. IEEE Computer Society Washington.

[5] M. Bernardo. TwoTowers 5.1 User Manual, 2006.

[6] M. Bernardo, P. Ciancarini, and L. Donatiello. Architecting families of software systems with process algebras. *ACM Trans. Softw. Eng. Methodol.*, 11(4):386–426, Oct. 2002.

[7] B. Berthomieu and J.-P. Bodeveix. Formal Verification of AADL models with Fiacre and Tina. In *Embedded Real-Time Software and Systems (ERTS 2010)*, 2010.

[8] B. Berthomieu, P.-O. Ribet, and F. Vernadat. The tool TINA: Construction of abstract state spaces for Petri nets and time Petri nets. *International Journal of Production Research*, 42(14):2741–2756, 2004.

[9] A. Bertolino, P. Inverardi, and H. Muccini. Software architecture-based analysis and testing: a look into achievements and future challenges. *Computing*, 95(8):633–648, 2013.

[10] M. Bozga, S. Graf, and L. Mounier. IF-2.0: A Validation Environment for Component-Based Real-Time Systems. In *International Conference on Computer Aided Verification*, pages 343–348. Springer, 2002.

[11] M. Bozga, S. Graf, I. Ober, I. Ober, and J. Sifakis. The IF Toolset. In *Formal Methods for the Design of Real-Time Systems*, volume 3185 of *LNCS*, pages 237–267. Springer Berlin Heidelberg, 2004.

[12] S. Burmester, H. Giese, M. Hirsch, and D. Schilling. "incremental design and formal verification with uml/rt in the fujaba real-time tool suite". In *"International Workshop on Specification and Validation of UML Models for Real Time and Embedded Systems, SVERTS2004, Satellite Event of the 7th International Conference on the Unified Modeling Language"*, 2004.

[13] M. Y. Chkouri and M. Bozga. Prototyping of distributed embedded systems using AADL. *ACESMB 2009*, pages 65–79, 2009.

[14] S. Chouali and A. Hammad. Formal verification of components assembly based on SysML and interface automata. *Innovations in Systems and Software Engineering*, 7(4):265–274, Oct. 2011.

[15] A.-L. Courbis and T. Lambolais. IDCM. http://idcm.wp.mines-telecom.fr. Accessed: 2017-04-01.

[16] A.-L. Courbis, T. Lambolais, H.-V. Luong, T.-L. Phan, C. Urtado, and S. Vauttier. A formal support for incremental behavior specification in agile development. In *The 24th International Conference on Software Engineering and Knowledge Engineering (SEKE)*, pages 694–699, 2012.

[17] A.-L. Courbis, T. Lambolais, and T.-H. Nguyen. Safe Incremental Design of UML Architectures. In *29th International Conference on Software Engineering and Knowledge Engineering*, 2017.

[18] A. Cuccuru. Meaningful composite structures. In K. Czarnecki, I. Ober, J.-M. Bruel, A. Uhl, and M. Völter, editors, *Model Driven Engineering Languages and Systems (MODELS 2008)*, volume 5301 of *LNCS*, pages 828–842. Springer Berlin Heidelberg, 2008.

[19] Z. Daw, J. Mangino, and R. Cleaveland. Uml-vt: A formal verification environment for uml activity diagrams. In *P&D@ MoDELS*, pages 48–51, 2015.

[20] S. Djaaboub, E. Kerkouche, and A. Chaoui. Generating verifiable LOTOS specifications from UML models: A graph transformation-based approach. *International Journal of Embedded Systems*, 10(6):453–469, 2018.

[21] H. Garavel, F. Lang, R. Mateescu, and W. Serwe. CADP 2010: A Toolbox for the Construction and Analysis of Distributed Processes. In P. A. Abdulla and K. R. M. Leino, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 6605 of *LNCS*, pages 372–387. Springer Berlin Heidelberg, Saarbrücken, 2011.

[22] A. Garis, A. C. Paiva, A. Cunha, and D. Riesco. Specifying UML protocol state machines in alloy. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7321 LNCS(June):312–326, 2012.

[23] M. Graiet, M. T. Bhiri, F. Dammak, and J.-P. Giraudin. Adaptation d'UML2.0 à l'ADL Wright. In *CAL*, pages 83–100, 2006.

[24] B. Hnatkowska and Z. Huzar. Transformation of Dynamic Aspects of Uml Models Into Lotos Behaviour Expressions. *International Journal of Applied Mathematics and Computer Science*, 11(2):537–556, 2001.

[25] M. Kherbouche and B. Molnár. Formal model checking and transformations of models represented in uml with alloy. In *International Workshop on Modelling to Program*, pages 127–136. Springer, 2020.

[26] T. Lambolais and A.-L. Courbis. Development and Verification of UML Architecture by Refinement and Extension Techniques. In *European Congress on Embedded Real Time Software and Systems (ERTS)*, 2018.

[27] T. Lambolais, A.-L. Courbis, H.-V. Luong, and C. Percebois. IDF: A framework for the incremental development and conformance verification of UML active primitive components. *Journal of Systems and Software*, 113:275–295, 2016.

[28] G. Leduc. Conformance relation, associated equivalence, and minimum canonical tester in LOTOS. *PSTV XI. North-Holland*, pages 249–264, 1991.

[29] A. L. Muniz, A. M. Andrade, and G. Lima. Integrating uml and uppaal for designing, specifying and verifying component-based real-time systems. *Innovations in Systems and Software Engineering*, 6(1):29–37, 2010.

[30] I. Ober and I. Dragomir. Unambiguous UML composite structures: the OMEGA2 experience. *SOFSEM 2011: Theory and Practice of Computer Science*, pages 418–430, 2011.

[31] F. Oquendo. π-Method: A Model-Driven Formal Method for Architecture-Centric Software Engineering. *ACM SIGSOFT Software Engineering Notes*, 31(3):1–13, 2006.

[32] F. Oquendo, B. Warboys, R. Morrison, R. Dindeleux, F. Gallo, H. Garavel, and C. Occhipinti. ArchWare: Architecting Evolvable Software. In *Software Architectures*, volume 3047 of *LNCS*, pages 257–271. Springer Berlin Heidelberg, 2004.

[33] T.-L. Phan. *Développement Incrémental de Spécifications d'Architectures en UML Intégrant des Procédures de Vérification*. PhD thesis, Montpellier 2, France, 2013.

[34] M. Y. Said, M. Butler, and C. Snook. A method of refinement in UML-B. *Software & Systems Modeling*, 14(4):1557–1580, 2015.

[35] F. B. Schneider. Decomposing Properties into Safety and Liveness using Predicate Logic. Technical report, Cornell Univ. Ithaca, NY, Dept. of Computer Science, 1987.

[36] Texas-Instruments. Automotive Adaptive Front-lighting System Reference Design. Technical Report SPRUHP3, Texas Instruments, System Application Engineering, July 2013.

[37] G. Waignier, A.-F. Le Meur, and L. Duchien. FIESTA: A Generic Framework for Integrating New Functionalities into Software Architectures. In F. Oquendo, editor, *Software Architecture*, volume 4758 of *LNCS*, pages 76–91. Springer Berlin Heidelberg, 2007.

[38] J. K. Wortmann, S. R. Olesen, and S. Enevoldsen. Caal 2.0 recursive hml, distinguishing formulae, equivalence collapses and parallel fixed-point computations, 2015.

Session We.4.A

AI:Assurance & Testing I

Wednesday 1st June

17:00

–

Amphithéâtre

Programming Neural Networks Inference in a Safety-Critical Simulation-based Framework

Bernard Dion[1], Max Najork[1], Nicolas Dalmasso[1], Jean-Louis Colaço[1], Olivier Andrieu[1],
Bernd Buettner[1], Thomas Most[1], Janaína Ribas de Amaral[2]

[1]Ansys [2]Airbus

Abstract

This paper presents a framework for the development and validation of a neural network in the context of a safety-critical function implemented in SCADE. It consists in a simulation-based training process to build a model in a deep reinforcement learning environment, to transform it into a SCADE model, and to perform again runs with the same simulation environment for validation of the safety-critical function. The approach is illustrated by a use case regarding an unmanned aerial system that calculates a maneuver to avoid collisions with other aircraft, infrastructure, and ground. The training and validation phase, including sensitivity analysis and reliability calculation have been performed successfully, as well as the implementation of the function in SCADE.

1 Introduction

The challenges raised by the introduction of autonomy in embedded systems and the success of Neural Network (NN) based approaches in many domains has generated strong interest from the safety-critical systems industry. Cross-industry working groups have started to systematically collect the challenges for machine learning in safety-critical applications and introduce first results for safe usage [1]. Other industry specific work focuses on the safe automation of vehicle functions while combining machine learning techniques and suitable verification techniques [2]. Key standardization bodies, such as SAE and EUROCAE, with the "Artificial Intelligence in Aeronautical Systems: Statement of Concerns" [3], have clearly identified that it is now necessary to incorporate existing work to produce a standard that provides the necessary accommodation to support the integration of Machine Learning (ML) enabled sub-systems into safety-critical aeronautics software, hardware, and system development.

In this context, Ansys and some of its customers, anticipating that safety-critical software will evolve to integrate machine learning functions, are working in this direction. The goal of this paper is to define an end-to-end workflow for the safety-critical development of a neural network-based vehicle function and to demonstrate its applicability to an Airbus use case.

The approach that we propose is based on three pillars:

- The formal notation and qualified code generation of the Scade language that provide a solid foundation for implementing neural networks
- The definition of the Operational Design Domain (ODD) and a scenario-driven evaluation of the ODD
- The combination of simulation-based Reinforcement Learning (RL) [4], and robustness and reliability analyses [11][12] to create and assess a Neural Network model together with its safe implementation in SCADE, as described in Figure 1 below.

The Operational Design Domain (ODD) is first created to describe the system's intended operating conditions. System safety analysis activities are then performed, and the system architecture is defined. When done with this initial phase, the neural network architecture and its parameters are defined, and training can be performed based on simulation scenarios (see Section 2). Complementing the training activities, the neural network behavior can be assessed through robustness and sensitivity analysis, and training scenarios where insufficient generalization was observed can be added to the training process until functional and safety goals are achieved.

Once training is completed, the Neural Network is automatically imported into SCADE to create a complete embedded Software (SW) application, including both the Neural Network and the traditional functions in a unique design model (see Section 3). The same simulation environment is finally used to validate the complete application.

Reliability analysis can be performed to assess the probability of failure (see Section 4) and classical verification activities be carried out for the complete application.

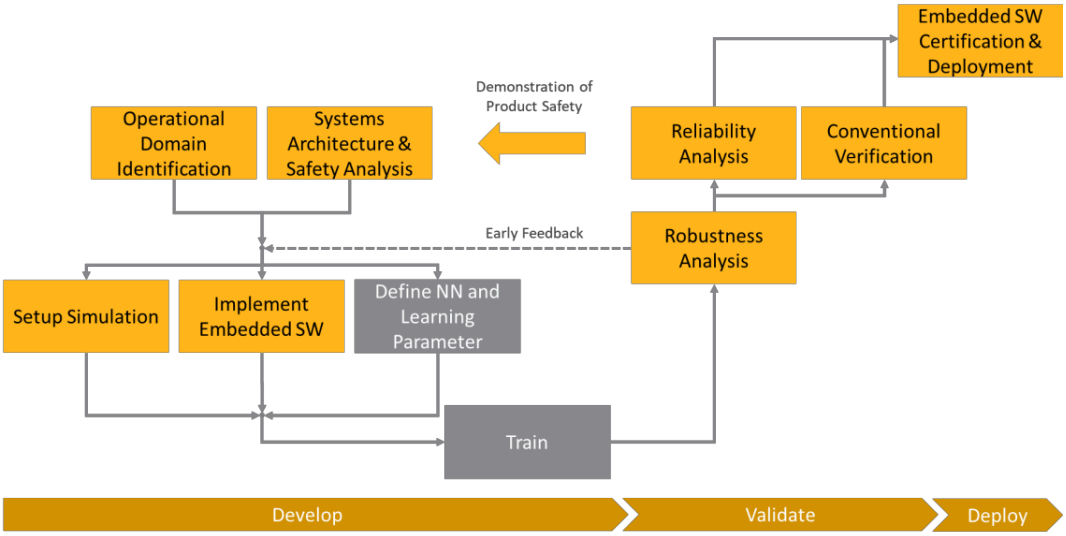


Figure 1: the overall workflow for training and implementing a neural network.

To illustrate this workflow, Airbus proposed the use case of an aircraft that must reach an item of interest, while avoiding an intruder. In Section 5, the use case and its implementation will be described in depth. The resulting artifacts from the training process, the import to SCADE, and the different forms of analysis will be presented.

2 Training the Neural Network Model

In the training phase (see Figure 1), we show variations of a flight scenario (an episode) to the reinforcement learning agent [4]. Over the execution of many episodes, the agent will learn based on trial and error. In the early training stages, it will pick actions randomly and observe the reward. It will then start to favor actions that result in a higher expected reward, becoming more and more accurate in the action selection.

The training environment for Deep Reinforcement Learning (DRL) is shown in Figure 2 and it requires:

- **The agent:** to perceive and interact with the environment for its trial-and-error learning approach. It includes the simulated physical aircraft (we used the Ansys AVxcelerate Simulator for that purpose [5]), and it requires a significant amount of traditional Software (SCADE) performing pre- and post-processing to the NN, as well as the implementation of subsequent conventional control algorithms.
- **The environment:** this includes the scenario configuration (start point, waypoints, buildings, etc.) as well as the underlying physics of the world, such as flight physics to simulate and propagate the objects (again using the AVxcelerate Simulator for that purpose).
- **The reward function:** to evaluate the agent's performance and to provide incentives for good behavior (SCADE). The total reward of an episode is the accumulation of the individual rewards over the time steps of the episode.
- **The Neural Network and DRL framework:** to hold the Neural Network. It collects the experiences (observation, action, and rewards) from the last episodes and evaluates a loss function to optimize for actions that yield the best expected rewards. The DRL framework then starts tuning every individual weight of the millions of neural network weights so that the Neural Network produces better actions. For the experiment in this paper, we have used minds.ai DeepSim [6] as the DRL framework.
- **Results analytics:** to assess the neural network behavior and visualize it so the engineer can understand it. This is performed with the Ansys optiSLang tool to uncover issues while performing and sensitivity (Figure 11) and robustness (Figure 14) analysis, and to understand the need for further training episodes.

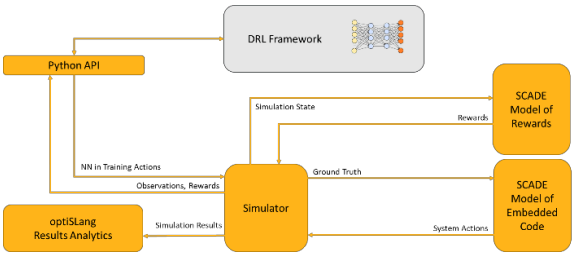


Figure 2: the training environment for DRL.

3 Implementing the Inference Model with SCADE

SCADE Suite is a development environment dedicated to the software design of safety critical systems. The toolset supports qualified workflows: code generation, test execution, and coverage analyses; all qualified for safety standards (DO-178C, EN 50128, and ISO 26262). The qualification credit of the code generator allows to compile C code from a SCADE design, without the need to verify that this code complies with the design. This approach has been used successfully and contributed to the certification of more than 300 systems in the last 20 years.

This environment is built on an underlying programming language called Scade [7], which is a major evolution of Lustre [8]. The Scade language allows the manipulation of composite data organized in structures or arrays. It is strongly typed, and the types include array sizes to ensure that all the accesses are within bounds. This property is statically verified in the front-end of the code generator, and no code is produced for incorrect models. To introduce and illustrate Scade arrays, we start with simple linear forms. Then, we address standard NN layers.

Programming Linear Forms in Scade

Scade provides a type construct to represent unidimensional arrays of a given type and size; this size is part of the type and verified by the type checker. For instance, a vector of 32 bits integers has type `int32^m` where `m` is a static integer. Multidimensional arrays are represented as vectors of vectors. The primitives proposed in Scade to define and use arrays ensure that *all the items have a defined value*, and *all the accesses are done within bounds.* These properties are strong, and they are not guaranteed by general purpose programming languages without additional activities (code review, tests, or proofs). While in a correct Scade model, they are guaranteed. To achieve this goal on array computation, Scade arrays are manipulated through dedicated primitives. Here are the main ones:

- `e^n` represents an array of size `n` in which all elements are equal to `e`
- `(A.[i] default e)` is a projection of `A` at index `i`. It returns `e` whenever `i` is out of `A` bounds
- `A[i]` is a projection at static index `i`
- `(map Op <<n>>)(A, B)` applies operator `Op` point wisely to the elements of arrays `A` and `B`
- `(fold Op <<n>>)(a, A, B)` cascades the application of `Op` with the elements of `A` and `B` starting with `a`

Figure 3 gives the complete definition of the scalar product operator, with its genericity parameters in size (`m`, `n`) and type (`'T`), with the constraint that it must be numeric. Mathematically, the product of a matrix `A` by a vector `u` can be defined as the vector `w` which is such that its i^{th} coefficient is defined by the scalar product of the i^{th} line of `A` by vector `u`. Figure 4 shows the corresponding definition; we can notice the presence of a transposition to iterate on the lines of `A`. These simple and very regular formulas can be written in a compact way without explicit access to the vectors' elements (*i.e.*, without projection).

```
function ScalProd <<n>> (u, v: 'T^n)
returns (w:'T) where 'T numeric
  w = (fold $+$ <<n>>)
                (0, (map $*$ <<n>>)(u, v));
```

Figure 3: scalar product in Scade.

```
function MatVectProd <<m, n>> (A: 'T^m^n; u: 'T^n)
returns (w: 'T^m) where 'T numeric
  w = (map (ScalProd <<n>>) <<m>>)
                      (transpose(A; 1; 2), u^m);
```

Figure 4: matrix-vector product in Scade.

Multilayer Perceptron in Scade

A classical design of neural network (see [9]) is the multilayer perceptron (MLP). Each layer computes output values from its input values by the composition of a linear combination of the weighted input values with a non-linear activation function. The output of the last layer of the MLP constitutes the output of the whole network. The implementation of a layer corresponds to a product of the layer input vector with the matrix of coefficients plus a bias; both coefficients and bias are tuned during the learning phase and do not vary during the inference phase. The definition of the operator `dot_bias` (*resp.* `InnerProduct_1D`) is basically the same as `ScalProd` (*resp.* `MatVectProd`) introduced earlier with the integration of the bias. Figure 5 shows how to define standard layers that can be grouped in a library of Layers that is then used to develop different applications involving MLP. Figure 6 defines the classical ReLU activation function that is straightforward to implement in Scade; other classical activation like tanh, sigmoid, softmax etc. may require floating-point math primitives to be defined.

```
function dot_bias <<n>>(x, w: 'T^n; b: 'T)
returns (y: 'T) where 'T numeric
  y = (fold $+$ <<n>>)(b, (map $*$ <<n>>)(x, w));

function InnerProduct_1D <<n, m>>
  (x: 'T^n; weight: 'T^n^m; bias: 'T^m)
returns (y: 'T^m) where 'T numeric
  y = (map (dot_bias <<n>>) <<m>>) (x^m, weight, bias);
```

Figure 5: a dense layer in Scade.

```
function _relu (x: 'T)
returns (y: 'T) where 'T numeric
  y = if x >= 0 then x else 0;

function ReLU <<n>> (x: 'T^n)
returns (y: 'T^n) where 'T numeric
  y = (map _relu <<n>>)(x);
```

Figure 6: a ReLU layer in Scade.

Convolutional Neural Networks in Scade

For some applications, the important information is more local, and this locality must be captured by the network. This led to a new kind of layer called convolution [10] that only combines neurons that are neighbors. A convolution is a mathematical operation that combines these neighbors based on a kernel of weights. Its definition involves a sliding window on the array content that is done with an iterator that captures the iteration index (mapi) and array projections to access the neighbors of a cell. Figure 7 shows a Scade definition of a pooling layer [10]. The entry point for this function is Pool_max. It takes a 3-D array as an input and, for each 2-D array of size D1*D2, returns an array that is half size in each dimension and whose content is defined as the maximum content of a kernel of size 2*2.

```
function private _Pool_max_cell <<D2, D1>>(j,i : int32; x : 'T ^D1^D2)
returns(y : 'T) where 'T numeric
  y = max( max((x.[i*2  ][j*2  ] default 0), (x.[i*2  ][j*2+1] default 0)),
           max((x.[i*2+1][j*2  ] default 0), (x.[i*2+1][j*2+1] default 0)));

function private _Pool_max_row <<D2, D1>>(i: int32; x: 'T ^D1^D2)
returns (y: 'T^(D1/2)) where 'T numeric
  y = (mapi (_Pool_max_cell <<D2, D1>>) <<D1/2>>)((i, x)^(D1/2));

function private _Pool_max_mat <<D2, D1>>(x: 'T ^D1^D2) returns(y: 'T^(D1/2)^(D2/2)) where 'T numeric
  y = (mapi (_Pool_max_row <<D2, D1>>) <<D2/2>>)(x^(D2/2));

function Pool_max <<D3, D2, D1>>(x: 'T^D1^D2^D3) returns (y: 'T^(D1/2)^(D2/2)^D3) where 'T numeric
  y = (map (_Pool_max_mat <<D2, D1>>) <<D3>>)(x);
```

Figure 7: a pooling layer in Scade.

4 Function Implementation and Validation

Figure 8 below illustrates the import of a trained NN into Scade from a machine learning framework. The trained NN model, as stored in the DRL framework, is transformed into a SCADE NN inference model based on the above library of layers, while discarding all unnecessary information used for training purposes. This transformation is automated through tool support within SCADE.

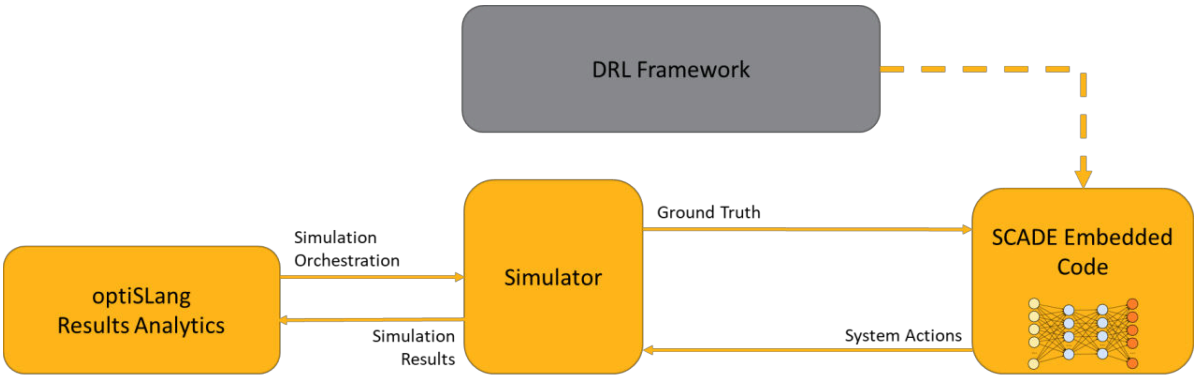


Figure 8: importing the trained inference model in SCADE and verifying the design.

Once the NN has been imported into SCADE, as shown in Figure 8 above, scenario-based co-simulation of the complete SCADE model together with the Simulator is performed. Based on statistical testing, the probability of failure of the application can be estimated, leveraging a probabilistic model for the occurrence of each scenario and the probability distribution of the input parameters of these scenarios. Rather than typical Monte-Carlo simulation, optiSLang is using Importance and Directional Sampling techniques to efficiently calculate the probability of failure of a given scenario, as described in [11], and as shown in Figure 15. This analysis will first be done using model-in-the-loop simulation. In addition, hardware-in-the-loop simulation will be needed to assess the uncertainty due to small numerical differences between the host and target platforms.

5 Airbus Use Case

Unmanned Aerial Vehicles (UAVs), which are aircraft that are guided remotely or autonomously without any human on board, are widely used on tasks like wildfire monitoring [14], surveillance [15], and search and rescue [16]. To be able to operate them autonomously in unknown environments and overcome uncertainties, they need a system that immediately calculates a maneuver to avoid collision with other aircraft, infrastructure, and ground. Existing systems like TCAS are limited because they only consider aerial collision avoidance, and not collision with ground obstacles.

Based on the holistic set of information available, a combined approach, such as Reinforcement Learning which fulfils the possible contradictory objectives of a mission “Approach of an item of interest” and “Avoid conflicts” (obstacles or other vehicles), is an interesting solution to be used for the next generation of maneuver planning systems, as it can deliver a real time response to complex scenarios. However, current certification standards do not consider the application of artificial intelligence in safety-critical systems yet. Therefore, Airbus and Ansys are collaborating in activities that perform investigations on this topic. One of these activities is the evaluation of the safety-critical simulation-based framework proposed by Ansys in this paper.

For that, Airbus proposed the use case “Approach of item of interest with collision avoidance”, illustrated in Figure 9. The goal is to train a Neural Network using Reinforcement Learning to ensure safety of the aircraft and its surrounding by performing collision avoidance maneuvers. Once the Neural Network is trained, the approach described in this paper is applied for the validation and safe implementation of the vehicle function.

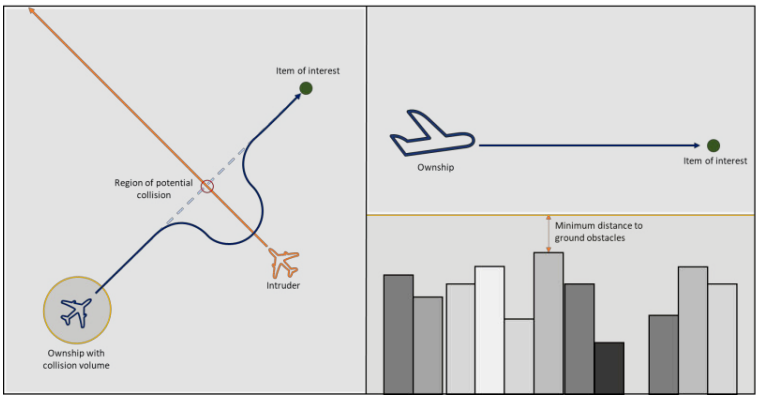


Figure 9: the aircraft must reach an item of interest while avoiding collisions with ground obstacles and an intruder.

The following sub-sections describe the learning problem and the training results, the implementation in SCADE, and the subsequent validation of the vehicle function using the robustness and reliability analysis methods.

Reinforcement Learning Problem Definition

The agent corresponds to a medium size fixed-wing UAV. It must learn how to reach the item of interest, avoid an intruder, and fly with an altitude higher than 30 meters. In DRL, the inputs to the neural networks are called observations. They contain three main types of information:

1. **Ego state:** the orientation of the aircraft (roll, pitch, yaw) and the altitude above ground
2. **Target information:** the Euclidean distance to the target and the 3D directional components to the target
3. **Intruder information:** the Euclidean distance to the intruder, the 3D directional components to the intruder, and the 3D velocity components of the intruder relative to the ego vehicle

All position vectors are measured in the vehicles body-fixed coordinate frame. All observations are pre-processed by the conventional Software, by clipping and normalization operations, so that all neural network inputs are within the interval [-1, +1]. The actions computed by the Neural Network are the pitch, roll, and yaw rate targets, as well as the desired level of thrust. The rate commands are further processed by the flight control functions that command the flight control surfaces (ailerons, elevator, rudder) and the aircraft engine.

Ansys AVxcelerate was used to simulate the physics of the UAV, the intruder, and ground. The agent interacts with this simulation environment to generate the trial-and-error experiences which are divided into episodes. At the beginning of each episode, the positions of the UAV and the item of interest are randomly initialized within a defined airspace volume (bounding box), and the intruder has its straight-line trajectory of fixed length also randomly placed in the environment. The episode ends when the UAV reaches the item of interest or violates the safe distance of 50 meters to the ground or the intruder.

As Deep Reinforcement Learning framework, minds.ai DeepSim [6] was used. It collects the observations, actions, and rewards generated during the episodes and trains the Neural Network. The Neural Network has 2 hidden layers with 256 neurons each. The parameters of the Neural Network were updated using an actor-critic policy gradient method for DRL called Proximal Policy Optimization (PPO) [17].

The reward function is composed of several positive elements. A main reward is provided for reaching the target and being efficient in time doing so. Additional guidance rewards are provided whenever the aircraft gets closer to the target or further away from the intruder.

Training Results

The training was performed over 1 million training steps. The resulting TensorBoard [18] average reward graph is shown in Figure 10.

In the beginning of the training process, one can observe a steep early climb where the basic principles of the scenarios were learned by the Neural Network. After 200,000 steps, we can see an asymptotic phase where small adjustments are made to improve the overall efficiency of the learned policy. The overall graph creates a good impression, indicating that the learning process was successful.

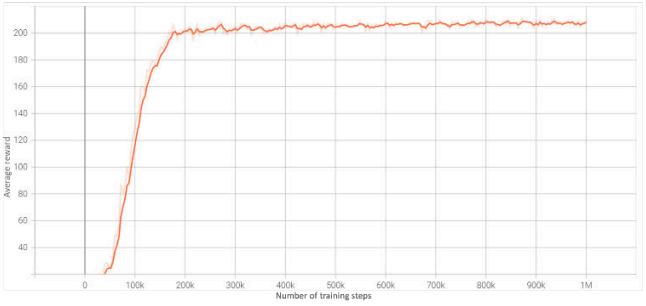


Figure 10: average reward over the training steps.

To confirm the result, a sensitivity analysis using Ansys optiSLang [12] was performed. Figure 11 shows the analysis results. Gray boxes or small percentages indicate the independence between an output and input and vice versa.

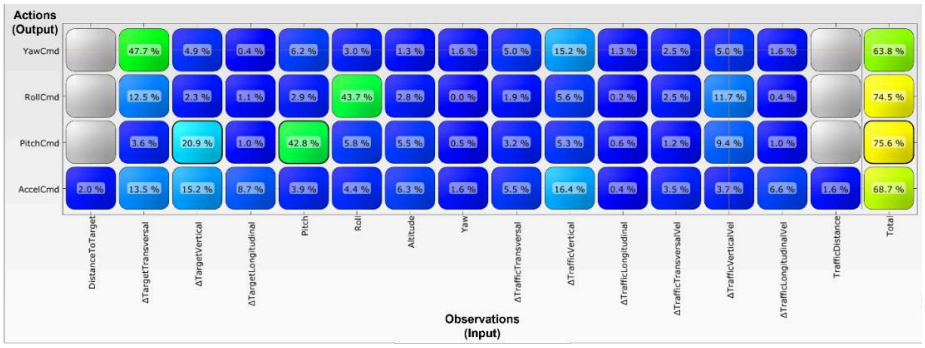


Figure 11: result of the optiSLang sensitivity analysis showing the mapping of neural networks inputs to outputs.

Several observations can be made:

- Firstly, several inputs to the Neural Network contribute very little to the computation of the actions, *e.g.*, the Euclidean distance to the target and the intruder, the measured yaw angle, or the longitudinal distance to the intruder. Those inputs can possibly be pruned.
- Secondly, we see strong dependencies on the measured pitch and roll angles, which is essential to maintain a stable orientation. Additionally, there is a strong effect of the vertical and longitudinal distances on the pitch and acceleration commands, that are consistent with our expectation of fixed-wing dynamics. However, the transversal distance should primarily affect the roll command, to introduce a turn through a combined roll pitch motion. Nevertheless, there is a strong yaw effect shown by the analysis, that may be due to the reinforcement learning algorithm exploiting inaccuracies of the simplified flight dynamic model used. Additional analysis of the representativeness of the flight dynamics model and possible improvements are required.
- Thirdly, we can observe different dependencies on the intruder. However, it is less evident how the precise commands are computed. One explanation is that the intruder distances and velocities are measured in the ego aircraft's body-fixed coordinate frame. For example, at 90-degree roll angle, a transversal motion of the intruder in an earth-fixed coordinate frame would be perceived along the vertical axis within the ego's body-fixed frame. This coupling through the rotations does not allow a trivial control policy but requires a more complex policy considering the orientation of the ego aircraft. From the sensitivity analysis we can observe this clear dependency on the orientation (pitch, roll) as well as the more complex dependencies on the distances and velocities measured in the body-fixed frame.

The open-loop sensitivity analysis confirms the positive impressions from the learning graph. Generally, the learned mappings are consistent with engineering judgement. However, further analysis is required to confirm the correct learning and compliance with safety requirements. In the following sections, we proceed to import the Neural Network to SCADE and perform further analysis based on the closed-loop simulation of the full application.

Import to SCADE

The automatic importer for Keras models [19] translates the Neural Network to a Scade representation, as shown in Figure 12, and the Scade inference model is integrated in the end-to-end vehicle function, as shown in Figure 13.

For the implementation of the complete vehicle function, we use the same functions for normalizing the observations and calculating the controls as we used during the training, while placing the imported Neural Network in between. This ensures full consistency between the training and the inference phase. In the next section, we describe the validation of the end-to-end vehicle function.

```
function #pragma kcg separate_io #end model(observations: float32^15) returns(fc_out: float32^8)
var fc_1, fc_2: float32^256;
let
  fc_out = (Layers::Dense <<256,8>>)(fc_2, fc_out_kernel, fc_out_bias);
  fc_2 = (Layers::TanH <<256>>)((Layers::Dense <<256,256>>)(fc_1, fc_2_kernel, fc_2_bias));
  fc_1 = (Layers::TanH <<256>>)((Layers::Dense <<15,256>>)(observations, fc_1_kernel, fc_1_bias));
tel

const
  imported fc_out_kernel : float32^8^256;
  imported fc_out_bias   : float32^8;
  imported fc_2_kernel   : float32^256^256;
  imported fc_2_bias     : float32^256;
  imported fc_1_kernel   : float32^256^15;
  imported fc_1_bias     : float32^256;
```

Figure 12: trained neural network model in Scade for inference.

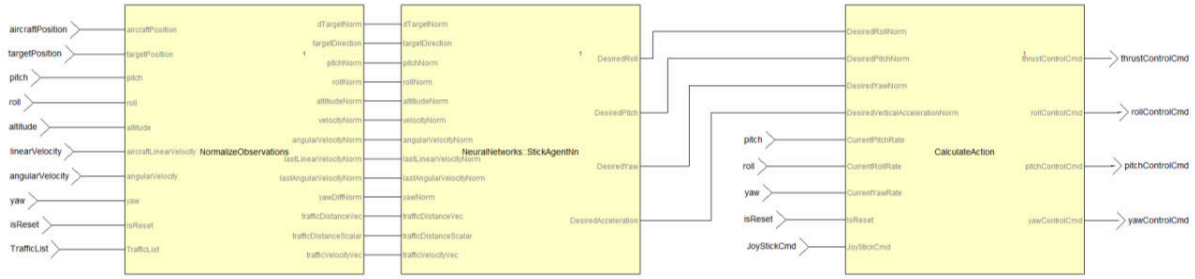


Figure 13: end-to-end vehicle function in SCADE - normalization (left), trained NN (center), and the control functions (right).

Function Validation

The function validation is divided into three steps. Firstly, we need to create a probabilistic model for our scenario parameters to describe the scenario variations we expect in real-world operations. Secondly, we leverage robustness analysis to get a first indication of the performance of the end-to-end vehicle function. Thirdly, we leverage the reliability analysis to quantify the failure probabilities for rare events.

Table 1 below shows the scenario parameters and their corresponding probability distribution. The values are directly entered into optiSLang implementing the probabilistic model. In terms of criteria, both robustness and reliability analysis will evaluate the minimum altitude and minimum collision distance where values below 50 meters are assumed as unsafe.

Table 1: probability distributions for scenario parameters.

Parameter Name	Distribution Type	Distribution values
Target X	Normal	mean: 0, std dev: 50
Target Y (Altitude)	Truncated Normal	mean: 100, std dev: 50, min: 51, max, 1000
Target Z	Normal	mean: -650, std dev: 100
Initial Pitch	Normal	mean: 0, std dev: 0.2
Initial Yaw	Uniform	min: -0.01, max: 0.01
Initial Roll	Normal	mean: 0, std dev: 0.2
Initial Position X	Normal	mean: 0, std dev: 50,
Initial Position Y (Altitude)	Truncated Normal	mean: 100, std dev: 50, min: 51, max, 1000
Initial Position Z	Normal	mean: 650, std dev: 100
Intruder Initial Altitude	Truncated Normal	mean: 100, std dev: 50, min: 51, max, 1000
Intruder Altitude Change	Truncated Normal	mean: 0, std dev: 50, min: -50, max, 1000
Intruder Range	Normal	mean: 50, std dev: 10,
Intruder Heading	Uniform	min: -3.14, max: 3.14

Based on the probabilistic model and the closed-loop simulation containing the end-to-end vehicle function, a robustness analysis is executed. The results are shown in Figure 14.

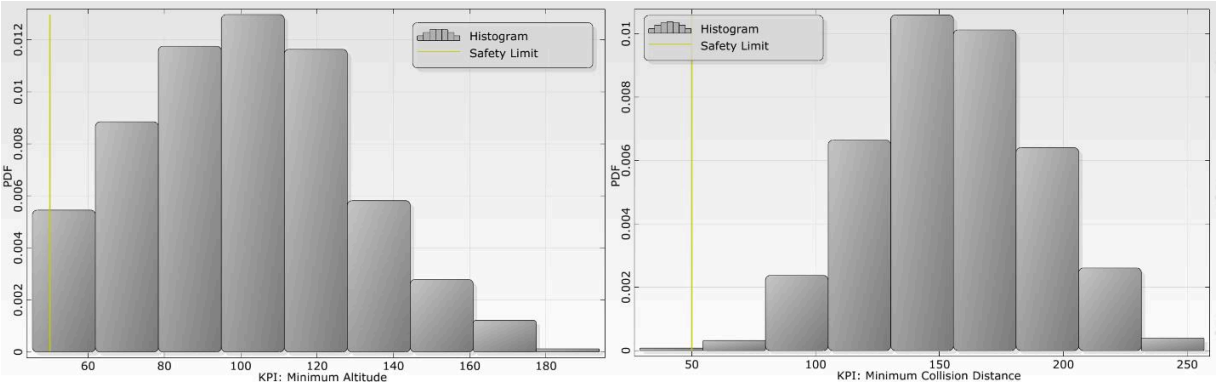


Figure 14: histogram for the Safety KPIs and their corresponding safety limit.

The results show a good compliance with the safety limits. Some individual failures can be observed. Therefore, the next step uses the reliability analysis to quantify the failure rate of the rare events.

The reliability analysis utilizes the same probabilistic model and closed-loop simulation as the robustness analysis. However, using the Adaptive Response Surface Model with Directional Sampling [13], more scenarios are created around the failure regions. The overall result of the reliability analysis for both safety criteria together is shown in Figure 15. The achieved probability is already low, which explains the rare occurrences of safety violations in the robustness analysis.

However, due to the adaptive nature of the reliability analysis, it could find different failure cases and allows us to further understand the scenarios in which failures occur. In this paper, we will summarize the results only for the most dominant scenario parameters and their effect on the safety limit.

Method : Adaptive Response Surface Method (ARSM)

Complete approximations : 10 / 10
Selected data : 10. Approximation

Probability of Failure : 0.000678995
Standard deviation error : 0.000244629
Reliability Index : 3.20343

Approximation errors : R^2 / R^2pred
lim_st_min_altitude: 0.989543 / 0.717368
lim_st_min_collision: 0.999071 / 0.731842

Number of designs
Total : 4100
Safe domain : 3404
Unsafe domain : 696
Failure strings : 0
Failed : 0

Directional Sampling on Response Surface

Complete directions : 5000 / 5000

Number of designs
Total : 26305
Safe domain : 25396
Unsafe domain : 909

Figure 15: results of the reliability analysis.

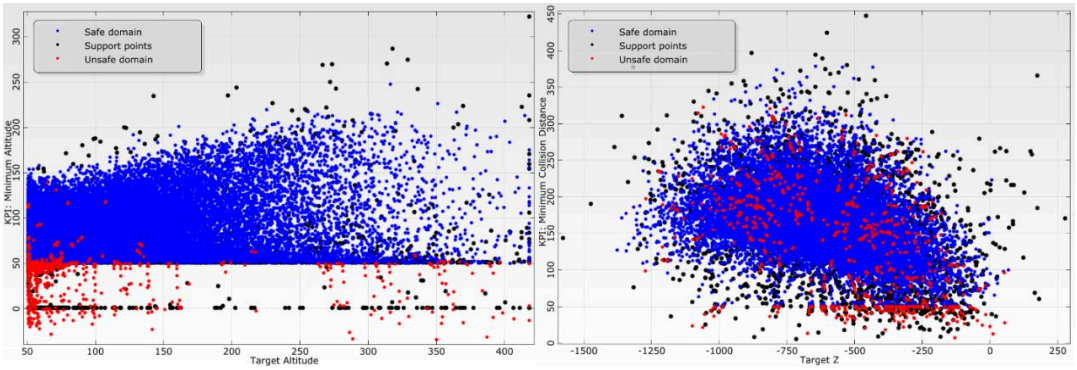


Figure 16: part of the results of the reliability analysis.

From the results graph (Figure 16), we have three types of colored dots. Black dots "Support Points" are the actual scenarios that have been evaluated through the closed-loop simulation. Those scenarios are used as support points to fit the Adaptive Response Surface Model (ARSM). The model is then used for a faster evaluation represented by the blue and red dots depending on whether the result meets the safety criteria or not.

On the left-hand side of Figure 16, we can see the effect the Target Altitude has on the Minimum Altitude of the aircraft. A lot of safety violations occur when the target is close to the ground leaving little room for error. This is in line with our engineering judgement. However, there is an additional failure region for target altitudes above 250 meters. This indicates a lack of generalization capability of the Neural Network for these target ranges. We can also see that the minimum aircraft altitude is the predominant failure cause for our scenarios.

On the right-hand side of Figure 16, we can see the relation between the Minimum Collision Distance and the Target Z coordinate that determines the distance of the target along the longitudinal axis[1] (backward - forward) of the aircraft. It will also naturally influence the distance between the intruder and the target. Here, the focus is on failed designs below the minimum collision distance of 50 meters, to discard the failures due to a ground collision. We can observe both from the support points (black) and failed designs (red) that intruder collisions predominantly occur when the Z coordinate is between -500 and 0. When we compare these values to our bounding box for the Target Z coordinate during training [-800, -500], we can explain this lack of robustness through the fact that the predominant failure range was not included in the training phase and the agent did not sufficiently generalize to this region.

In summary, the reliability analysis shows that the algorithm performs well on most scenarios with a good failure rate. Additionally, it pinpoints specific failure regions that can be improved in further training iterations.

6 Conclusion and Future Work

This work has applied the described methodology proposed by Ansys on the Airbus use case, approaching an item of interest, while avoiding an intruder. While doing so, we achieved the following results:

- The agent was successfully trained.
- The sensitivity analysis helped to get valuable insights into the learned control policy.
- When evaluating the end-to-end vehicle function implemented in SCADE, consisting of the traditional Software as well as the imported Neural Network, the robustness analysis showed overall good results with little failures.
- Leveraging the reliability analysis, we could quantify the failure probability of the vehicle function on the described scenario and pinpoint the different failure regions.
- Future work can build on top of the analysis results, and further improve the vehicle function and the training process.

[1] References are in the frame of the general-purpose dynamics engine used, which may deviate from aeronautical reference conventions.

Additionally, as explained in Section 4, the numerical differences between host and target computations could possibly influence the performance of the vehicle function and the compliance with safety requirements. Future work should elaborate more in detail on the requirement and methodology for target testing to prove the absence of significant differences between computation platforms.

Finally, as major certification credit is taken from the analysis and closed-loop simulation, tool qualification considerations need to be established. This is not only limited to the correct functioning of the tools guaranteeing the absence of implementation errors, but it also relates to the accuracy of the simulation itself and the methods used for robustness and reliability analysis.

References

[1] E. Jenn, A. Albore, F. Mamalet, G. Flandin, C. Gabreau, H. Delseny, A. Gauffriau, H. Bonnin, L. Alecu, J. Pirard , B. Lefevre, J-M. Gabriel, C. Cappi, L. Gardès, S. Picard, G. Dulon, B. Beltran, J-C. Bianic, M. Damour, K. Delmas, and C. Pagetti, "Identifying Challenges to the Certification of Machine Learning for Safety Critical Systems", 10th European Congress Embedded Real Time System (ERTS), March 2020

[2] "Safety First for Automated Driving", APTIV, Audi, Baidu, BMW, Continental, Daimler, FCA, Here, Infineon, Intel, and VW, July 2019, available at: https://www.daimler.com/innovation/case/autonomous/safety-first-for-automated-driving.html

[3] AIR6988, "Artificial Intelligence in Aeronautical Systems: Statement of Concerns", SAE, April 2021

[4] R. Sutton and A. Barto, "Reinforcement Learning: An Introduction", MIT Press, 2018

[5] Ansys AVxcelerate, 2022R1, Ansys, December 2022

[6] DeepSim, https://www.minds.ai/deepsim, minds.ai, 2021

[7] J-L. Colaço, B. Pagano and M. Pouzet, "Scade 6: A formal language for embedded critical software development", 11th International Symposium on Theoretical Aspects of Software Engineering, TASE 2017, September 2017

[8] P. Caspi, N. Halbwachs, D. Pilaud and J. Plaice, "Lustre: a declarative language for programming synchronous systems", 14th ACM Symposium on Principles of Programming Languages, October 1987

[9] I. Goodfellow, Y. Bengio, and A. Courville, "Deep Learning", MIT Press, 2016

[10] Y. LeCun, "LeNet-5, convolutional neural networks," available at: http://yann.lecun.com/exdb/lenet/

[11] Rasch, M., P.T. Ubben, T. Most, V. Bayer, R. Niemeier, "Safety Assessment and Uncertainty Quantification of Automated Driver Assistance Systems using Stochastic Analysis Methods", NAFEMS World Congress, Quebec, Canada, June 2019

[12] T. Most and J. Will, "Sensitivity analysis using the Metamodel of Optimal Prognosis", Proceedings Weimar Optimization and Stochastic Days (WOST), Weimar, 2011

[13] D. Roos and U. Adam, "Adaptive Moving Least Squares approximation for the design reliability analysis", Proceedings Weimar Optimization and Stochastic Days (WOST), Weimar, 2006

[14] M. A. Akhloufi, A. Couturier, and N. A. Castro, "Unmanned Aerial Vehicles for Wildland Fires: Sensing, Perception, Cooperation and Assistance", Drones 2021, 5, 15, available at: https://doi.org/10.3390/drones5010015

[15] A. Ahmadzadeh, J. Keller, G. Pappas, A. Jadbabaie, and V. Kumar, “An Optimization-Based Approach to Time-Critical Cooperative Surveillance and Coverage with UAVs”, Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 491–500, available at: https://doi.org/10.1007/978-3-540-77457-0_46

[16] C. Recchiuto, C. Nattero, A. Sgorbissa, and R. Zaccaria, “Coverage algorithms for search and rescue with UAV drones”, AI*IA Symposium on Artificial Intelligence (AIRO Workshop), Pisa, December 2014

[17] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, “Proximal Policy Optimization Algorithms”, Cornell University, arXiv:1707.06347, July 2017

[18] TensorBoard, https://www.tensorflow.org/tensorboard, TensorFlow

[19] F. Chollet, “Deep Learning with Python”, Second Edition, Manning, October 2021

Leveraging Influence Functions for Dataset Exploration and Cleaning

Agustin Martin Picard*†, David Vigouroux†, Petr Zamolodtchikov‡,
Quentin Vincenot‖†, Jean-Michel Loubes§, Edouard Pauwels¶
* Scalian, † IRT Saint Exupéry, ‡ University of Twente, ‖ Thales Alenia Space,
§ Institut Mathématique de Toulouse, Université Paul Sabatier, ¶ IRIT, CNRS, Université de Toulouse,

***Abstract*—In this paper, we tackle the problem of finding potentially problematic samples and complex regions of the input space for large pools of data without any supervision, with the objective of being relayed to and validated by a domain expert. This information can be critical, as even a low level of noise in the dataset may severely bias the model through spurious correlations between unrelated samples, and under-represented groups of data-points will exacerbate this issue. As such, we present two practical applications of influence functions in neural network models to industrial use-cases: exploration and clean-up of mislabeled examples in datasets. This robust statistics tool allows us to approximately know how different an estimator might be if we slightly changed the training dataset. In particular, we apply this technique to an ACAS Xu neural network surrogate model use-case[14] for complex region exploration, and to the CIFAR-10 canonical RGB image classification problem[20] for mislabeled sample detection with promising results.**

Keywords: Intelligent Systems, Artificial Neural Networks, Influence functions, Dataset Exploration, Mislabeled Sample Detection.

I. Introduction

In recent times, the field of artificial intelligence (AI) and machine learning has been shifting to a paradigm with an extreme reliance on huge, over-parametrized models and equally large pools of data. These models leverage correlations between samples of the same class to perform their predictions with an impressive precision on domains ranging from image classification [20, 8], object detection [8, 22] and semantic segmentation [22, 6] in computer vision, to sentiment analysis [23] and natural language translation [16] in natural language processing (NLP). The exponential growth of widely available, vast amounts of data for these applications has been a major catalyst in this process, and along with it comes the need to validate it and clean it.

Traditionally, domain experts would be requested to perform this task, but when dealing with millions of examples, it becomes impossible to realistically verify each one. As such, there has been an emergence of different techniques to remedy this issue by automatically proposing interesting samples that might need to be verified before integration into the dataset.

Related work

Namely, Koh et al. recovered a classical statistics tool and adapted it to differentiable statistical models and applied it to dataset cleaning and white-box model explainability [18] with interesting results in the former task. It is this formulation that we will be employing in this work. They later went further and studied the effect groups of data-points have on models in [17], with Basu et al. providing a second-order formulation that takes into account pairwise interactions between samples [3]. Choosing a different starting point, Giordano et al. have proposed alternative formulations based on the infinitesimal jack-knife for first [9] and higher order approximations [10], where they also provide bounds on the error incurred during the approximation.

More recently, Kong et al. have adapted the notion of influence function to Variational AutoEncoders in [19], where they address the problem of computing the loss of test samples over the expectation over the encoder, and they apply it to dataset cleaning tasks on the MNIST [21] and CIFAR-10 [20] computer vision datasets. With another application in mind, Alaa et al. have employed higher-order influence functions to measure uncertainty in deep learning models [1], showcasing the numerous applications of this tool.

There are also numerous techniques for estimating the influence of training samples based on other principles. In [32], the authors propose to leverage the representer theorem [26] to find the training points that contribute the most – both positive and negatively – to the model's output. In particular, they apply this technique to offer explanations to the neural network's decisions. With the same application in mind and basing themselves off of the same principle, Sui et al. derive a technique that employs a local Jacobian Taylor expansion instead of re-fitting a new output layer [30] – as in [32]. They also measure its capacity to detect mislabeled examples in very noisy scenarios. Finally, in [25], Pruthi et al. propose to compute a first-order approximation of the influence function by capitalizing on the model's checkpoints saved during the training process. This allows them to spare themselves the difficulties related to handling with hessian matrices of considerable size – a problem that, depending on the dimensionality of the task, can complicate matters in our case.

However, they all performed tests on canonical datasets and focused on single model architectures. In contrast, in this work, we successfully demonstrate the utility of influence functions on practical industrial applications, namely on the ACAS Xu use-case, and we perform experiments on multiple types of models and training schedules to conclude that they constitute a critical component to

obtaining satisfactory results.

In practice, we know that mislabeled data-points heavily influence the shape of the decision boundary, and thus, when fixed, force the optimal model to change drastically to accommodate this deformation. In much the same way, removing samples from insufficiently dense regions of the input space will result in a model whose parameters are significantly different. However, it would be prohibitively time-consuming to re-train a model leaving out one sample at a time in most use-cases. This is why we turned to the concept of the influence function, that allows us to approximately measure the importance of each data-point without the need for expensive re-trainings – assuming certain criteria are met.

II. Influence Functions

A. Theory

In the literature of classical robust estimation, the question of how to appropriately measure the influence of subsamples of data on a given estimator is often raised [11, 5, 13]. In other words, how an estimator would behave if some of the data-points were not provided. An accurate way to answer this question would be to re-fit the estimator on a dataset with these points held-out. However, for modern models on high-dimensional data, this process can be computationally very expensive and time-consuming. Fortunately, when working with neural network models, it is possible to leverage any of the modern, widely available *automatic differentiation* frameworks to approximate the answer.

In [18, 17], the authors adapt the concept of influence function to neural networks trained using empirical risk minimization (ERM) and leveraging the automatic calculation of gradients.

Definition 1. *Let $z_1, ..., z_n$ be a group of training points, where $z_i = (x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$ – where $\mathcal{X}$ and $\mathcal{Y}$ are the input and output spaces respectively –, $\theta \in \Theta$ be a set of parameters, $\ell(z, \theta)$ a loss function, and $\hat{\theta} = \arg\min_{\theta \in \Theta} \frac{1}{n} \sum_{i=1}^{n} \ell(z_i, \theta)$ the empirical risk minimizer. Then, they posit that the influence that a test point z_{test} has in the model's weights induced by infinitesimally up-weighting a data-point z is equal to [18]:*

$$\mathcal{I}_{\text{up,loss}}(z, z_{\text{test}}) = \nabla_\theta \ell\left(z_{\text{test}}, \hat{\theta}\right)^T H_{\hat{\theta}}^{-1} \nabla_\theta \ell\left(z, \hat{\theta}\right), \quad (1)$$

where $H_{\hat{\theta}} = \frac{1}{n} \sum_{i=1}^{n} \nabla_\theta^2 \ell\left(z_i, \hat{\theta}\right)$ is the average Hessian with respect to the model's weights.

This formulation requires us to make some very strong assumptions about the underlying problem. Namely, we assume:

1) the loss $\ell : (x, y, \theta) \mapsto \ell(f_\theta(x), y)$ to be convex and twice continuously differentiable.
2) that $\exists \theta_0 \in \mathbb{R}^d$ such that $\nabla_\theta \mathbb{E}[\ell(f_{\theta_0}(x), y)] = 0$.
3) the matrix $H_{\hat{\theta}}$ to be invertible.

These hypotheses are reasonable when the model has been trained long enough to be close to a local first and second-order minimum. However, they can constrain the applicability of this formulation considerably, but to work around them, we may limit ourselves to the regression – linear or logistic, depending on the task – between the embedding engendered by the feature extraction chain (from the input to the last layer of the neural network) and the output. If this embedding was created carefully enough, the first and last assumptions should be true, and the second one, approximately verified ($\nabla_\theta \mathbb{E}[\ell(f_{\theta_0}(x), y)] \approx 0$). This also allows us to drastically reduce the computational cost associated to these calculations for the large models that are typically used in high-dimensional applications.

Returning to Eq. 1, when $z = z_{\text{test}}$, this quantity is exactly identical to the influence measure introduced by Cook in [5], also known as Cook's distance, and we will apply this concept to our practical applications.

B. Implementation

Up until now, we have considered a theoretical problem leaving out all possible practical issues we might run into, but it turns out that the exact computation of Cook's distance through the formulation we described above can raise plenty of technical challenges. Namely, calculating the hessian – and thus, its inverse – can be extremely computationally intensive, and even impossible depending on the amount of weights our model has, as this matrix grows quadratically in size with the model's parameters.

The first measure to alleviate the computational burden is to reduce the amount of parameters to consider during the estimation of the $H_{\hat{\theta}}^{-1}$. In particular, we considered only the neural network's last layer containing trainable weights, thus both reducing the size of the hessian and verifying the hypothesis we need for accurate calculation of these quantities.

Secondly, when this does not suffice to render the problem tractable, it is still possible to estimate the *inverse-hessian-vector product* directly without explicitly computing the hessian by using a Conjugate Gradient Descent algorithm [29]. Furthermore, by leveraging *forward-over-backward automatic differentiation*, we can do so in a very memory-efficient manner. However, although this scheme has been successfully implemented, it was not employed for the results we showcase below; **we do estimate the hessian matrix of the loss with respect to the model's weights in our experiments**.

C. Methodology

Intuitively, we can use the concept of influence introduced in 1 to obtain a measure of how "interesting" a data-point in the training dataset is to a given model. If we consider that those that are difficult to learn will be so for any statistical model, we can use the information relayed by these influence functions to assign a value of interest to each data-point and provide them to the user for validation.

It is important to note that we will be focusing on the influence functions with respect to the neural network's last layer. This not only renders the computation considerably more memory-efficient, but also places us closer to the hypotheses that are necessary for accurate computation of these values.

In all of our experiments and for both of the different applications – detection of important data-points and mislabeled example detection –, the procedure we followed was to:

1) Train a neural network model until convergence on 80%/20% training/test dataset splits as it is conventionally done.
2) Compute the value of Cook's distance for each training point separately.
3) Sort the training dataset by its data-points' Cook's distance.
4) Consider the top-most points – i.e. those whose Cook's distance was the highest – as important or mislabeled, and **request human input for validation**.

By performing this procedure, we intend to determine which training points are maximally important to the model, and without whom the final neural network would be the most different. This typically means samples that are under-represented in the dataset – as the model would rely heavily on the little information it has to learn how to predict in those cases –, mislabeled data-points – for much the same reason –, or those that lie right in the decision boundary – as they help the model fix it correctly. In our case, examples coming from all three situations are useful for our tasks.

Concerning the detection of mislabeled samples, as this technique aims to provide us with the training points that would change our model the most if they were removed, we posit that it could help in their detection. These data-points, that, due to a variety of conditions during the consolidation of the dataset, have labels that do not correspond to the correct classes, would introduce confounding factors to our model during the training process. As such, we want to rid ourselves of them, and to do so as efficiently as possible, as inspecting the whole dataset one example at a time might be prohibitively time-consuming for large pools of data. Thus, we postulate that by sorting our data-points by their Cook's distance value, we should be able to accelerate this process and find the mislabeled examples among the most influential points in the dataset.

As a matter of fact, this technique has already been used as a baseline for other methods [32, 25], but with what seems to be different model configurations and training schedules. Indeed, by testing out different architectures, *learning rate* decay functions and layer regularization strategies, we have found that these have a considerable impact on its capacity to single out mislabeled examples.

III. Results

For each of the tasks, we started by testing our intuitions on synthetically generated toy datasets, and then moved on to the actual use-cases. In particular, for the detection of interesting data-points, we focused on simple, two-dimensional binary classification datasets with an increasing level of complexity, and then applied what we had learned to a drone collision avoidance problem. For the mislabeled point detection, we followed the same procedure: we corroborated that we were able to correctly identify a single mislabeled data-point on a simple, two-dimensional binary classification synthetic dataset, and then moved on to noisy versions of the CIFAR-10 image classification dataset.

A. Detection of interesting training points

Experiments on toy examples

As a means to test the validity of this mathematical tool in the context of deep learning models, we began with some examples where we controlled perfectly the inputs and already knew what to expect – i.e. some toy examples. In particular, we generated groups of points in two-dimensional space, with different decision boundaries and local down-sampling strategies to corroborate our intuitions.

For all the following toy problems, a simple 2-hidden layer multi-layer perceptron with sigmoid activations was employed. They were all trained until convergence on splits with 80% of the whole data being used during the training process, and the remaining 20% was held out for validation. Then, we applied our methodology and plotted the whole training dataset with the transparency set as a function of each point's influence.

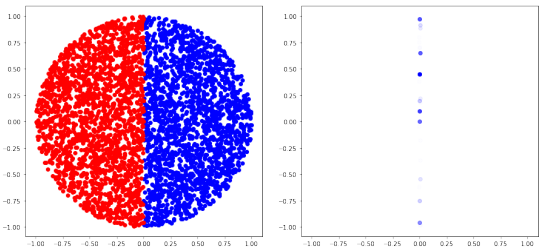
In Fig. 1, we showcase the results, and we observe that for each problem, the most influential points are always close to the decision boundary. This applies to the more complex examples as well, telling us that our intuitions about the capabilities of the power of this tool might be correct. In particular, it is interesting to note how the data-points near under-sampled parts of the boundary of the more complex example carry some influence, as otherwise the model would not know where to place its decision boundary. Similarly, we observe the whole region with mislabeled samples in the noisy boundary toy example to be influential, as the model is not quite confident inside of it and is forced to memorize each data-point, thus rendering them important to the model in question. This last intuition will be important for the mislabeling sample detection task later on.

Experiments on ACAS Xu

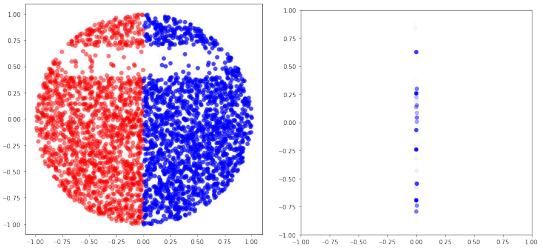
The ACAS Xu problem [14] is an unmanned drone collision avoidance use-case, where, knowing some information about the drone and the intruder's states, and the previous action that was taken, a **look-up table** (**LUT**) is employed to compute the next optimal step our drone has to take to successfully avoid a collision between the two aircraft. Considering that these decisions will depend on 7 variables – 6 describing the current state of both drones + 1 for the previous action –, this cost table can be quite difficult to manipulate efficiently, specially due to the fact that it contains 93 million points.

The six variables that are presented in Fig. 2 are:

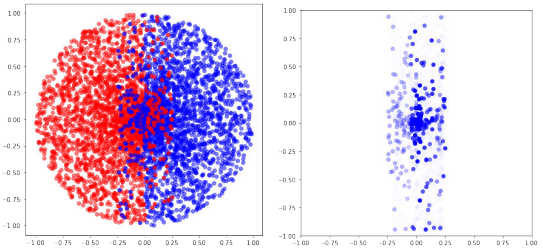
- ρ [ft]: The distance from ownship to intruder.
- θ [rad]: The angle to intruder relative to ownship heading direction.
- ψ [rad]: The heading angle of the intruder relative to ownship heading direction.
- v_{own} [ft/s]: The speed of the ownship.
- v_{int} [t/s]: The speed of the intruder.
- τ [sec]: Time until the loss of vertical separation.



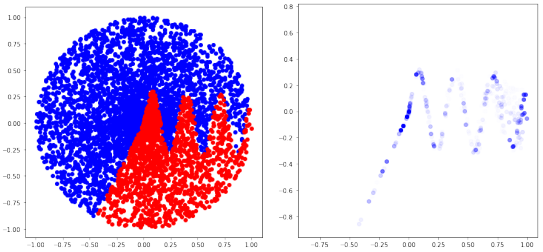
(a) Unit circle dataset (b) Influential points of the unit circle dataset



(c) Unit circle dataset with a down-sampled region (d) Influential points of the unit circle dataset with a down-sampled region



(e) Unit circle dataset with noisy boundary (f) Influential points of the unit circle dataset with noisy boundary



(g) More complex 2D dataset with a line boundary that transforms into a sinus (h) Influential points of the more complex 2D dataset.

Fig. 1: The synthetic 2D toy datasets and their corresponding most influential samples according to simple two-layer perceptrons, in increasing complexity from top to bottom. In the scatter plots on the left, each color indicates the class label for each point. On the right, all the points are colored blue and their transparency is a function of their influence value.

Consequently, a solution to reduce its cost is to train a neural network to operate as a surrogate model, which, in inference, would be able to produce predictions much faster and at a much lower memory cost. This technique would allow us to accurately approximate the predictions of the 2GB+ table with a simple ≈3MB neural network[14, 7].

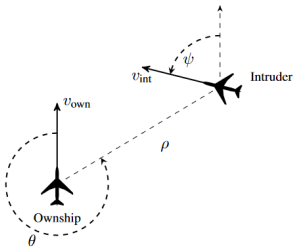


Fig. 2: Illustration of the ACAS Xu problem, as per [14].

Placing ourselves in this frame of work, we wish to increase the confidence we have on what the model is learning without having to manually search for regions of the input space that may be hard to learn for the model. As such, will apply the methodology described above to determine whether we need to gather more data on specific regions with the help of a human domain expert.

To do so, we trained a simple, 6-hidden-layer multi-layer perceptron with ReLU activations with an Adam optimizer on 8096 sized mini-batches with the table's contents for 200 epochs until convergence, reaching 99% accuracy. We chose to optimize directly for the classification between the different 5 actions – 'COC' (clear of conflict), 'WR' (weak right), 'WL' (weak left), 'R' (right) and 'L' (left) – and starting always from the state 'COC', to keep the optimization and interpretation of the results simple, as well as be able to easily apply our current formulation of the influence functions. Additionally, as it is typically done in these sorts of scenarios, we held out 20% of the dataset for validation, which left us with 80% for training.

Thus, we applied the influence function's method to single out the most influential datapoints in the training dataset. For these points, we plotted the two-dimensional cuts obtained by sweeping the range (i.e. the distance to the intruder) and the theta (i.e. the angle between the ownship and the intruder) while keeping the rest of the parameters constant. These plots simulate a situation where the intruder moves in a straight line and our drone tries to avoid it. We expect to find some interesting scenarios, and the most influential points to be close to the decision boundary.

We observe from Fig. 3 that these influential regions usually contain either groups of points that are quite close to each other but belong to different classes, or are sparsely sampled. In both cases, the influence is mostly monopolized by the samples that are close to the decision boundary.

Once these regions have been identified, we can present them to a domain expert for further analysis and validation. If they are indeed important for the task, more data could be gathered to facilitate the learning process near them, and if not, then they can be filtered out to prevent the addition of confounding data-points.

In particular, when we showed our results to some experts on the ACAS Xu drone collision avoidance problem, they

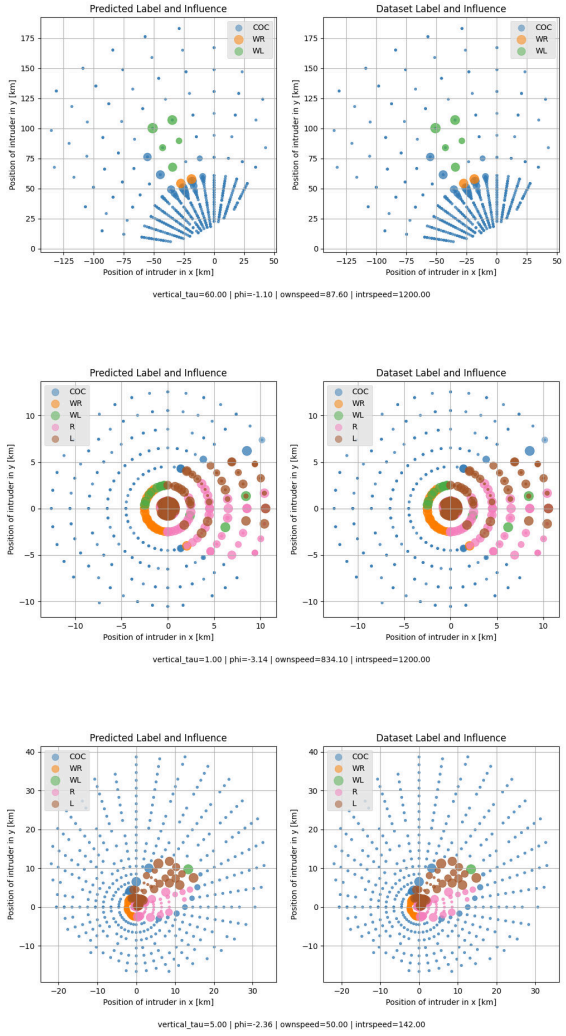


Fig. 3: Some 2D cuts of influential regions in the input space. The scatter plot on the right of each sub-figure represents the ground-truth's decisions, and the one on the left, the model's. Each dot is a point in the training dataset, and its size depends on its influence, bigger meaning more influential.

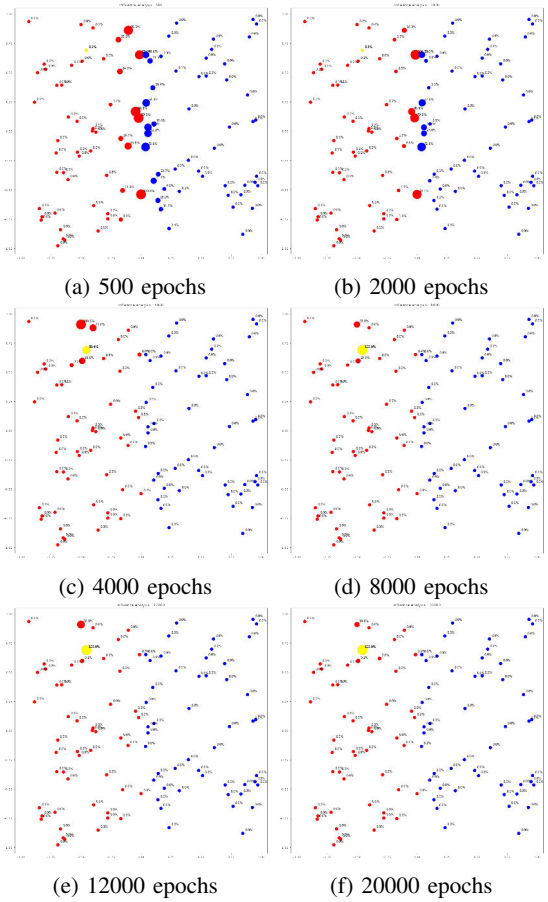
found some interesting patterns: in most of the most influential 2D cuts we generated, there was either one or both drones that were traveling at high speeds. Additionally, they found that there were some situations where the region was very sparsely sampled, and contained some very sudden changes in the **LUT**'s decisions – i.e. switching from left to right and back in contiguous data-points. This is important information to have before finishing the consolidation of the training dataset, as it could guide experts to push for asking for more data from the regions in input space in question.

B. Mislabeled sample detection

Experiment on a toy example

As we have done for the previous task, we test our intuitions on a synthetic dataset so as to be able to easily visualize and understand the problem at hand. In this case, we will generate a set of uniformly distributed points, with a decision boundary at the center of the horizontal axis, of which one point will have its label flipped.

In this experiment, we will attempt to trace the point's influence during the model's training phase to verify whether we can retrieve it successfully. Given the previous ones, we would expect the point to dominate the rest as the model starts to gradually *overfit* on it to minimize the training loss. As the problem is quite a simple one, we solve it with a simple, one-hidden-layer MLP, and we minimize a binary cross-entropy loss.



(a) 500 epochs (b) 2000 epochs

(c) 4000 epochs (d) 8000 epochs

(e) 12000 epochs (f) 20000 epochs

Fig. 4: Influence of a simple 2D binary classification dataset as training progresses. Blue and red indicate each class and the yellow data-point has been incorrectly labeled as blue in the red zone (i.e. has been mislabeled). The size of each point is proportional to its Cook's distance.

In Fig. 4, we observe that the model attributes a high influence to the decision boundary points at first, but as training progresses and it starts to overfit, the influence shifts

to the points close to the mislabeled one – showing that the model is forced to bend its decision boundary to accommodate this outlier –, and ends up with only the mislabeled point dominating the rest of the dataset in terms of influence value.

Experiments on CIFAR-10

Now that our intuitions have been confirmed, we will gauge the informative power of this technique on the CIFAR-10[20] RGB image classification dataset. This problem consists on assigning the correct category to natural images of 32×32 pixels and belonging to 10 classes ranging from cars to frogs and in different contexts.

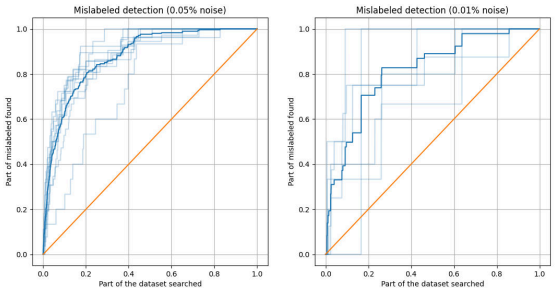
In particular, we have tested the **EfficientNetB0**[31], **ResNet-20**[12] and **VGG-19**[27] architectures, trained with Adam[15] optimizers and custom learning rate schedules, and Dropout[28] and layer-wise elastic net reguralizations (`L1L2`). For each of these configurations, we have trained sets of 6–12 models on the CIFAR-10[20] dataset on two noisy regimes: $\approx 0.05\%$ and $\approx 0.01\%$ of randomly changed labels throughout the training set. These proportions of noise were chosen to simulate what we would expect to have on standard, clean data in industrial use-cases.

In Fig. 5, we observe that, in most cases, our technique of searching the most influential samples for mislabeled images is a good strategy for cleaning up a slightly noisy dataset. Furthermore, it seems that the choice of architecture can have a considerable impact on the results.

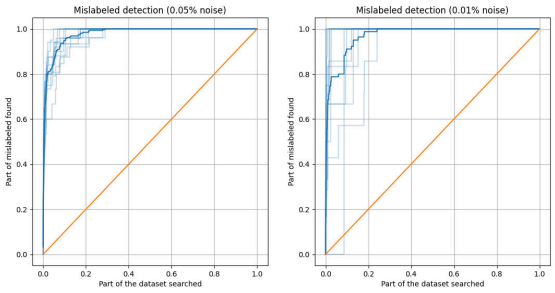
In particular, as we are comparing ROCs, we expect the best curves to be those that get as close to as possible to detecting every mislabeled image with as little samples as possible. Thus, we notice that the **EfficientNetB0** architecture does not seem suitable for this task as the **VGG-19**, which itself performs quite well if we exclude the outliers. Ideally, we would not have any poorly performing models from a given architecture, and this seems to be the case for the **ResNet-20** models. This is why we used this last architecture for the rest of the experiments.

Additionally, we tested the effect of adding elastic net regularization to the classification head of the ResNet-20, and of training with a smaller *learning rate* for more epochs. These results are presented in Fig. 6.

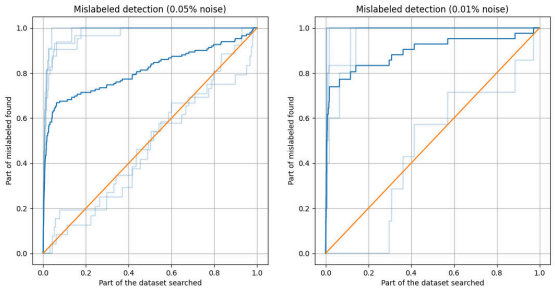
In [2], it was demonstrated that it is possible to more accurately compute influence values on layers that have been trained with layer-wise regularization, and in Fig. 6a, we corroborate this and we obtain an even better performance. However, contrary to our intuition, by training for a longer period of time – and potentially overfitting the model on the training set –, we severely deteriorate the network's capacity to retrieve these samples. We surmise that once it has reached convergence, the cross-entropy loss encourages the model to assemble all the points from each class together and form tightly knit clusters, and to separate these groups from each other as much as possible. This leads to it not being able to differentiate individual points through their influence, and hence, to not be capable of detecting these artificially generated mislabeled images.



(a) EfficientNetB0



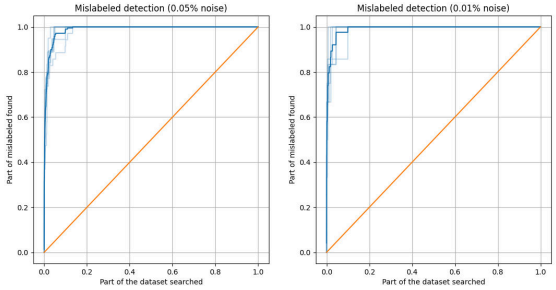
(b) ResNet-20



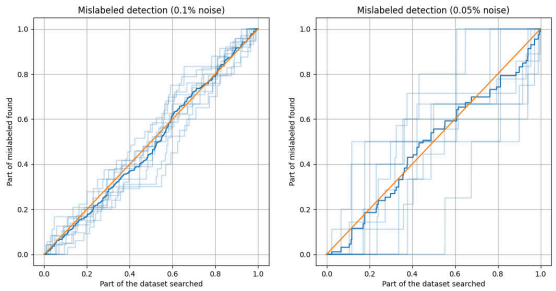
(c) VGG-19

Fig. 5: ROC curves for the detection of mislabeled examples for both noise regimes. In each case, we plot the mean ROC in solid blue, each individual ROC in transparent blue, and the random baseline in orange.

These two phenomena of performing better when constrained by the `L1L2` regularization and worse when overfitting are the two sides of the same coin: once an unconstrained network starts approaching convergence, it can decrease the surrogate classification loss (i.e. the negative log-likelihood or cross-entropy loss) without altering the 0-1 loss – the actual loss we would like to optimize, that indicates whether an element was correctly classified – by increasing its Lipschitz constant [4]. This can be easily demonstrated by leveraging some of the *softmax* activation function's properties when the neural network has already achieved the 100% accuracy in the training dataset, but this phenomenon has been observed to also occur slightly before reaching this point [4]. In layman terms, this means that the model will attempt to maximally



(a) ResNet-20 with `L1L2`



(b) ResNet-20 with slower schedule

Fig. 6: ROC curves for the detection of mislabeled examples for both noise regimes when training the model with an `L1L2` constraint, and a slower *learning rate* schedule. In each case, we plot the mean ROC in solid blue, each individual ROC in transparent blue, and the random baseline in orange.

distance the points from different classes, theoretically converging to N_{class} clusters of points infinitely far away from each other in the form of a simplex [24].

We leave for future work the analysis of when this happens and how to guarantee that our model has been correctly trained for useful information retrieval through influence functions.

IV. Conclusions

Initially applied to simple models and computed in its exact form, the influence function fell into disuse when models and datasets started to grow, making the procedure practically intractable. However, with the advent of frameworks that efficiently implement *auto-differentiation* and the popularization of GPUs capable of greatly accelerating computation times, it became possible to develop an approximate version specific to neural network models.

In this work, we have successfully leveraged them to determine influential points in the dataset and retrieve potentially problematic regions on the ACAS Xu use-case, and for the mislabeled example detection on the CIFAR-10 image classification dataset. In both cases, despite the size and complexity of the models at hand, the results were quite impressive, and come to show of its usefulness in real-life scenarios.

Acknowledgments

This work was conducted as part of the DEEL project[1]. Funding was provided by ANR-3IA Artificial and Natural Intelligence Toulouse Institute (ANR-19-PI3A-0004). The authors thank Florence de Grancey, Claire Pagetti and Adrien Gauffriau for their input on our results on the ACAS Xu problem.

References

[1] Mihaela van der Schaar Ahmed M. Alaa. Discriminative jackknife: Quantifying uncertainty in deep learning via higher-order influence functions. In *International Conference on Machine Learning*, 2020.

[2] Samyadeep Basu, Philip Pope, and Soheil Feizi. Influence functions in deep learning are fragile, 2021.

[3] Samyadeep Basu, Xuchen You, and Soheil Feizi. On second-order group influence functions for black-box predictions. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 715–724. PMLR, 13–18 Jul 2020.

[4] Louis Béthune, Alberto González-Sanz, Franck Mamalet, and Mathieu Serrurier. The many faces of 1-lipschitz neural networks, 2021.

[5] R. Dennis Cook and Sanford Weisberg. Characterizations of an empirical influence function for detecting influential cases in regression. *Technometrics*, 22(4):495–508, 1980.

[6] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding, 2016.

[7] Mathieu Damour, Florence De Grancey, Christophe Gabreau, Adrien Gauffriau, Jean-Brice Ginestet, Alexandre Hervieu, Thomas Huraux, Claire Pagetti, Ludovic Ponsolle, and Arthur Clavière. Towards certification of a reduced footprint acas-xu system: A hybrid ml-based solution. In Ibrahim Habli, Mark Sujan, and Friedemann Bitsch, editors, *Computer Safety, Reliability, and Security*, pages 34–48, Cham, 2021. Springer International Publishing.

[8] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[9] R. Giordano, W. Stephenson, Runjing Liu, Michael I. Jordan, and T. Broderick. A swiss army infinitesimal jackknife. In *AISTATS*, 2019.

[10] Ryan Giordano, Michael I. Jordan, and Tamara Broderick. A higher-order swiss army infinitesimal jackknife, 2019.

[1] www.deel.ai

[11] Frank R. Hampel. The influence curve and its role in robust estimation. *Journal of the American Statistical Association*, 69(346):383–393, 1974.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[13] Louis A. Jaeckel. The infinitesimal jackknife. https://faculty.washington.edu/fscholz/Reports/InfinitesimalJackknife.pdf.

[14] Kyle D. Julian, Mykel J. Kochenderfer, and Michael P. Owen. Deep neural network compression for aircraft collision avoidance systems. *Journal of Guidance, Control, and Dynamics*, 42(3):598–608, Mar 2019.

[15] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[16] Philipp Koehn. Europarl: A parallel corpus for statistical machine translation. In *MTSUMMIT*, 2005.

[17] Pang Wei Koh, Kai-Siang Ang, Hubert H. K. Teo, and Percy Liang. On the accuracy of influence functions for measuring group effects, 2019.

[18] Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1885–1894, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

[19] Zhifeng Kong and Kamalika Chaudhuri. Understanding instance-based interpretability of variational auto-encoders, 2021.

[20] A. Krizhevsky. Learning multiple layers of features from tiny images. 2009.

[21] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

[22] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2015.

[23] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics.

[24] Vardan Papyan, X. Y. Han, and David L. Donoho. Prevalence of neural collapse during the terminal phase of deep learning training. *Proceedings of the National Academy of Sciences*, 117(40):24652–24663, Sep 2020.

[25] Garima Pruthi, Frederick Liu, Mukund Sundararajan, and Satyen Kale. Estimating training data influence by tracking gradient descent. *CoRR*, abs/2002.08484, 2020.

[26] Bernhard Schölkopf, Ralf Herbrich, and Alex J. Smola. A generalized representer theorem. In David Helmbold and Bob Williamson, editors, *Computational Learning Theory*, pages 416–426, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[27] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[28] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[29] Trond Steihaug. The conjugate gradient method and trust regions in large scale optimization. *SIAM Journal on Numerical Analysis*, 20(3):626–637, 1983.

[30] Yi Sui, Ga Wu, and Scott Sanner. Representer point selection via local jacobian expansion for post-hoc classifier explanation of deep neural networks and ensemble models. In *Thirty-Fifth Conference on Neural Information Processing Systems*, 2021.

[31] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks, 2020.

[32] Chih-Kuan Yeh, Joon Sik Kim, Ian En-Hsu Yen, and Pradeep Ravikumar. Representer point selection for explaining deep neural networks. *CoRR*, abs/1811.09720, 2018.

Towards the certification of vision based systems: modular architecture for airport line detection

Esteban Perrotin[12] Matthieu Roy[2] Ariane Herbulot[2]
Michel Devy [2] Fabrice Bousquet[1]

[1] AIRBUS Operation
[2] LAAS-CNRS

esteban.perrotin@airbus.com

Abstract

This paper addresses the certification issues for vision based systems embedded on civil aircraft to assist pilots during the ground navigation phase. We propose a design methodology, through the example of a modular architecture to detect ground mark lines in a color image. Our detection method is based on the combination of classical image processing algorithms, deep learning methods and a particle filter algorithm. We argue that the main interest of this architecture is to ease the certification problem when compared to an end-to-end neural network. We discuss about difficulties around certification and propose some arguments.

1 Context

Computer vision applications have made considerable progress in recent years, with applications to many fields, from healthcare to autonomous vehicles. In this context, the civil aeronautic is considering using vision based systems to support pilots in their operations during the ground navigation tasks. For instance, functions such as obstacle detection, axis keeping, runway detection and others are currently studied. However, due to the complexity of such functions, it is challenging today to achieve the certification of systems largely based on computer vision algorithms in civil aeronautic field.

Actually, all functions embedded in civil aircraft have to be compliant to standard design processes defined by the authorities. These processes are described in documents such as Certification Specification [1], the ARP-4754a [2] and DO-178-C [3]. These documents define complete guidelines to design, develop and implement functions to be embedded on civil aircraft. This whole process is called certification. The standard certification process follows a V-model. It consists in designing, developing, implementing, verifying and validating a system by following specific steps described in guidelines. An example of V-model is shown in Figure 1. More details are provided in the section 2.1.

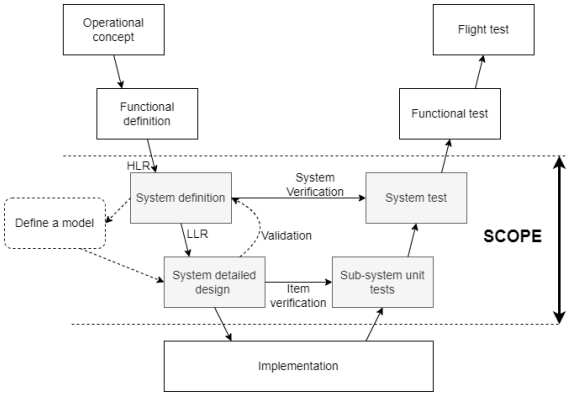


Figure 1: General V-model and our scope.

Some functions embedded on civil aircraft are already using vision based systems (VBS). For example, the A380 from AIRBUS is equipped with a camera integrated in the aircraft fin. The video input can be sent to both passengers and pilots. The pilot can use the video output as an help to drive the aircraft during the taxi phase. The image is directly used and interpreted by the pilot without any algorithm used for its interpretation. However, actual certified functions based on vision systems let the pilot as the principal actor to extract and interpret information represented

in each image. New functions, such as line detection, motion estimation or obstacle detection are using complex algorithms to extract and process features from the image in order to improve the pilot assistance. The certification of such algorithms has never been done in the European civil aeronautic field.
That is why, previous studies [4] highlighted the fact that the design of computer vision-based systems opens a new paradigm in certification. These difficulties are mostly due to the complexity of algorithms. For example, most of recent computer vision algorithms are based on machine learning (ML). Machine learning algorithms learn through examples how to analyze images in order to achieve a desired task. This approach is not compliant with standard design process [5]. Actually, current standards require an explicit description of detailed requirements. This is the opposite of ML systems which they learn the operational behaviors through examples and not from requirements. In order to prepare a new standard, the European Union Aviation Safety Agency (EASA) is working on guidelines [6] [7] to safely design such systems. There are also many public research projects on this topic. The DEEL project is also currently working on this problematic and and have published a White Paper [8]. They address many questions on the risks, challenges and potential solutions of using ML in systems submitted to certification constraints. However, vision based algorithms are not limited to machine learning.

In this paper, we present our position with regards to VBS certification issues. In order to illustrate how to solve these problems, we will take the example of line detection on airport runways/taxiways and describe our methodology to design the modular architecture of a specific line detection algorithm. Line detection is a useful function that could make easier the ground navigation for the aircraft pilot. Due to its simplicity, it is probably one of the first task using vision based algorithms which could be certified. However, most of recent methods repose on end to end deep learning architecture [9]. Because of the certification aspects, we desire to construct a functional architecture without using end to end deep learning methods. Figure 2 shows an example of system architecture that could be used to guide an aircraft on a taxiway, and/or to display information to the pilot. We will focus our discussions on the design and tests of this line detection function architecture. We will not discuss the hardware implementation or

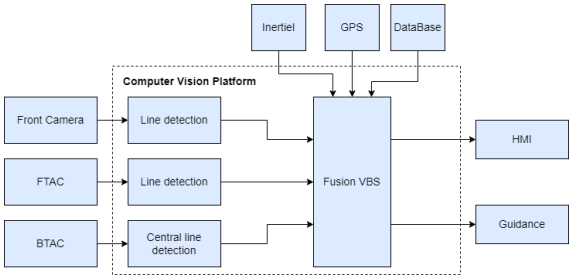


Figure 2: Example of visual based functions integrated in a civil aeronautic system

interface with other systems. The input of the function is an image provided by the Fin Taxi Aid Camera (FTAC) of the aircraft. We make the assumption that this image gives a correct representation of the real world.

The paper is organized as follows: Section 2 presents difficulties that could be encountered in the certification of computer vision based systems. Section 3 presents our proposed architecture in the specific line detection application. Section 4 explains our arguments to help certification of this architecture. Then the last section concludes.

2 General aspect of the certification of Computer Vision-based systems

Computer vision-based algorithms can be summarized in two main parts: extract from each image discriminant information (often called features) and then process this information to achieve the desired operation. Extracting pertinent information is a difficult task. An image can be seen as a 2D signal and the undesired information can be seen as a structured noise. In our application, we aim at extracting features that characterize lines. Hence, we desire to remove all other elements of the image.
In order to answer such problems, complex algorithms of computer vision are used. We will show in the following section that standard guidelines used for the development of civil avionic function are difficult to be applied on complex computer vision items.

In this section, we briefly present general aspect of certification, then we expose steps that could lead to difficulties in the certification of vision based algorithms and expose some elements from the state of art.

2.1 General aspects of certification

The principal guidelines of the civil aeronautics certification is provided by standards such as ARP-4754A [2] and DO-178C [3]. To design a new avionic high level function, the first point of the design process consists in analyzing the impact of this function in case of potential hazards. This step is called functional hazard analysis (FHA). For instance, if a failure of the function could lead to endanger the aircraft or passenger safety, the function will probably be classified as "catastrophic". It will be necessary to prove that the probability for the failure of this function during a flight hour is less than 10^{-9}. This FHA step will drive requirements that should be verified by the implementation of the function.

Then, this high-level function is refined into lower level functions and a preliminary functional architecture is proposed. This architecture is submitted to a preliminary system safety assessment (PSSA). It verifies that requirements from the FHA are fulfilled by the architecture. If requirements are not fulfilled the architecture should be consolidated or a new one should be proposed.

Once a correct architecture is proposed, for each item's architecture a Deasign Assurance Levels (DAL) is addressed. It drives hardware objectives for the specific item and software guideline activities to develop the function. The highest critical level is of DAL A and the lowest is of DAL E. For instance, in case of a DAL A item, the software will be executed on different hardwares reducing the risk of hardware failure and ensuring the availability of the function at every time.

During the design process, multiple tests have to be defined to guarantee and verify that the implementation of items is correct and fulfill requirements at each level. Tests are driven by the system safety assessment (SSA) to ensure the availability and the performance of the function in all identified scenarios.

This safety design process is used to develop high level safety functions for civil avionic system.

2.2 Difficulties around certification for VBS algorithms

The certification process ensures that the proposed high-level function is correctly designed and implemented following specific guidelines depending on the level of criticality of the function. Each sub-level item should verify specific requirements. However, in the case of computer vision systems some aspects of the guidelines seem complicated to satisfy, in particular in systems using machine learning. The authors in [5] and [10] discuss on this problematic for adaptive systems. In [4] the authors detail the certification challenges on a specific VBS dedicated for two applications: visual odometry and obstacle detection.
Considering this previous work and with regards to the difficulties we encountered during our development, we precise some points that seem difficult to answer considering the current certification standards:

- **Comprehensible requirements** have to be written. In computer vision applications writing high level requirements - HLR (*e.g.* detect a specific object) could follow the usual process. However, writing lower level requirements could be harder (e.g, explicit evidences that characterize a given object in all kind of environments).

- **Verifiable requirements**: In order to verify that a computer vision algorithm is correctly working, we have to present different inputs in the system and verify that the outputs correspond to the desired ones. Ideally, we would like to test all potential inputs. But, due to the high dimensionality of images ($255^{1024\times768\times3} \approx 10^{4718592}$ combinations for 1024px $\times$768px color images) it is not realistic to test all possibilities. Most of these images are pure noise or not relevant. A solution could be to provide a minimal dataset, with a distribution of images close to the real application. Furthermore, the desired outputs have to be known for each image tested. Manual labelling on a huge dataset seems impossible and it raises an issue on the accuracy on the labelling process. When using automatic or semi-automatic labelling one has to prove that the algorithm used for labelling is robust with regard to the performance for the high-level function. Validating an automatic labelling process raises similar certification activities. Section 2.3 contain more details about the acquisition of a test data set.

- **Robustness**: The recent advances in computer vision (in particular with deep learning methods) have greatly enhanced performance for many applications. However,these

performances must be defined and evaluated in particular in degraded conditions (fog, rain, low illumination, etc.). In such conditions with environmental hazards algorithms could perform poorly. Avionic systems have to operate correctly despite abnormal inputs and conditions.

In addition the stability of the results have to be considered. For a small variation in the input image, computer vision algorithms have to output the same value. There is no evidence that computer vision algorithms achieve this stability. In particular this case appears with deep learning methods and adversarial attack [11].

- **Traceability**: During the design process, the requirements of a given level are broken down into one or more requirements of the next level. It has to be shown that each low level requirement (LLR) corresponds to a requirement of the higher level (HLR).

- **Interpretability**: This point is not essential for the process of certification but it argues about trustworthiness of the system and helps to convince authorities about the viability of the system. In order to trust a functionality, each step of a function has to be explained and must have its proper purpose based on rationals. This is not true for machine learning based applications. Machine learning learn to extract and process features from the image through examples. Many works have been done to explain which features are extracted, how an interpretation is generated from these features, especially for deep learning methods [12]. However, this aspect should be accepted for classic computer vision algorithms.

2.3 Related works

To address these difficulties, many works have been done, for example to ensure robustness of algorithms, to discuss and propose methods for the creation of test data sets and develop solutions to interpret complex algorithms such as convolutional neural networks.

To ensure robustness of algorithms (accuracy and confidence on the output), methods have been proposed to compute bounces of confidence [13]. These methods could be based either on the demonstration of a formal proof that gives a guarantee about envelope of the algorithm output, or on the execution of enough tests to cover all potential use cases.

In computer vision, due to the complexity of algorithms, the large variety of input images and the lack of precise requirements, it is difficult to develop formal proofs for applications. However, this is already the case in many avionic systems. This is why functions are tested on large data tests. The construction of these test data sets raises many interrogations. Since it is not feasible to test every images, it is necessary to select pertinent scenarios to test algorithms. But algorithms have to be tested in many corner cases, so that acquiring real data for every desired scenario is impossible. A solution consists in generating synthetic images from simulators. Many works have been done to provide image generators useful to test computer vision algorithms for given applications. For instance, we use OKTAL-SE simulator. It generates color or infrared images based on a physical model. However, there is a gap between their simulated image and real image, in particular in the level of detail for the textures representation.

On the other hand, Zendel et al. [14] have proposed a system of guide-words to quantify and qualify hazards that could appear in computer vision applications. Their works provided an answer to "Which situations should be covered by the test data and have we tested enough to reach a conclusion?". By using CV-HAZOP, in [15] and in [16] they propose a standard procedure devised by the safety community to validate complex systems.

In the next section, we describe our line detection algorithm, so that we could illustrate how to deal with the verification of the properties summarized here above.

3 Modular architecture of our line detection method

As said previously, constructing an argument of certification for VBS is a difficult task. This section describes our methodology to construct our line detector. We start by modeling the problem using prior information. Then a general architecture solving this problem. Finally we describe each item in this architecture. Our principal objective is to ease the certification process by diminishing the complexity of verification and validation on each item of our architecture.

3.1 Problem modeling

As said in the previous section, an important point to follow in the certification process is the definition of requirements. The objectives of the function (high level requirements - HLR) should be as clear as possible. In our application, we aim at detecting lines (ground marks). That is why, we define the objectives of our function as the following example: "The function has to detect ground marks that define lines for aircraft ground navigation. The input of the function is an RGB image acquired by an embedded camera (typically in the fin or the cockpit as shown in Figure 3) and parameters set from operational concept. Internal and external parameters of the camera are known. Lines have to be detected until a desired distance to the camera. Output lines will be defined by a set of points.".

Figure 3: Input RGB image from the fin camera.

We start by constructing a model of the lines that will help to fulfill HLR. We construct this model by following some key points:

- Use the maximum of prior information to select important regions of interest (ROI) in the image. These regions have a high probability of containing desired information (presence or absence of ground marks). A ROI is constructed by using the parameters of the camera and the desired maximum range. In practice, it will mask unwanted elements like the aircraft (if the image is from the fin camera) or the sky.
- Construct it in order to facilitate broken high level requirements into lower level requirements. Construct a set of information that represents the object and its evolution (spatial and/or temporal). In our case, the HLR specifies the output: a set of points. Thus, we consider this set of points as the spatial propagation of points that describe a line. So, we have to detect a first point, define a dynamic law that describes the propagation of points, and precise an end criteria.
- Find and write most of the properties verified by the desired object in the image (size, structure, texture, etc.). These properties will help to verify that extracted information is correct. These properties could also help in writing an end criteria.

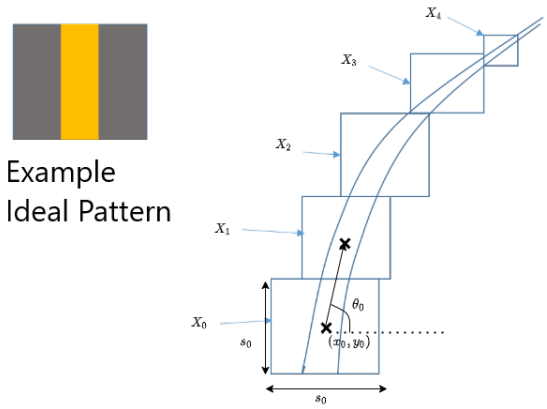


Figure 4: Model of a line.

Following these points, we model a line as a spatial repetition of a pattern (Figure 4). The pattern is considered known. The pattern will repeat himself in the presence of some noise (degradation, orientation, illumination variation, deformation, etc.). This repetition is done in respect to some properties provided by the International Civil Aviation Organization (ICAO) [17]: the curve made by central points of the patterns must satisfy constraints. In addition, depending on the illumination, the color of the painting on each pattern should be close to yellow [18]. The width of lines is known.

To describe a pattern in a thumbnail, we use a state vector $X_k = \{(x_k, y_k), \theta_k, s_k\}$. (x_k, y_k) is the position of the center of the thumbnail in the image, θ_k is the orientation of element in the thumbnail and s_k is the size of the thumbnail (considered as a square).
The evolution between state vectors is described by a function g, where g uses the current state vector X_k to update the next state vector X_{k+1}. The update depends also on external factor noted in an unknown noisy term U_k:

$$X_{k+1} = g(X_k, U_k) \tag{1}$$

This function is unknown and we consider a simplified model. We use the following equations to update the variables inside the dynamic model:

$$x_{k+1} = x_k + s_k cos(\theta_k) \quad (2a)$$
$$y_{k+1} = y_k + s_k sin(\theta_k) \quad (2b)$$
$$\theta_{k+1} = \theta_k \quad (2c)$$
$$s_{k+1} = min(max(s_k, a), b) \quad (2d)$$

Where the constants a and b determine the minimal and maximum size of thumbnails. It models the line's trajectory as a linear trajectory and supposes the two variables s_k and $theta_k$ are not evolving. This model is not quite precise. But the particle filter will correct the evolution of these parameters according to the observations from the image. In practice it is good enough to catch simple line patterns. In the future, we plan to add other variables to capture and describe environmental perturbations (such as the illumination, the principal color, etc.) and also enhance the dynamic model to improve performances.

3.2 General architecture

Using this model, we split our specification into three parts: find potential starting points of the line, recognize a pattern of the line from a thumbnail described by a state vector and predict the evolution of the pattern in the image. Each item has its own requirements driven from HLR and its own unit tests. The coverage of the HLR by LLR is ensured by the model. Figure 5 shows our design. Once the interface between modules is defined, it is important to note that each part can be developed independently.

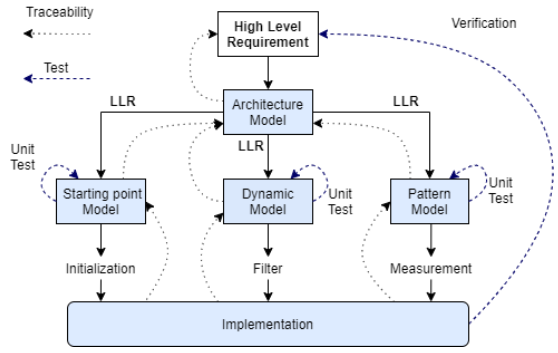


Figure 5: Design and test of our model.

Following the key points expressed above and using our model, we propose a modular architecture to detect lines from an image. The architecture of the system is described in Figure 6. The main point of our architecture is to separate prediction from a model to information extracted from the image (since the image can be considered as a complex noisy signal). The proposed algorithm works as follows:

1. The image is acquired by the camera and considered as a representative data of the reality.

2. Using the image, an initialization based on simple assumptions (such as gradient, color and position) will propose some initial conditions (potential starts of lines) as an initial state vector X_0.

3. Using the initial condition and the dynamic model of lines a filter will predict the next state of the line $\hat{X}_{k+1}$.

4. The prediction will be sent to an observation function that will attribute a measure corresponding to the probability of the correctness of the prediction $\hat{X}_{k+1}$ with regards to the image and modeled feature of the desired pattern.

5. Using the measures, the filter will update his predictive model: $\hat{X}_{k+1} \rightarrow X_{k+1}$.

6. Until an end criterion is verified, the program will repeat from step (3).

This algorithm detects one line. In practice, we applied this algorithm many times to detect many lines in the image. Some heuristics are used to discard some elements like the ridges of the lines (see Figure 3).

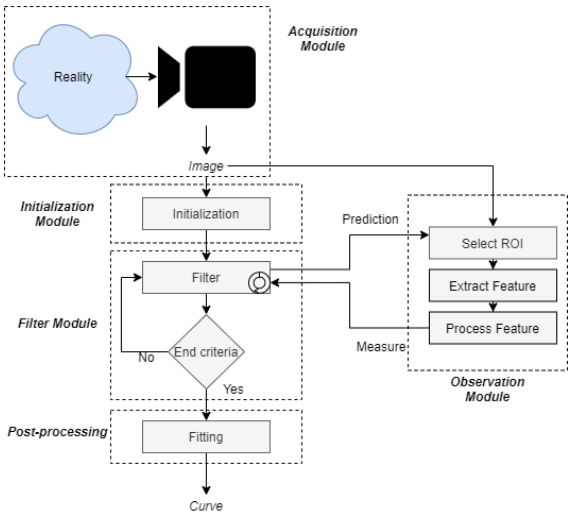


Figure 6: Simplified schema of our modular line detection architecture.

3.3 Description of modules

Now, we detail and explain our choice for each module.

Initialization module. The initialization function uses the image as input. It has to propose potential starting points of a ground line. To detect these starting points we developed an algorithm based on classical image processing, using gradient and color prior from the model of line. This algorithm is not detailed in this paper. The robustness of this function is low. However, in case of failure (wrong starting point), the proposed point might be discarded by the filtering and observation function. That is why we favor recall over to precision.

Filtering module. To predict the next state of the line, we have to estimate the solution from the equation 1. To solve this equation many filters can be used such as the Kalman filter. However, in practice, that model is limited due to the noisy term U_k. If the distribution of U_k was Gaussian, the Kalman filter would be a suitable solution. In our case, we choose to use a particle filtering algorithm similar to the one presented in [19]. Firstly because it can solve this problem without knowledge on the distribution of U_k. Secondly, to provide a first analysis on problems encountered by the certification process of particle filters. These filters are useful for navigating, positioning and tracking [20], they could be used in more applications in the future. Since our project is to put on the table new technologies, we made the choice to construct our line detector using a particle filter.

The particle filters are a set of Monte Carlo algorithms. The objective is to compute the posterior distribution of a stochastic process. The first step is to create particles $\{X_k^n\}_{n=1}^N$ from an initial prior distribution. Each particle corresponds to an estimation of the state vector. A weight w_k^n is associated with each particle. This weight corresponds to the confidence in a particle to represent the actual real state. The closer the particle is to the real state the higher its weight is. In most systems using particle filters the weight is computed by using a measure provided from external sensors (such as GPS or inertial sensor in the case motion estimation). In our case, we define a sub-item called "observation module" that attributes a weight to a particle using as input the image and the variables contained in the particle. This observation module is detailed below. Once all particles have an associated weight, the particle filter algorithm will re-sample particles using the distribution created by $\{(X_k^n, w_k^n)\}_{n=1}^N$. More information about particle filtering can be found in [21][22] and more about this particle filtering method are provided in a french paper [23].

The principal advantage of particle filter algorithms is their ability to solve nonlinear problems tainted by a noise without prior knowledge about this noise. The principal inconvenience of these algorithms is the computing time. The more a problem is complex, the more it will require particles to compute a good estimation. We don't cover computation time problematic in our approach. However, it could be possible to cover this issue by adapting the amount of particles depending on the confidence in the observation module [24].

Observation module. Using a state vector provided by the filter (through a particle), it selects a thumbnail in the image (between 15 and 51 pixels depending on the variable s_k in the state vector), resize the thumbnail to 33×33 pixels and attributes it a score. This score is between 0 and 1 and evaluates the pertinence of components in the state vector in regard to the thumbnail. The score is the product of the output of tree sub-items:

- The first one is a binary classifier. It scores the presence or absence of ground marks in the thumbnail. After a preliminary study [23], we selected a small convolutional neural network (CNN) as classifier. This choice is also motivated to provide a real application to use CNN in a restrained context controlled by other elements. By using a CNN at this part of the line detection method, it permits to achieve correct result in the line detection without using end-to-end CNN. To train this CNN we constructed a dataset of 4000 thumbnails with manual annotation (50% positive and 50% negative). The score used is directly the output of the classifier. More details about the creation of the CNN are provided in [23].

- The second item measures correlation between the orientation θ_k in the state vector and the principal orientation in the thumbnail. We compute the orientation of the element in the thumbnail using a method based

on the Gabor filter.

- The last item measures the centering of the line in the thumbnail using classical image processing techniques such as gradient, color processing, Otsu's binarization and Hough transformation. It extracts the principal straight yellow lines in the thumbnail. Then it computes a score (between 0 and 1) depending on the projection of the center of the thumbnail on the line equation.

The product of these scores produces the final score that will serve as the weight for the particle in the filter.
In the next section, we discuss the verification and validation of these modules.

4 Certification Arguments

This section presents our arguments and thoughts about the feasibility of the certification for this architecture. This architecture proposes to reduce the amount of tests required to verify the system by using unit tests on each item. The principal assumption is: "If each item works correctly the function works correctly". This assumption holds because high level requirements are covered by lower level requirements. Each item has a simple and comprehensive task. The next parts develop discussion and arguments about verifying whenever each item is doing its task correctly.
This modular architecture should make the certification easier. The more an architecture is decomposed on items the more we can hope to reduce the complexity of each item and facilitate the creation of tests to verify performance of each item. It also helps to implement monitoring systems. Each item can have its own monitoring system and improve confidence on its results. In addition, when a module is changed (or improved) the certification should be done for this module only. It can significantly reduce certification cost when updating a system.

4.1 Filtering module

From the point of view of certification, Monte Carlo methods are often used to test algorithms but are generally discarded in embedded avionic systems. This is partially due to the fact that these algorithms use random number generation and one requirement for certification is repeatability. However, it is possible to generate offline a set of distributions that verify desired prior distribution. In addition, some particle filter algorithms use fewer random steps. A survey of recent advances in particle filtering can be found in [25].

Errors in the particle filter could occur depending on three conditions. Firstly, if the dynamic model is incorrect and does not match the real trajectory of the line. Secondly, if the prior distribution is not realistic (*e.g.* if the distribution has a huge bias). Thirdly, if the measure provided by the observation module is wrong and does not bring information about the evolution of the dynamic system. Figure 7 shows these potential errors.

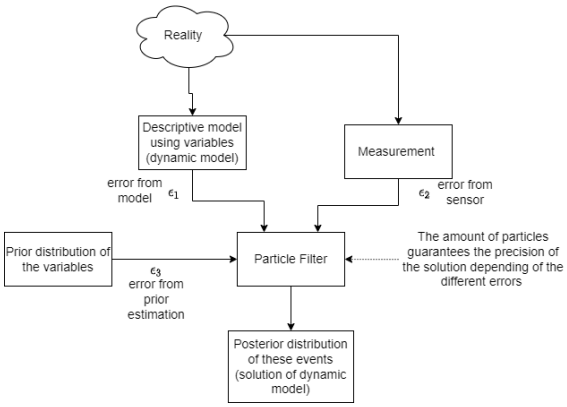


Figure 7: Potential errors of the particle filter

To test and validate performances of the filter, we build a generator of lines. An example of the generated line is shown in Figure 8. In this generator, it is possible to compute the perfect measure (it is computed by checking if the position of the particle is on the line or not and if the orientation of the particle is correct or not). We selected a prior distribution as a Gaussian distribution, where the variance is fixed in regards to the width of the line. These tests allow validating that within our dynamic model and prior distribution this method can follow line until a certain curve and depending of the amount of particles.
Furthermore, authors have been conducting research to prove the convergence of the particle filter [26] or computing bound of errors in particle filtering algorithms depending on incorrect model assumption [27]. Also, they have proposed solutions to enhance the algorithm in presence of noise and bias on the measure [28]. In regards to the state of the art, it should be possible in some situations to use formal proof to guarantee confidence of the particle filter. Otherwise, the feasibility of tests ensure the possibility to compute performance on such algorithms.

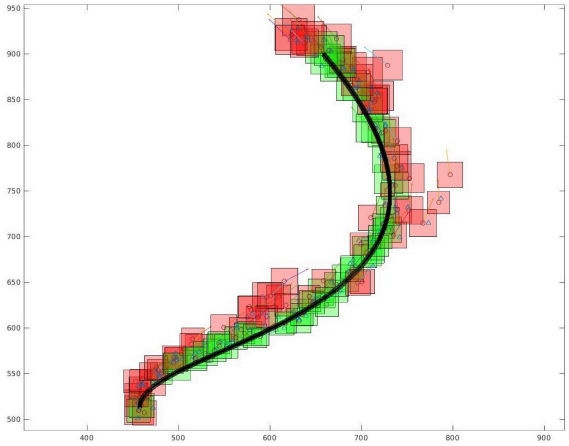


Figure 8: Example of test to validate the particle filter under the proposed dynamic model. In white the generated line. In green, particles with high weight (positioned on the line). In red particles with low weight (not on the line).

4.2 Observation module

The observation module obtains as input a state vector from the filtering module and an image from the camera acquisition. It has to judge the pertinence of the state describing a line pattern. The first step of this module is to create a thumbnail by selecting a specific region of interest in the image using information in the state vector. Then it attributes a measure by combining the result of three sub functions. The first one is a classifier that predicts the presence or absence of the desired pattern in the thumbnail. The second one is a function that estimates the principal orientation of elements in the thumbnail. And the last is a function that estimates if the element is centered in the thumbnail.

Classification. We provide here a minimal scope to use deep learning methods. Instead of using them to achieve complex tasks in high dimensional images, we reduce the problem to a simple binary classification of small thumbnails. In addition, the output of the classifier does not require a very high integrity. As said previously, the filtering module accepts a part of error from the measure. It is acceptable for the classifier to make some wrong prediction.

In this context, it is still not possible to test the classifier on every possible image ($256^{3\times33\times33} \approx 10^{7843}$). At the same time it should be possible to compute confidence bounds for the classifier by using methods proposed in the state of art. For example, the Pac-Bayes theory [29], adversarial method [11] or bounce generalization[30] seem promising.
This part is still ongoing and at the moment the only confidence in our classifier comes from a data test consisting of approximately 4000 thumbnails (50% positive and 50% negative). This dataset is constructed by manual annotation and we lack arguments in the confidence of this test. Figure 9 shows examples of thumbnails of the dataset. Our convolutional neural network achieved 88.3% of accuracy on training and 87.2% on the test data set.

Figure 9: First line shows thumbnails considered as positive. Second line shows thumbnails considered as negative.

Orientation and center measure. The orientation measure defines the general orientation of a line in the thumbnail. The orientation is found by filtering the thumbnail using the Gabor filter on every possible orientation. The maximum response of the filter appears when the direction of the line is in the same orientation that the filter. In the case of absence of line in the thumbnail, the response of the filter is not predictable. This function is checked from a dataset similar to the one built for the classifier. In addition, because of the pattern of line, we can ensure that the Gabor filter has one of the best response when the line is the principal element in the thumbnail.
Another measure consists in determining whenever the line is or not in the center of the thumbnail. This is done by geometric image processing without much difficulty. It is verified on the same data set as for orientation measure.

4.3 Initialization

The initialization extracts features that lines should verify: color, gradient and position assumptions. From camera parameters we defined the region of interests where the starting points of the lines can be. In each region, we use color processing to separate "yellow" elements from others. Then we use the morphological operator and skeleton algorithm to select potential starting points. Due to the use of color processing, environmental hazards impact the result. To im-

prove performance and robustness against illumination variation and other hazards, the defined color "yellow" is adapted in each region by using a method proposed in [18]. In case of a false starting point, other modules should detect and discard this point (absence of ground marks, length of line, etc.). It is still possible to miss a line because the minimal requirements are not fulfilled (illumination, camera resolution, etc.).

4.4 Verification

The final verification is done qualitatively on real images from different scenarios. Figure 10 shows results to illustrate the function. Points of the same color correspond to the result of the filter at each iteration. Red curves correspond to the quadratic regression of these points. Currently, the system is not designed to be robust to occlusion. As we can see, the initialization step captures only three lines. Lines far away from the camera are not detected because of this step. The architecture is designed to reduce false positives (erroneous detection).

Figure 10: Results with our architecture.

5 Conclusion and future works

This paper focuses on the design methodology of vision based algorithms. We proposed a modular architecture for line detection applications designed to ease the certification process. This proposition avoids the use of end to end deep learning methods. They seem to have better accuracy but they are not compliant with the actual certification process. The main interest of decomposing vision architecture into smaller parts is to facilitate the generation of tests. Each part of the architecture has its own specifications and its own test data set. In addition, it provides a minimal restrained context where complex algorithms, such as deep learning methods, could be studied. Also, it should be easier to update and improve components one by one.
To build this architecture, we propose a general method describing the spatial or temporal dynamic of the visual object and decompose the architecture into two parts. The first one will predict features that should describe the object. The second one will check in the image if this feature corresponds to reality. This architecture should work for object tracking.
We are currently working to build larger test data sets and a monitoring system using temporal information (video processing). The question about test data sets is still ongoing. The use of guidelines such as CV-HAZOP helps the identification and construction of potential hazardous scenarios. However, data from these scenarios are not easily acquired and we have to base tests from simulators. It raises the question about the certification of such simulators.

ACKNOWLEDGMENT

This work was supported by AIRBUS Operations. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views AIRBUS Operations. We would like to thank the OKTAL-SE company for providing the simulation tool, used to build synthetic images.

References

[1] EASA, *Certification Specifications and Acceptable Means of Compliance for Large Aeroplanes CS-25*. 2007.

[2] SAE., *ARP4754a - Aerospace Recommended Practices (Development of Civil Aircraft and Systems.)*. 2010.

[3] RTCA, *DO-178C, Software Considerations in Airborne Systems and Equipment Certification.* 2008.

[4] F. Boniol, A. Chan-Hon-Tong, A. Eudes, S. Herbin, G. Besnerais, C. Pagetti, and M. Sanfourche, "Challenges in the certification of computer vision-based systems," 09 2020.

[5] S. Bhattacharyya, D. Cofer, D. Musliner, J. Mueller, and E. Engstrom, "Certification considerations for adaptive systems," *2015 International Conference on Unmanned Aircraft Systems, ICUAS 2015*, 2015.

[6] EASA and Daedalean, "Concepts of design assurance for neural networks (codann) ii," tech. rep., 5 2021.

[7] EASA, "Easa concept paper: First usable guidance for level 1 machine learning applications," tech. rep., 2021.

[8] H. Delseny, C. Gabreau, and A. G. et al., "White paper machine learning in certified systems," *CoRR*, vol. abs/2103.10529, 2021.

[9] Z. Wang, W. Ren, and Q. Qiu, "Lanenet: Real-time lane detection networks for autonomous driving," 2018.

[10] E. Mirzaei, C. Thomas, and M. Conrad, *Safety Cases for Adaptive Systems of Systems: State of the Art and Current Challenges*, pp. 127–138. 09 2020.

[11] Y. Lin, H. Zhao, X. Ma, Y. Tu, and M. Wang, "Adversarial attacks in modulation recognition with convolutional neural networks," *IEEE Transactions on Reliability*, vol. 70, no. 1, pp. 389–401, 2021.

[12] C. Molnar, *Interpretable Machine Learning*. 2019.

[13] P. Germain, A. Lacasse, F. Laviolette, M. March, and J.-F. Roy, "Risk bounds for the majority vote: From a pac-bayesian analysis to a learning algorithm," *Journal of Machine Learning Research*, 2015.

[14] O. Zendel, M. Murschitz, M. Humenberger, and W. Herzner, "Cv-hazop: Introducing test data validation for computer vision," in *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015.

[15] O. Zendel, K. Honauer, M. Murschitz, M. Humenberger, and G. Fernandez Dominguez, "Analyzing computer vision data - the good, the bad and the ugly," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[16] O. Zendel, M. Murschitz, M. Humenberger, and W. Herzner, "How good is my test data? introducing safety analysis for computer vision," *International Journal of Computer Vision*, vol. 125, pp. 1–15, 12 2017.

[17] ICAO, *AERODROMES: aerodromes design and operations*. International Civil Aviation Organization (ICAO), 2018.

[18] C. Meymandi-Nejad., E. Perrotin., A. Herbulot., and M. Devy., "Colorimetric space study: Application for line detection on airport areas," in *VEHITS*, 2021.

[19] C. Meymandi-Nejad, S. E. Kaddaoui, M. Devy, and A. Herbulot, "Lane detection and scene interpretation by particle filter in airport areas," in *VISAPP*, 2019.

[20] F. Gustafsson, "Particle filter theory and practice with positioning applications," *IEEE Aerospace and Electronic Systems Magazine*, vol. 25, no. 7, pp. 53–82, 2010.

[21] B. Ristic, S. Arulampalam, and N. J. Gordon, "Beyond the kalman filter: Particle filters for tracking applications," 2004.

[22] A. Doucet and A. M. Johansen, "A tutorial on particle filtering and smoothing: Fifteen years later," 2008.

[23] E. Perrotin, C. Meymandi-Nejad, A. Herbulot, M. Devy, and F. Bousquet, "Détection des lignes aéroportuaires par méthode de filtrage particulaire: Évaluation de fonctions d'observations," in *ORASIS 2021*.

[24] X. Zhang, J. Peng, W. Yu, and K.-C. Lin, "Confidence-level-based new adaptive particle filter for nonlinear object tracking," *International Journal of Advanced Robotic Systems*, vol. 9, no. 5, p. 199, 2012.

[25] X. Wang, T. Li, S. Sun, and J. Corchado Rodríguez, "A survey of recent advances in particle filters and remaining challenges for multitarget tracking," *Sensors*, vol. 17, p. 2707, 11 2017.

[26] X.-L. Hu, T. B. Schon, and L. Ljung, "A general convergence result for particle filtering," *IEEE Transactions on Signal Processing*, vol. 59, no. 7, pp. 3424–3429, 2011.

[27] N. Vaswani, "Bound on errors in particle filtering with incorrect model assumptions and its implication for change detection," in *2004 IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2004.

[28] F. Abdallah, A. Gning, and P. Bonnifait, "Box particle filtering for nonlinear state estimation using interval analysis," *Automatica*, vol. 44, no. 3, pp. 807–815, 2008.

[29] K. Pitas, M. Davies, and P. Vandergheynst, "Pac-bayesian margin bounds for convolutional neural networks," 2018.

[30] P. M. Long and H. Sedghi, "Size-free generalization bounds for convolutional neural networks," *CoRR*, vol. abs/1905.12600, 2019.

Session We.4.B

Simulation

Wednesday 1st June

17:00

–

Room Lauragais

Combining Real and Virtual Electronic Control Units in Hardware in the Loop Applications for Passenger Cars

Jan Röper[1] Arpita Bhattacharjee[2] Franz Kramer[3] Purushottam Kuntumalla[2]
Andreas Junghanns[3]

[1]Mercedes-Benz AG jan.roeper@daimler.com
[2]Mercedes-Benz Research and Development India Pvt. Ltd.
arpita.bhattacharjee3@gmail.com purushottam.kuntumalla@daimler.com
[3]Synopsys GmbH {franz.kramer, andreas.junghanns}@synopsys.com

Abstract

For the testing of modern electronic control unit (ECU) software, different test-platforms like Software-in-the-Loop (SIL) and Hardware-in-the-Loop (HIL) but also in vehicle testing are used. To ensure a realistic behavior of the software in SIL and HIL testing, models of the environment as well as residual bus simulation are required. A detailed representation of other controllers in the vehicle network that feeds into the residual bus simulation is needed to test complex functions of the software and to close distributed control loops. We present a novel approach, where a virtual ECU designed for SIL testing is reused to generate the input to the residual bus simulation. For export from the SIL tool and import into the HIL modelling environment, a C-Code functional mockup unit is used. To show the benefit of this approach, the setup is compared to an existing setup, which uses both a simplified model of an ECU or the real ECU. In addition, drawbacks of the presented approach are presented and the potential of other methods is discussed.

1. Introduction

For the testing of modern electronic control units (ECU), multiple test environments exist. In the early stage, these include software in the loop (SIL) and hardware in the loop (HIL) testing. In SIL environments, the ECU code is executed on a standard PC, which requires virtualization of the ECU [1]. In HIL testing, the ECU code is executed on the target hardware, which is connected to a simulator for stimulating input signals and measuring output signals. In both cases, a complex simulation of the environment of the device under test (DUT) is required, which consists of plant models and a simulation of the communication also known as residual bus simulation (RBS). The RBS stimulates the communication interfaces of the DUT using both static fixed values as well as dynamic signals. This is necessary to ensure that the test conditions match the conditions in the real application, which in this case, is the control of an automatic transmission in a passenger car by a transmission control unit (TCU). Additional features of a RBS are the manipulation of signals, messages, message counters and various other mechanisms that control the data flow in an ECU network.

A major task in the development of a HIL test environment is to provide a mechanism to generate the dynamic signals for the RBS. One approach is to provide the dynamic signals through the test automation that executes test scripts. However, for complex controllers with multiple parallel control loops, this approach is unfeasible. An alternative can be simulated ECUs, where the basic functionality of an ECU is modeled in the same fashion as a plant. For an aggregate component HIL environment, multiple simulated ECUs provide the dynamic signals to the RBS to satisfy the DUT with adequate stimulus. While simulated ECUs allow for a more detailed representation of the DUT environment, their implementation effort is high: the ECU to be modeled has to be analyzed, the targeted functionality reengineered and the result tested on the HIL with the DUT. Additional challenges are the need for parameter management based on the specific variant of the simulated ECU and the incessant demand by testers to add further functionalities to the simulated ECU. In this paper, we present a novel approach where virtual and real ECUs are combined in a HIL simulator for component testing.

2. Previous Work

Because of the shortcomings of the simulated ECU concept, multiple approaches have been made to replace them with a more desirable design. Common goal of all these approaches is to reduce the engineering effort by reusing code or precursors of the code of the targeted real ECU. One approach is to import fragments of the models, which are the base for the ECU code into the HIL model development tools. While this approach is straightforward, it leaves the developer with the problem that the interfaces of these fragments do not directly correspond to the dynamic signals of the RBS. For proper integration, wrapper models are necessary. However, this approach faces the same challenges as the simulated ECU approach. In turn, the reuse of the code for smaller ECUs as discussed in [2] is carried out in the same fashion by building a wrapper around the ECU code. This process resembles the process of virtualizing ECUs for SIL applications mentioned above but is specific to the specific HIL environment. Another approach, presented by [3], employs a SIL that runs on a

standard PC in parallel to a HIL. The SIL communicates with the HIL and the DUT via bus systems. This represents a distributed RBS, where the part represented by the PC is not a hard real time system but runs in a standard Windows setup with additional tweaks that ensure acceptable stability. In this scenario, full reuse of an existing SIL reduces the effort for providing dynamic signals to the RBS. At the same time, this approach leads to more complexity, as the test automation has to control two environments and special mechanisms for manipulating signals in both RBSs are necessary. [4] discuss a similar scenario, where a virtual ECU is executed on a separate system, which communicates with a HIL and other test systems. A complex control system is presented that handles the resulting hybrid test system. This work does not state whether the virtual ECU running on a separate system is operating in real-time. Table 1 summarizes the above-mentioned approaches to generate dynamic signals for the RBS and their most relevant properties.

Table 1: Types of Sources for Dynamic RBS Signals

Signal Generation Type	Relevant Properties
script in test automation	limited to non-parallel control loops
simulated ECU	high effort for reengineering
reuse of ECU code fragments or precursors	wrapper for integration required
virtual ECU in a SIL on separate PC	distributed system with stability and automatization challenges
real ECU	replaces RBS; limited ability for manipulation; late availability

3. Proposed Approach

The approach presented in this paper combines the reuse of existing code in a SIL by [3], while maintaining the integrity of the RBS as described by [2]. A C-Code based virtual ECU of a powertrain control unit (PCU) that was originally designed for the usage on a Windows PC was adapted for platform independence. The effort and cost for adapting a virtual ECU for platform independence depend on the specific virtual ECU and can vary drastically. In addition, cost and effort are hard to estimate in advance and the complete virtual ECU has to be available as C-Code. The resulting platform independent virtual ECU represents a pretested black box for the HIL developer and can be parameterized for different variants before the export from the SIL environment.

The C-Code functional mockup unit (FMU), which was originally designed for plant model exchange, is chosen as a container for the exchange of the platform independent virtual PCU [5]. An important argument for using a C-Code FMU in this scenario is the support by many different simulator platforms. The ability of FMUs to be used as a container for ECU code by exporting a plant model to an actual ECU is shown in [6], while in [7] FMU is only used to import models into a HIL environment and a proprietary format is used to import controller code that is not further described. For future applications, the FMU for embedded systems (eFMU), as described by [8], can also be a potential container for the code of a virtual ECU. Like FMU, eFMU are designed for plant models as well but with the goal of integration in an embedded real time target. Specifically the features to add information that is relevant for the build process, e.g. target compiler options, can be useful when it comes to integrating a virtual ECU in a HIL environment.

After importing the virtual PCU in its FMU container into the build environment of the HIL application, all rest bus relevant signals are routed to the respective RBS model interfaces. The real PCU that is replaced by the virtual PCU uses CAN, Flexray and LIN as bus systems. An overview of the challenges faced when integrating the virtual PCU with the existing RBS model is presented below. Some of these challenges result from the characteristics of the RBS models, while others result from the characteristics of the different bus systems.

4. Generation of Virtual ECU FMU

The virtualized PCU is a level two virtual ECU according to [9]. It contains the full application layer and a simulation basic software providing necessary functionality to increase test coverage of the software under test. The tool for creating and simulating the virtual ECU is Synopsys Silver®, a virtual ECU tool available for Windows and Linux PCs. Notable components of the simulated basic software layer are signal and PDU based COM for multiple bus systems including network routing, NVM for nonvolatile memory and a replacement OS. This level of abstraction of this ECU was chosen as it provides high complexity with multiple bus systems and large target code size in application layer, without involving multiple stakeholders in the code to be virtualized. Integrating target code deeper in the ECU stack should pose little additional challenge provided it is microcontroller independent.

The move from a PC simulation to a platform independent virtual model can be seen in Figure 1. The challenge during FMU generation is removing dependencies on the compiler and linker toolchain as well as automatically generating platform independent C-Code for all required features and components provided by Synopsys Silver®. Developing the methodology initially required in depth compiler knowledge as well as a profound understanding on modern operating systems and CPU architectures. Once the tooling was developed subsequent updates of the target software or the simulation basic software require little effort.

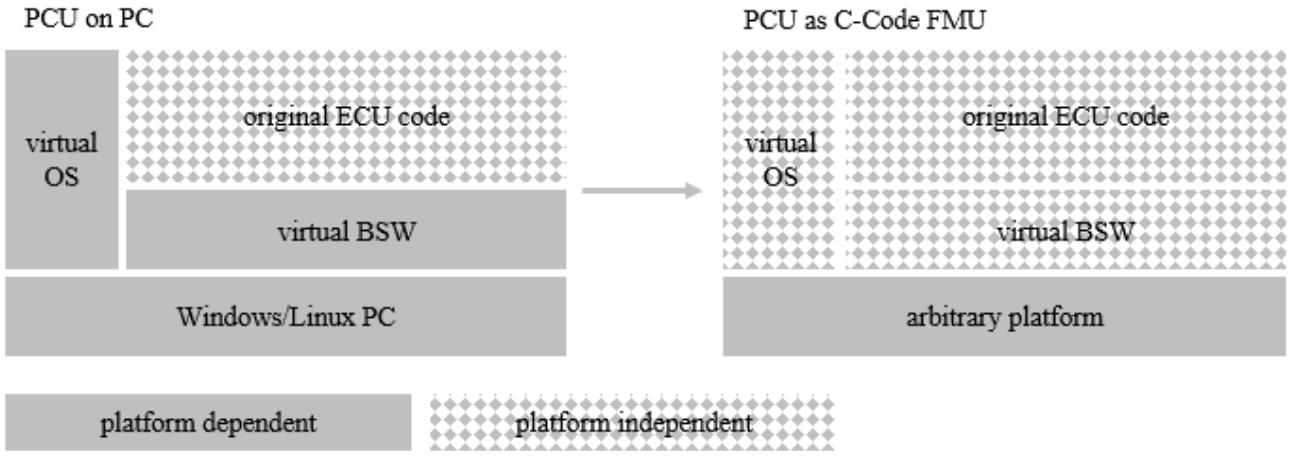


Figure 1: Porting a PC based simulation to be platform independent.

While the approach is conceptionally straight forward and the project was successfully integrated, the team gained key insights which influenced the initial favorability assumptions. The use of open C-Code can be problematic when integrating intellectual property from multiple parties. This issue was limited by keeping the number of involved parties small but must be kept in mind when applying this method to future projects.

There are mandatory virtual ECU features which cannot simply be provided as platform independent code. One example is the parametrization of ECU CHARACTERISTICs using production parameter and A2L files. Exporting such a feature as C-Code requires that certain assumptions like endianness or type sizes are known about the platform the FMU is later executed on. This is natural as this information is also required by industry measurement tools and is encoded in the ECU A2L. Removing such a feature to achieve true platform independence would drastically reduce the useability of the resulting FMU, so these assumptions were verified on the HIL platform and introduced into the export process. Doing so voids the platform independence of the FMU, as it is now compatible to the two specific PC and HIL platforms.

The use of pure source code to become compiler/linker independent shifts the build process to the HIL environment. To minimize the frequency of feedback loops related to the build step, the compiler toolchain of the HIL and the PC were aligned, as it is available for both platforms. Even then it is only possible to functionally verify the FMU built for PC without consuming HIL resources. A high-quality acceptance test on PC kept the number of functional errors and the time spent validating the FMU on the HIL system to a minimum. However, even validation using source level debugging on the PC system only assures the quality of the PC-built binary and due to the mandatory rebuild on the HIL it is possible that the different HIL platform introduces side effects.

In general, there are not many hard limitations. One is the computing power of the HIL system. In this case it was a multi core HIL x86 CPU and the FMU runs on a single HIL core without encountering task overruns. The second limitation is some required compatibility between the HIL and PC architecture, as true platform independence would have only been achieved with an unreasonable amount of engineering. When designing the complexity of such a virtual ECU export to the HIL it is advisable to evaluate level of detail versus the potential gain to avoid lengthy and costly iterations in the HIL environment.

5. Integration

In order to integrate the virtual PCU in the HIL model, the FMU is imported into the build environment. In our setup the build environment consists of two main components: 1. The tool Configuration Desk® that is used to configure the dSpace® HIL hardware, the task configuration and the signal routing between plant models or FMUs and the IO of the HIL and 2. The modeling tool Simulink® that is used to implement plant models as well as the RBS. Figure 2 shows the structure of the overall setup after integrating the virtual PCU. As the PCU only interacts through bus systems, only the parts relevant to the integration of bus systems is visualized in Figure 2. All other parts that a HIL system normally requires like power supplies and multi IO are omitted.

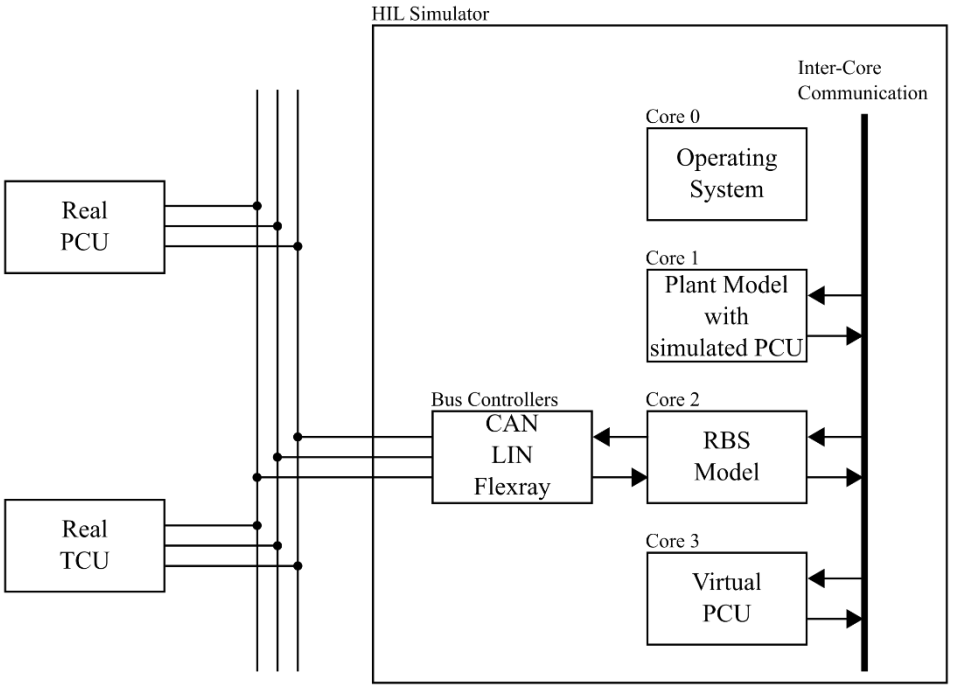


Figure 2: Integration of Real, Simulated and Virtual PCU in the HIL System with Focus on Bus Systems

Like the plant model as well as the RBS model and the operating system, the FMU is executed on a dedicated core of the HIL simulator's processor. The cores exchange signals through an inter-core communication mechanism. FMU do not allow calling portions of their code from different tasks. The virtual PCU integrated into the HIL setup handles task scheduling by using internal subtasks and is called by the HIL by an external 5 ms task. For applications with time critical tasks, a design that imports multiple FMU and calling them by separate tasks would be required. Note that the setup allows running either a simulated, a real or a virtual PCU with the same executable by configuring the model through the HIL automation. The TCU is always kept as a real component and is the DUT, as described above.

The signal routing for the three implementations real, virtual and simulated PCU integrated into the HIL model is fundamentally different and in the case of the simulated and the virtual PCU depends on the way the RBS is implemented. Figure 3 shows an example for the signal routing for all three PCUs in a minimal example. Only the different implementations of the PCU, the real TCU as well as an additional simulated ECU are displayed for the implementation of signal routing of CAN bus signals.

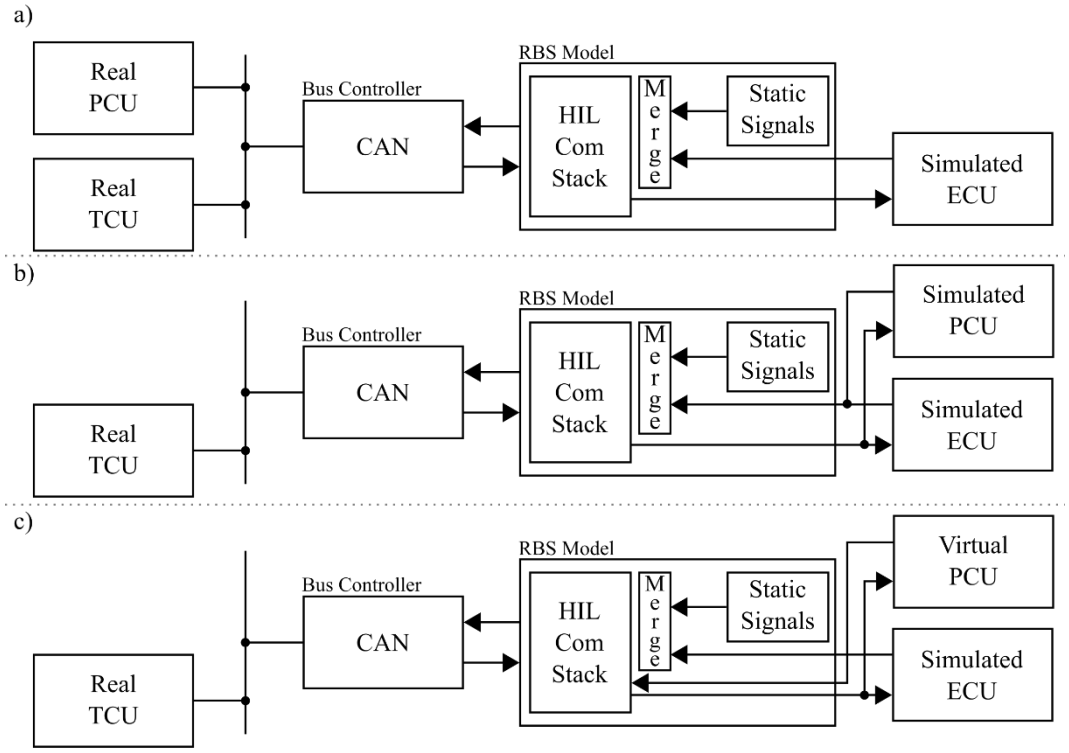


Figure 3: Signal Routing for a Setup with a) Real PCU, b) Simulated PCU and c) Virtual PCU

Figure 3 a) shows the case where the real PCU communicates with the real TCU and one additional simulated ECU. In this setup, three physical bus controllers are connected on the bus lines. The controller of the real TCU, the controller of the real PCU and the controller of the HIL RBS. In the case of the real TCU and the real PCU all signals are calculated in their respective application and are then transferred to the controller and vice versa. In the case of the simulated ECU, only a subset of the signals that require dynamic changes during the HIL simulation are calculated in the model of the simulated ECU. These signals are then merged with static default signals in the RBS and the resulting values are then transferred to the communication stack of the HIL and the HIL bus controller. The same is true for incoming signals from the bus. Only signals that are required for the computations in the simulated ECU are selected in the HIL communication stack for transfer to its model.

Figure 3 b) shows the case where the simulated PCU communicates with the real TCU and one additional simulated ECU. What was described above for the generalized simulated ECU is also true for the simulated PCU. Dynamic signals from the simulated PCU model are merged with static default signals, are processed by HIL communication stack and then sent on the bus via the bus controller. The switching between the setup from Figure 3 a) and the one from Figure 3 b) requires deactivating the power supply as well as activating the nodes for the simulated PCU in the HIL RBS. When modifying the physical setup of the bus, proper CAN termination must be ensured as well.

Figure 3 c) in turn shows the case where the virtual PCU communicates with the real TCU and one additional simulated ECU. Compared to the setup in Figure 3 b), all signals are computed by the virtual PCU and merging with static default signals is only required for the simulated ECU. As the virtual PCU runs on a separate core, signal routing to the Simulink® model containing the RBS is required. The RBS model implementation accepts signals of the type double and converts these internally to the required data type. As the virtual PCU provides all signals in the equivalent data type real, no additional handling of datatypes is required. While this simplifies the implementation, drawbacks regarding performance have to be accepted due to usage of unnecessarily expensive data types for some signals. The large amount of signals in a vehicle network requires an automated process to generate the signal routing parts of the model. The automated process requires naming of the virtual ECU input and output signals based on the description files of the bus systems. Due to differences between the implementations of RBS models for the bus types CAN, LIN and Flexray, both the signal routing as well as the merging with static default signals requires variants of the approach presented above.

6. Experiment and Results

The resulting HIL setup can execute tests with the newly integrated virtual PCU, the simulated PCU that allows manipulation but limited features and the real PCU. The DUT in all three scenarios is the TCU as mentioned above. To show the validity of our approach and to compare the three different implementations of the PCU, a test scenario is executed on the HIL. A common test scenario is to use a vehicle speed profile such as the worldwide harmonized light vehicles test procedure (WLTP) and to use a speed controller to set accelerator and brake pedals accordingly. This approach has the disadvantage that the speed controller can mask the differences in the behavior of the PCU on the vehicle speed level. To ensure that the differences in the behavior stay unmasked, a simple test scenario with a fixed sequence of inputs to accelerator and brake pedal is executed on the HIL. Figure 4 shows the stimulus as well as an example of the resulting rescaled dimensionless vehicle and engine speed.

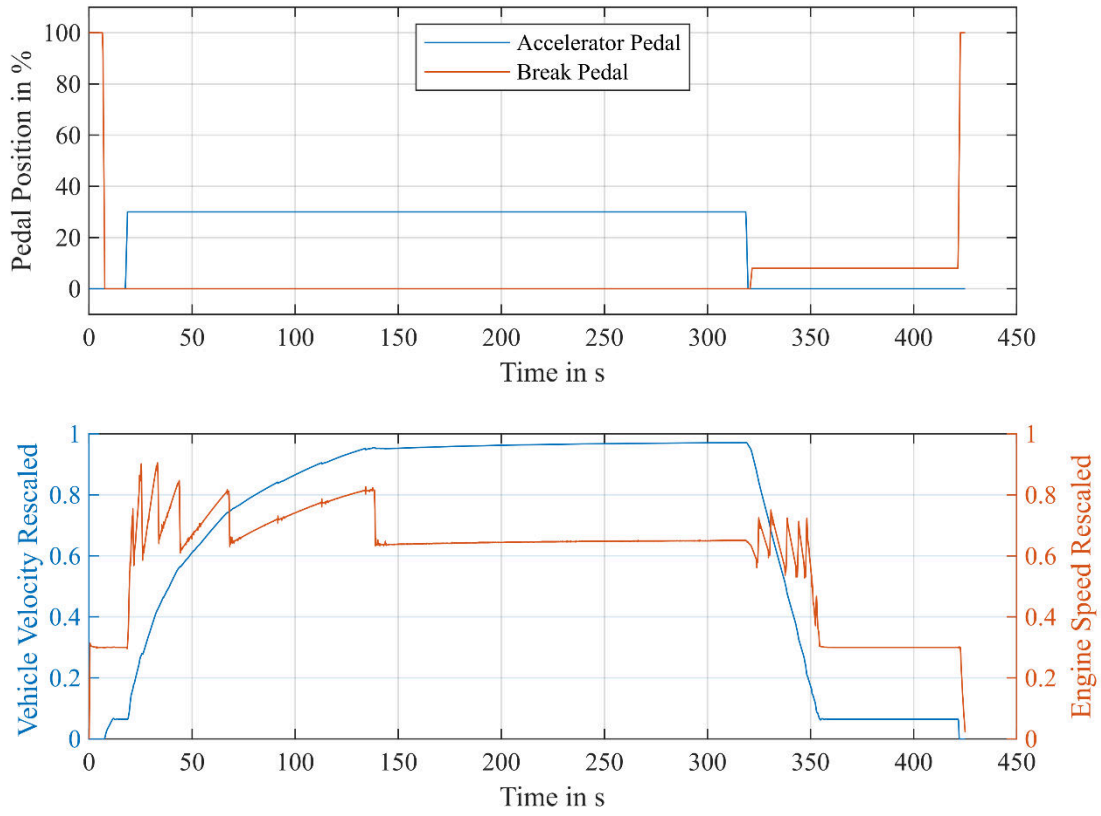


Figure 4: Stimulus and Reference Result for Comparison

At the beginning of the sequence, the combustion engine of the vehicle is started and the gear selector lever is set to drive. In the next step, the brake pedal is released and the vehicle starts to creep (t = 10 s). After reaching a steady creep velocity, the accelerator pedal is set to a constant value of 30 %, which leads to a rising vehicle velocity with moderate acceleration (t = 20 s). After a fixed wait time that is sufficient for the vehicle to reach a constant velocity, the accelerator pedal is released and the brake pedal is set to 5 % (t = 320 s), which causes the vehicle to reduce the velocity to creep velocity (t = 355 s). Finally, after another wait time, the vehicle comes to a stop by fully applying the brake pedal (t = 420 s). During the phases of accelerating and decelerating the vehicle, multiple up- and downshifts are visible in the engine speed profile. They stand out as steep decreases or increases of engine speed. The engine speed level at which a gearshift occurs is relevant for the overall powertrain strategy, as the shift levels affect the ability of the combustion engine to produce torque as well as its fuel consumption. The ECUs in the powertrain network negotiate the shift levels during runtime, which makes them ideal for a comparison of the performance of simulated and virtual PCU versus real PCU.

Figure 5 shows the rescaled dimensionless vehicle and engine speed for virtual, simulated and real PCU when executing the stimulus from above. The close matching of virtual and real PCU are visible, while the results of the run with the simulated PCU deviate. The reasons for these deviations are in the limited functionality of the simulated PCU as well as in a simplified variant dependent parameterization. The difference of the results with the simulated PCU become even more

apparent when comparing the shift points for upshifts while accelerating the vehicle. The reasons for the deviation in Figure 5 lie in the simplified representation of functions for torque coordination. This can be addressed by adding more detailed functions to the simulated PCU and parameterizing them as required for the specific variant. If this action is carried out until no deviations occur in any scenario, the effort for implementing the simulated PCU will match the effort for implementing the real PCU. The reuse of the code of the real PCU in a virtual PCU yields the same result with limited effort, which makes it much more attractive.

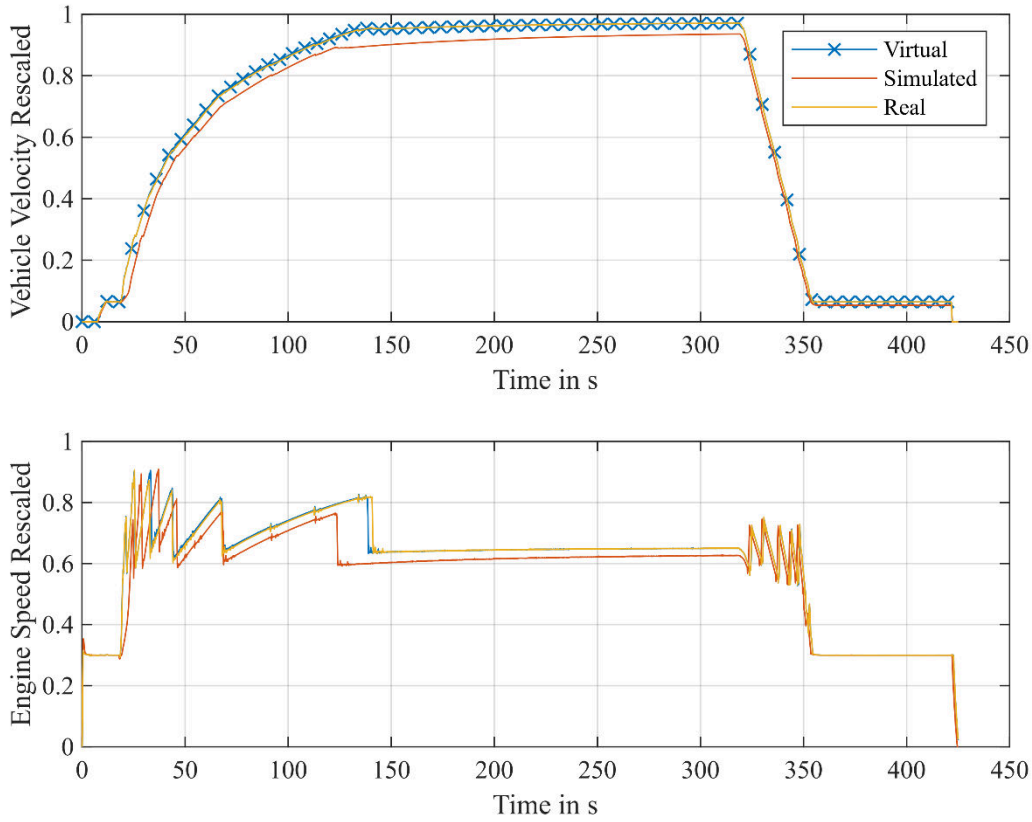


Figure 5: Results for Virtual, Simulated and Real ECU

For the further discussion, the results acquired with the simulated PCU are set aside and we focus on the deviations between the virtual and the real PCU. For both setups, three runs of the scenario shown in Figure 4 are carried out. Subsequently, the recorded vehicle speed is used to compare the fluctuations between runs for the same setup to assess the reproducibility, as well as the deviations between the setups to assess the accuracy of the representation of the real PCU by the virtual PCU. The scope of the scenario used covers the most important functionalities in a drive train and is useful for characterizing the accuracy of the representation. However, functionalities of the PCU software that are not active in this scenario cannot be assessed by this analysis.

Columns two and three of Table 2 show the root mean square errors (RMSE) acquired from the three runs with real and simulated PCU. The RMSE is calculated between first and second run, second and third run, as well as third and first run for each setup. The results show that the RMSE representing the variations of repeated runs with real PCU is higher than the RMSE for runs with virtual PCU. This is true both for all three calculated values of RMSEs as well as for the mean of the RMSEs shown in the last row of Table 2. On average, the RMSE of the vehicle speed calculated with real PCU with a value of 0.18 km/h is almost double the average RMSE of the one calculated with the virtual PCU with 0.1 km/h. This analysis itself only shows that the setup with real PCU varies more between repeated executions than the setup with virtual PCU. It does not give any indication of the accuracy of the representation of the real PCU by the virtual PCU.

Table 2: RMSE of Vehicle Speed for repeated Executions with Real and Virtual ECU

Run	real PCU RMS Vehicle Speed [km/h]	virtual PCU RMS Vehicle Speed [km/h]
1 vs. 2	0.22	0.09
2 vs. 3	0.18	0.09
3 vs. 1	0.13	0.11
mean	0.18	0.10

Reasons for the higher fluctuations between runs with real PCUs become apparent when comparing the nature of the two setups. In the setup with the real PCU, the code runs on a separate processor with its own clock. Drift and jitter between the clocks of the HIL and the PCU are one example for a source of deviation if they are not compensated for by special methods [10], which is not the case in our setup. In addition, the communication via bus systems like CAN is not deterministic and the dataflow between the real PCU and the TCU can vary between executions. The same is true for other I/O interfaces. For example, analog I/O with noise can introduce additional variance in the behavior. Most of the abovementioned effects are absent for the virtual ECU, as it runs on the HIL processor and does not possess any real I/O. The remaining sources of variance between the runs are the CAN communication with the TCU, the loopback through the communication controllers of the HIL, as described above, as well as the fluctuations introduced by the TCU. In summary, the setup with a virtual PCU shows a better reproducibility than the setup with a real PCU, which satisfies the minimum requirement that using a virtual PCU may not negatively affect the reproducibility.

Table 3 lists the RMSEs calculated between runs with real and virtual PCU and allows us to assess the accuracy of the representation of the real by the virtual PCU. The RMSEs in this assessment are in the same range as the RMSEs for repeated runs of the real PCU in Table 2. This shows that the error introduced by using a virtual PCU instead of a real PCU is below the level for repeated executions of the real PCU. In other words, the difference between one test execution with a real PCU and one test execution with a virtual PCU does not exceed the difference witnessed when executing a test twice with the real PCU. This means, that for this specific implementation and scenario, the virtual PCU suffices the reproducibility as well as accuracy expectations if the reproducibility with the real PCU is satisfactory.

Table 3: RMSE of Vehicle Speed for Comparison between Executions with Real and Virtual ECU

Run	real vs. virtual PCU RMS Vehicle Speed [km/h]
1	0.22
2	0.10
3	0.19
mean	0.17

The data used for the discussions in the last paragraphs is both limited in the number of datasets and in its variation. For a more detailed analysis, more executions with more complex scenarios are required. However, the limited analysis already shows that a high level of reproducibility, as well as accuracy can be achieved when replacing real ECUs with virtual ECUs.

7. Summary and Outlook

In this work, we were able to show, that virtual ECU can be reused to enhance dynamic inputs to a HIL residual bus simulation and to thus enable a more realistic representation of the DUT environment. The results obtained with this novel approach are not distinguishable from results obtained when using a real ECU instead. Regardless of the good results during runtime, some drawbacks remain during the implementation phase of the HIL executable. These include the large

number of signals that have to be routed to the RBS, the workarounds necessary in order to combine the virtual ECU signals with signals of other simulated or virtual ECUs, as well as the complex process to make the C-Code FMU buildable in the modelling environment of the HIL simulator.
While the first two items can only be addressed by completely automated signal routing and by redesigning the model structure, the latter requires a change in the process of the generation of the virtual ECU in a FMU container. In the most recent version 3.0 of FMU that will be released in 2022, additional features for specifying the process for compiling and linking of the sources will be available [11]. Furthermore, other new features like events for the triggering of sub functions can be useful for extending the approach to virtual ECUs that are more complex. With the current tool chain, only FMUs of the Version 2.0 are supported and the new features should be evaluated once full support for FMUs of the version 3.0 is available.
A different approach to address the build issues discussed above can be to work with precompiled object files, which need to be compatible to the x86-based platform and compiler of the HIL modeling environment. In addition to moving the compile process out of the complex HIL modeling environment, this change in the process also ensures protection of intellectual property. Particularly larger ECU software projects, where supplier and OEM software modules are combined to create the ECU software, can benefit from this.

Acknowledgments

This project was only possible with the continuous support of a large group of engineers. Many thanks go to Muralidhar Rajaram, who supported in commissioning the modified project on the HIL simulator and to Gyaneshwar Singh for his support with acquiring the data for offline analysis. For the modification of the network model and their creative approaches to the problems arising, we thank Ashutosh Singh and Felix Maier.

References

[1] A. Junghanns, R. Serway, T. Liebezeit and M. Bonin, "Building Virtual ECUs Quickly and Economically," *ATZelektronik worldwide,* pp. 48-51, 01 06 2012.

[2] C.-F. Nicolas, I. Ayestaran, T. Poggi, G. Sagardui and J.-M. Martin, "A CAN Restbus HiL Elevator Simulator Based on Code Reuse and Device Para-Virtualization," in *IEEE 20th International Symposium on Real-Time Distributed Computing (ISORC)*, Toronto, Canada, 2017.

[3] I. Matheis, T. Dörsam and W. Hoffmann, "Integrating a SiL into a HiL Test Platform," in *Simulation and Testing for Vehicle Technology, 7th Conference*, Berlin, 2016.

[4] A. Himmler, L. Stockmann, S. Walter and S. Laux, "Developments Targeting Hybrid Test Systems for HIL Testing," in *AIAA SciTech Forum*, Sandiego, California, USA, 2019.

[5] T. Blochwitz, M. Otter, M. Arnold, C. Bausch, C. Clauß, H. Elmqvist, A. Junghanns, J. Mauss, M. Monteiro, T. Neidhold, D. Neumerkel, H. Olsson, J.-V. Peetz and S. Wolf, "The Functional Mockup Interface for Tool independent Exchange of Simulation Models," in *Proceedings of the 8th International Modelica Conference*, Dresden, 2011.

[6] C. Bertsch, J. Neudorfer, E. Ahle, S. Arumugham, K. Ramachandran and A. Thuy, "FMI for Physical Models on Automotive Embedded Targets," in *11th International Modelica Conference*, Versailles, 2015.

[7] A. Himmler, A. Pillekeit, B. Loyer and V. D. Stéphane Mouvand, "Using FMI- and FPGA-Based Models for the Real-Time Simulation of Aircraft Systems," in *AIAA Modeling and Simulation Technologies Conference*, Kissimmee, Florida, USA, 2018.

[8] "Functional Mock-Up Interface for embedded systems (eFMI)," [Online]. Available: https://emphysis.github.io/pages/downloads/efmi_specification_1.0.0-alpha.4.html. [Accessed 02 09 2021].

[9] "Requirements for the Standardization of Virtual Electronic Control Units (V-ECUs)," prostep ivip association, Darmstadt, 2020.

[10] M. Akpınar, E. G. Schmidt and K. W. Schmidt, "Drift Correction for the Software-based Clock Synchronization on Controller Area Network," in *IEEE Symposium on Computers and Communications (ISCC)*, 2020.

[11] "Functional Mock-up Interface Specification, Version 848f13a," [Online]. Available: https://fmi-standard.org/docs/3.0-dev/. [Accessed 21 12 2021].

Investigation of Scheduling Algorithms for DAG Tasks through Simulations

Michael Schmid, Jürgen Mottok
Laboratory for Safe and Secure Systems - LaS^3
University of Applied Sciences Regensburg, Germany
Email: {michael3.schmid,juergen.mottok}@oth-regensburg.de

***Abstract*—In the real-time systems sector, various task models and corresponding tests exist to model and verify the schedulability of task sets on the system at hand. While those models and schedulability tests have intensively been studied from a theoretical point of view, it is hard to make use of them to compare the actual execution behavior of scheduling algorithms on a real system. In contrast to schedulability tests, simulators can help to investigate the performance of specific scheduling algorithms. One of the most generalized task models to describe parallel tasks is the Directed Acyclic Graph model that allows to represent tasks as a series of subtasks that depict the potentially parallel computations and precedence constraints that denote the order in which the subtasks are allowed to execute.**

In this paper, we investigate various scheduling algorithms for the Directed Acyclic Graph model. For that, we first recapitulate the examined scheduling algorithms in detail and point out relevant differences. Subsequently, we present the evaluation of different global and federated scheduling algorithms using fine-grained parallel tasks. To this end, we generate random Directed Acyclic Graph tasks and simulate their execution on multiprocessor systems using scheduling algorithms such as global rate-monotonic and semi-federated scheduling as well as global scheduling policies using the thread pool model.

***Index Terms*—real-time, scheduling, task models, simulation**

I. Introduction

As physical constraints have put an end to the ever increasing processor frequencies, hardware manufacturers have switched to parallel hardware architectures that allow to further increase the computation capabilities of the system. This trend has also reached the embedded real-time sector where multicore and multiprocessor architectures have found a lot of research attention in recent years. In order to make use of the parallel hardware architectures, sequential task models have cleared the way for parallel task models that allow to reduce the response times of time critical tasks. One of the first parallel task models analyzed in the real-time community is the fork-join model which is well-known from the Open Multi-Processing (OpenMP) standard. Tasks are hereby modeled as an alternating sequence of sequential and parallel segments. Only when all subtasks of one segment (either sequential or parallel) have completed their execution, the subsequent segment is allowed to execute. The parallel synchronous task model is a generalization of the fork-join model where each segment is allowed to have an arbitrary number of parallel subtasks but still, subtasks of one segment are only allowed to execute when all subtasks of the previous segment have finished. In this paper, we consider the Directed Acyclic Graph (DAG) task model. As the name suggests, the tasks are modeled as a DAG where each node represents a computation and the edges between the nodes depict the precedence constraints between the subtasks. In this model, nodes that are not either directly or transitively connected by edges can be executed in parallel.

Various scheduling algorithms exist that try to schedule the DAG tasks on the processors without deadline misses and the real-time community provides many schedulability tests to determine the feasibility of task sets beforehand. While the acceptance ratios provided by those tests can be compared to each other, they do not really assess the performance of the scheduling algorithms, but rather the performance of the tests. To this end, different performance metrics of scheduling algorithms have been introduced, such as utilization and resource or capacity augmentation bounds.

Another great tool to evaluate the actual runtime behavior of the scheduling algorithms are simulations. This paper focuses on the simulation of real-time scheduling algorithms. The algorithms evaluated later on include preemptive global fixed-priority (FP) and earliest deadline first (EDF) scheduling, both with and without using thread pools for the execution of the subtasks, as well as semi-federated (SFED) scheduling.

A. Motivation and Contribution

Modern embedded real-time systems have to process enormous amounts of data in in a timely manner. In order to allow modern applications[1] to finish their computations in time, parallel task models have been introduced to the embedded real-time sector. In order to determine the schedulability of the system beforehand, modern applications, where the computations are broken down into many small subtasks that are potentially executed in parallel, need to be modeled after the aforementioned task models.

In addition, the task models also provide the means to derive performance metrics, such as the capacity or resource augmentation bounds, that allow to asses the performance of different scheduling algorithms. While those bounds offer formal guarantees that cannot be obtained from simulations, other drawbacks arise from resorting to such performance

[1]The notations of applications and tasks are used interchangeably throughout this paper.

metrics. On the one hand, the tests suffer from pessimistic mathematical assumptions which leads to unfair comparisons. On the other hand, the performance metrics have to be derived individually for each scheduling algorithm. In contrast to that, once the scheduling simulator framework has been established, only the new scheduling algorithms and task models have to be implemented. Furthermore, the examination through simulations allows not only to draw estimations on the scheduler performance, but also examine the actual runtime behavior of the system at hand and therefore, gain insight about various performance characteristics, e.g. the response time distributions of the individual jobs, the number of thread preemptions and the number of processor migrations. For this reason, this paper presents the results of the simulations conducted on various scheduling algorithms for the DAG task model. As the trend in the real-time community goes towards highly parallel hardware and software architectures, e.g. for object detection algorithms, the evaluations will be conducted by the use of highly parallel tasks in our evaluations. The given results shall give insight about the design and implementation of modern real-time systems that employ fine-grained parallel tasks, i.e. tasks where the computations are broken down into a large number of subtasks that are potentially executed in parallel.

To the best of our knowledge, we are the first to consider the comparison of the scheduling algorithms proposed in the later section of this paper, especially on the thread pool model. For this reason, we add the following contributions to the current state-of-the-art:

- We perform scheduling simulations on classic as well as modern real-time scheduling algorithms for the DAG task model. The investigated algorithms include global deadline monotonic (DM) and EDF for both, the classic and the thread pool DAG model, and semi-federated scheduling.
- We evaluate the performance of the scheduling algorithms by investigating the feasibility of task sets, the number of preemptions occurring during the simulation process and the response times of the individual tasks.
- We discuss the use of the investigated scheduling algorithms and their results to gain insight about the use in actual real systems.

B. Structure

The remainder of this paper is structured as follows. Section II introduces related work considering schedulability tests for the DAG task model and its evaluation using simulations. Subsequently, the task model used throughout this paper is introduced in Section III. Section IV then explains the functionality of the scheduling algorithms. The simulation results are presented in Section V and are discussed in the following section. Finally, Section VII concludes our work.

II. Related Work

In recent years, much research has been conducted in the scope of real-time schedulability tests for the various parallel task models. Response time analyses for the fork-join model, which is known from the OpenMP programming model, cover for instance partitioned FP [1] or various task stretching approaches as in [2], [3] where the parallel segments are turned into sequential segments as much as possible. Schedulability tests regarding global scheduling are provided for the parallel synchronous task model which is a generalization of the fork-join model and therefore, also provides the possibility of analyzing the fork-join task sets. Maia et al. [4] as well as Chwa et al. [5] provide response time analyses in terms of global FP and EDF scheduling, respectively. Regarding the DAG task model, response time analyses have been proposed for global scheduling amongst others by Melani et al. [6], Fonseca et al. [7] and Parri et al. [8]. In addition, Bonifaci et al. [9] provides a schedulability test for global EDF which is later improved by Baruah [10]. The thread pool model has also been recently brought to the real-time sector by Casini et al. [11]. They assign each task a number of threads equal to the number of processors in the system and realize the precedence constraints in the DAG with blocking mechanisms. In our previous paper [12], we present a response time analysis for DAGs without a blocking subtask scheduler and the possibility of providing a limited number of threads for each task. This model will be used in the evaluations presented in this paper.

Another approach to scheduling DAG tasks is federated scheduling [13]. In federated scheduling, high density tasks execute exclusively on dedicated processors whereas low density tasks are executed together on the remaining processors. Especially, semi-federated scheduling [14] which we will present later in short show exceptional results in their schedulability analysis.

In terms of performance metrics, Li et al. [15] prove a capacity augmentation bound of $4 - \frac{2}{m}$ for global EDF and improve their results for large m in their subsequent paper [13]. In the latter paper, they provide an augmentation bound of ≈ 2.62 for global EDF, ≈ 3.73 for global rate monotonic (RM) and 2 for a federated scheduler using implicit deadline task sets. In terms of constrained deadlines, Sun et al. [16] prove a capacity augmentation bound of $\beta + 2\sqrt{\beta + 1}$ for global EDF where β is the largest ratio of task period to deadline in the task set. While the capacity augmentation bound can be used as a schedulability test, it yields fairly poor results as shown in [14] where the authors evaluate the resource augmentation bound of Li et al. [15] in their experiments. As for resource augmentation bounds, Saifullah derive a resource augmentation bound of 4 for their DAG task decomposition method for systems with global EDF scheduling. Bonifaci et al. [9] introduce a bound of $3 - \frac{1}{m}$ for global DM and both, Bonifaci et al. and Li et al. [13] prove a resource augmentation bound of $2 - \frac{1}{m}$ for global EDF for arbitrary deadline task sets.

We did not find much evaluations of the DAG task model using simulators in the current state-of-the-art. Qamhieh et al. [17], [18] present their scheduling simulator `YARTISS` to evaluate the performance of their task stretching algorithm in comparison with the classic scheduling of DAG tasks with

the global scheduling approaches of DM and EDF. In their papers, they observe that their DAG stretching algorithm yields a better acceptance ratio than the classical approach under global DM but worse on global EDF. In contrast to the work presented in this paper, they perform their evaluations on tasks with low numbers of subtasks (at most 12 subtasks per task throughout both of their papers).

III. Task Model

In this paper, we consider the scheduling of sporadic DAG tasks on a multiprocessor platform with m identical processors with uniform memory access. Without loss of generality, all time intervals are assumed to be multiples of the system clock and therefore, non-negative integers.

Each task τ_i in the task set Γ is represented by a three tuple $\tau_i = (G_i, T_i, D_i)$ where T_i denotes the minimum inter-arrival time, D_i represents the relative deadline and G_i is the DAG modeling the parallel computations of the task. The graph $G_i = (V_i, E_i)$ contains n_i subtasks $V_i = v_{i,1}, v_{i,2}, ..., v_{i,n_i}$ which are interconnected by the edges in $E_i = V_i \times V_i$. At least every T_i time units apart, task τ_i will release a job, i.e. a sequence of subtasks that are to be executed on the processors in their respective order. The job released at the time instant r_i has to complete its execution before its absolute deadline $d_i = r_i + D_i$. In this paper, deadlines can be either implicit which means that deadline equals the task period, i.e. $D_i = T_i$ or they can be constrained which denotes that the deadline is less than or equal to the period, i.e. $D_i \leq T_i$.

Each subtask $v_{i,j}$ is characterized by a worst-case execution time (WCET) $C_{i,j}$. The worst-case workload W_i of the task is equal to the sum of all its subtasks, i.e. $W_i = \sum_{\forall v_{i,j} \in V_i} C_{i,j}$, and represents the time it takes to complete the execution of the task using a single processor. Throughout the course of this paper, we will omit the node's subscript i if it does not cause confusion. Each directed edge $(v_a, v_b) \in E_i$ represents a dependency between the *predecessor* v_a and the *successor* v_b. A subtask is said to be *ready* if and only if all its predecessors have completed their execution and is only then allowed to be executed. A subtask is called a *source* if it has no predecessors while a subtask with no successors is called the *sink* of the DAG. Without loss of generality, we assume that each DAG has only one source and one sink. This can be achieved by adding a dummy source and dummy sink, each with a WCET equal to zero, to the DAG. An example of a DAG task can be observed in Figure 1 where the nodes are represented by the circles and the edges are shown as arrows. The number above the subtasks denote their WCET.

As commonly defined for DAGs, we use the notation of a path $\lambda = (v_1, ..., v_{n_i})$ to describe the sequence of nodes $v_j \in V_i$ such that v_1 is the source of G_i, v_{n_i} is the sink of G_i and $\forall v_j \in \lambda \backslash \{v_{n_i}\}, (v_j, v_{j+1}) \in E_i$. The length of a path λ is denoted as the sum of WCETs of all nodes v_j in λ, i.e. $len(\lambda) = \sum_{\forall v_j \in \lambda} C_{i,j}$. The critical path length L_i is defined as the path with the biggest length, i.e. $\max_{\forall \lambda \in G_i} \{len(\lambda)\}$. In the example DAG of Figure 1, the critical path is the path $\lambda = (v_{i,1}, v_{i,4}, v_{i,5}, v_{i,6}, v_{i,11})$ and the critical path length yields 13.

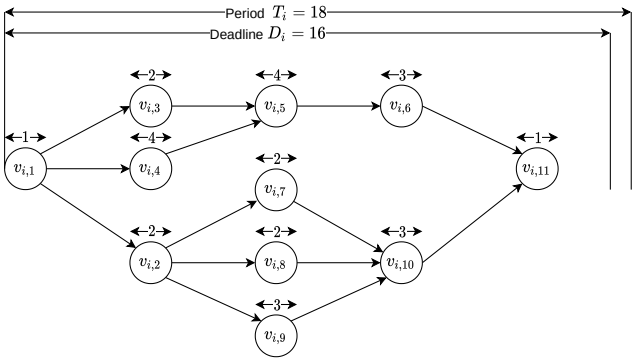


Fig. 1. Example of a DAG task with eleven subtasks.

IV. Investigated Scheduling Algorithms

This section shortly presents the scheduling algorithms evaluated in Section V so that subsequent deductions can be understood with greater ease.

A. Global Scheduling

Global scheduling is one of the most straightforward scheduling algorithms and is, especially in combination with a RM priority assignment, frequently implemented in actual embedded systems [19] due to their simplicity. In the sequential task model, the scheduler must simply maintain a single global priority ordered queue of all ready jobs.[2] At each scheduling instant, the available processors then pop the m highest priority jobs and execute them. In the DAG model, all subtasks inherit the priority of their respective task. Whenever they become ready because all their predecessors have completed execution, they are pushed into the global queue and the processors pop the m highest priority subtasks to execute them. In our evaluations, we will consider preemptive scheduling that allows to suspend running subtasks in order to schedule subtasks of higher priority tasks. Various priority assignment strategies exist. A simple method is to assign priorities statically in advance. A prominent example for this method is RM, where the priorities are assigned by the inverse of the task period, respectively, meaning that tasks with smaller periods, have higher priorities. In our experiments, we will use the DM priority assignment algorithm, which works as RM except that the relative deadline is used instead of the period to assign priorities.[3] Another example which is often considered in real-time literature is EDF which computes the priorities during runtime and assigns the job with the closest absolute deadline the highest priority. In the evaluations, we consider EDF and DM priority assignments which will be denoted as `GlobalEDF.Classic` and `GlobalDM.Classic`, respectively.

[2]In the sequential task model, the notation of subtasks is not used. Instead, a job represents a single sequential computation and executes for at most C_i units in time, where C_i represents the worst-case execution time of the task.

[3]Note that RM and DM are equal for implicit deadline tasks.

B. Global Scheduling with Thread Pools

In the thread pool model [12], each task τ_i receives a distinct thread pool with m_i threads that share the priority of the task. In contrast to classical global scheduling, the ready subtasks of a task are not pushed into the global queue. Instead, the threads, which inherit the priority of the tasks, are pushed into the global queue, while the subtasks are maintained in a private scheduling queue of each task. The system then follows the approach of a two-level scheduler: the m processors pop m threads from the global queue while the executed threads pop and execute subtasks from the task's private scheduling queue. It is possible that there are less than m_i subtasks ready which implies that some threads of τ_i will be without work. In practice, those threads are suspended and therefore not inserted in the global queue until new subtasks are pushed into the task queue. This functionality is important so that processors do not idle when they could instead process lower priority threads.

The system engineer assigns each task a certain number of threads. In this paper, we point out two similar methods. The first strategy follows the approach of federated scheduling and assigns each task the minimum number of threads that is necessary to meet its deadline:

$$m_i = \left\lceil \frac{W_i - L_i}{D_i - L_i} \right\rceil \tag{1}$$

The aforementioned thread assignment strategy ensures that all tasks have enough threads to complete their execution before their deadline when executed in isolation while also keeping the thread count in the system as low as possible, thus, reducing preemptions and context switching overhead. However, this method does not take into account the interference of higher priority tasks which might lead to deadline misses for the low priority tasks.

The second method improves the schedulability for FP systems by assigning the lower priority tasks more threads so that they can make better use of the parallelism provided by the system. In that case, each low priority task is assigned m threads to increase the maximum parallelism they can exploit.

$$m_i = \begin{cases} \lceil \frac{W_i - L_i}{D_i - L_i} \rceil, & \text{if } \sum_{\forall k \leq i} m_k \leq m \\ m, & \text{otherwise} \end{cases} \tag{2}$$

In this thread assignment strategy, we start with the highest priority task and assign it a minimum number of threads according to Equation 1. We continue to assign the minimum number of threads to each subsequent high priority task until the thread count in the system exceeds the number of processors m. At this point, all subsequent tasks will be assigned a number of threads equal to the number of processors in the system. This strategy leverages the same benefits as in the first strategy for the high priority tasks while also improving the schedulability of low priority tasks by allowing them to exploit the available parallelism capabilities.

In the evaluations, the respective plots will be denoted as `GlobalEDF.TP` and `GlobalDM.TP` when scheduling the threads according to global EDF and DM with the thread assignment strategy shown in Equation 1. The assignment strategy in Equation 2 will be referred to as `GlobalDM.ITP`.

C. Semi-Federated Scheduling

In semi-federated scheduling, tasks are divided into heavy ($\frac{W_i}{D_i} > 1$) and light ($\frac{W_i}{D_i} \leq 1$) tasks. Each heavy task is assigned n_i dedicated processors where n_i is computed by

$$n_i = \left\lfloor \frac{W_i - L_i}{D_i - L_i} \right\rfloor. \tag{3}$$

As n_i is less than the number of required processors to finish before its deadline, each heavy task requires an additional container task. This container task executes workload of subtasks for a fractional part of at most $\frac{W_i - L_i}{D_i - L_i} - n_i$. The container tasks are scheduled together with the light tasks on the remaining processors using any scheduling algorithm such as partitioned or global EDF. In this work, we use partitioned EDF which performed best in the original paper [14] and denote the algorithm as `SemiFederated.Partitioned` in our evaluations. The interested reader is referred to the original paper [14] for further information about the dispatching algorithm and how to compute the deadlines of the fractional parts computed on the container tasks.

V. Simulations

In our experiments, we evaluate the scheduling algorithms introduced in the previous section by generating random task sets and simulating the execution of a multiprocessor system by use of a discrete event simulator.

A. Experimental Setup

The task set generation in this work follows the same procedure used in various papers of real-time schedulability analysis, e.g. in [12], [14], [20], in order to allow for a comparison between the results of the schedulability tests and the simulations. First, DAGs are created using the Erdös-Rényi model $G(n, p)$ [21]: each DAG is assigned a random number of nodes chosen within the range $[50, 250]$. Each node receives a WCET randomly selected from the range $[50, 100]$. Then, for each pair of vertices an edge is created with a probability of $p = 0.1$. The value of p indicates the parallelism of the DAG where low values of p imply a low number of edges and therefore, a high intra-task parallelism. Given the structure of the DAG and the values of the WCET, the period of each task is computed following the approach presented by Jiang et al. [14]:

$$T_i = (L_i + \frac{W_i}{0.4 \times U_{tot}}) \times (1 + 0.25 \times Gamma(2, 1)) \tag{4}$$

where U_{tot} is the predefined total utilization of the system and $Gamma$ represents the gamma distribution. Using the previous method of generating a single DAG task, we generate tasks until the predefined value of the total utilization is exceeded. The period T_i of the last added task is then modified so that the desired system utilization U_{tot} is satisfied. For each

configured value of U_{tot}, we generate and simulate 100 task sets.

We do not evaluate typical parallel real-time applications in this paper as many different task sets are needed for the evaluations of the schedulability and number of preemptions. However, by using this method of generating DAG tasks, we obtain task sets containing various tasks with distinct graph structures and a sufficient level of parallelism even for low system utilizations. As an example, a fast Fourier transform with a parallel spawn depth of seven yields 190 subtasks and a maximum parallelism of 64. While the probability is low, such a DAG structure could potentially be created by our task generation procedure. In addition, comparable DAGs are randomly produced so that typical parallel applications are represented and a variety of different tasks are evaluated in the experiments. Using Equation 4, valid periods are computed while also generating a reasonable number of tasks for each utilization step.

B. Experimental Results

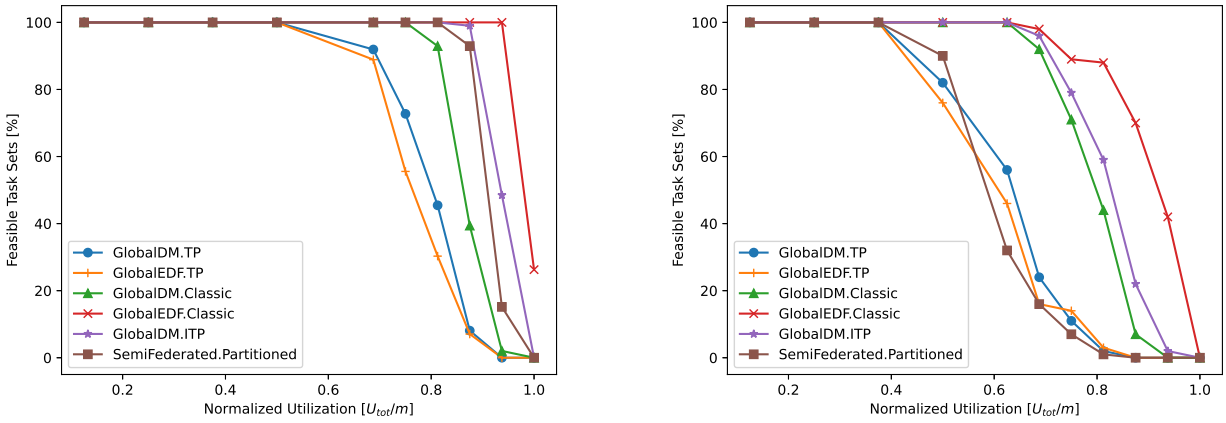
The first set of experiments considers the feasibility of the generated task sets when the system utilization varies between 1 and 8 on a multiprocessor system with $m = 8$. Figure 2 shows the results for implicit and constrained deadline task sets. In Figure 2(a), we can see that the thread pool model with a thread assignment according to Equation 1 is not able to achieve good feasibility results compared to the other approaches. This is caused by the low number of threads of low priority tasks which prolongs the computation time of the task and, in conjunction with the interference by other tasks, leads to missed deadlines already with low utilizations. Another significant difference can be observed when comparing global DM and EDF scheduling, where the latter shows great results and is able to correctly schedule few task sets having a system utilization of 100%. In contrast to that, semi-federated scheduling as well as the thread pool model with improved thread assignment perform almost equally, however, not as well as `GlobalEDF.Classic`. Looking at the constrained deadline results in Figure 2(b), we can easily observe the huge drop of success rate for the semi-federated scheduler. This can easily be explained by regarding the processor assignment function in Equation 3: For smaller deadlines, the number of processors n_i becomes larger, yielding an infeasible schedule when the tasks require more processors than available. Even though the thread assignment strategy proposed in this work uses a similar procedure, a feasible schedule might still be found due to the global scheduling mechanism, even if there are more threads than processors in the system. Apart from that, the results are as expected and all scheduling algorithms suffer more or less equally from the constrained deadline setup, leading to a earlier drop of the success rate.

Figure 3 shows the second set of experiments where the average number of preemptions per task set is shown, again for implicit and constrained deadline task sets. Both figures clearly show the downsides of global scheduling without the use of thread pools. Due to the high parallelism and the resulting processor contention, the number of preemptions is immensely high. We can also see that semi-federated scheduling also suffers from a lot of preemptions. These preemptions are essentially caused by the runtime scheduler of the container tasks which execute small portions of the subtasks on the shared processors. Furthermore, both figures show a drop of preemptions for semi-federated scheduling. These drops happen for two reasons: First, the tasks can no longer be correctly allocated to the processors. This leads to unfeasible task sets, which are also included in the graph even though no preemptions are accounted for in these cases. Second, for higher utilizations the number of light tasks ($\frac{W_i}{D_i} \leq 1$) that execute for rather long durations on the shared processors decreases. In addition, more fractional parts of heavy tasks are executed on the shared processors. In contrast to the light tasks, these fractional parts are rarely preempted. Another finding can be seen when comparing the implicit with the constrained results. In the case of the thread pool model, the number of preemptions rises, whereas the average number of preemptions in the classical global scheduling model remains equal. Again, this can be explained with the thread assignment strategies: For smaller deadlines, more threads are assigned to tasks according to Equations 1 and 2, and due to the higher thread count in the system, more preemptions occur in the execution process.

Finally, our last set of experiments shown in Figure 4 shows the response time distributions of a single task set. The box plots depicting the response times as a fraction of the task period can be interpreted as follows: The whiskers illustrate the 99th percentile and all outliers are depicted as `x`, the median is shown in orange and the values at 1.0 indicate the failed deadlines. For the global FP scheduling algorithms shown in Figures 4(a) - 4(b), we can immediately see the highest priority tasks which are never preempted and therefore suffer no interference at all. In the thread pool model, those tasks effectively execute on dedicated processors and therefore, the response time does not vary. In contrast, lower priority tasks have to contend for the free processors and thus, suffer from interference which is especially prevalent in the classical model shown in Figure 4(c) where already the second highest priority task has to make way for the highest priority task at some point in time.

In Figures 4(d) and 4(e), the graphs show wider response time distributions which we would expect from the dynamic priority assignments of EDF scheduling. Figure 4(e) reproduces the results of the schedulability evaluations and shows only very little deadline misses for all tasks even for a system utilization of 100%. When comparing DM with EDF in the thread pool model, the fixed-priority algorithm misses more deadlines (note that the number cannot be completely inferred from the figures) but the deadline misses are limited to the lowest priority tasks, whereas EDF scheduling misses most deadlines on tasks with high relative deadlines.

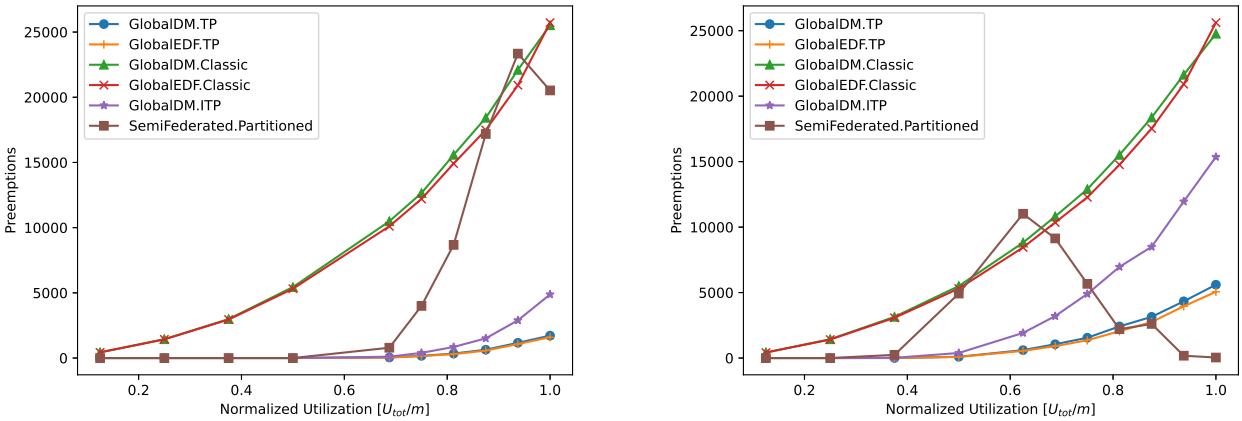
Finally, the plots of semi-federated scheduling shown in Figure 4(f) eminently illustrate the processor allocation: Five heavy tasks run mostly on allocated processors and thus,



(a) Implicit deadlines.

(b) Constrained deadlines.

Fig. 2. Feasibility of task sets with $m = 8$.



(a) Implicit deadlines.

(b) Constrained deadlines.

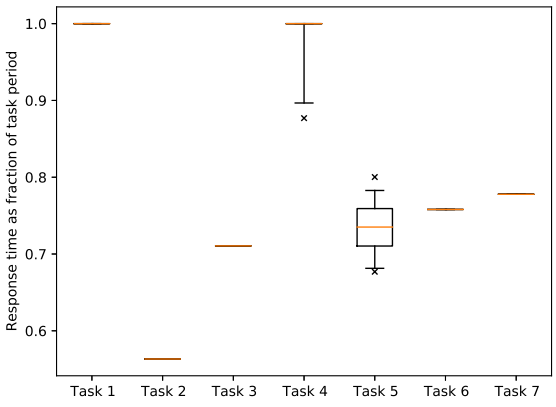
Fig. 3. Average number of preemptions per task set with $m = 8$.

execute without much interference. Those five tasks are therefore able to meet most of their deadlines, whereas the two light tasks are constantly preempted by the early deadline fractional parts of the heavy tasks and therefore, miss all of their deadlines. Note that the heavy tasks all have a very high ratio of response time to task period, e.g. 0.92 for `Task 6`. This results from the processor allocation mechanism in Equation 3, where each task is assigned as little processors as possible, thus, prolonging the response time as much as possible.

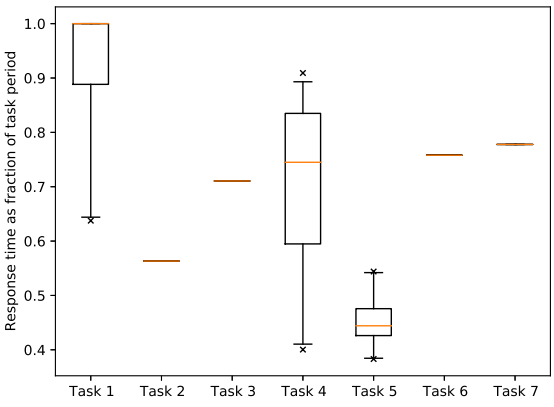
We did not include the response time distribution plots for constrained deadline task sets because they do not provide new relevant insight about the investigated scheduling algorithms.

VI. Discussion

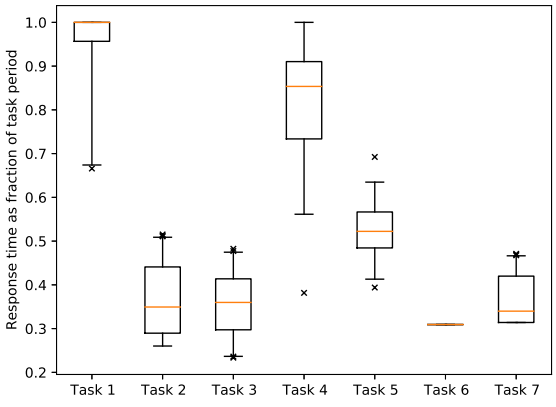
When it comes to the implementation details the classic global scheduling algorithm with fixed priorities should be the first choice due to its simplicity. However, different metrics, e.g. response times, criticality and runtime behavior amongst others, have to be considered as well when the feasibility of the system has to be guaranteed. While classical global EDF scheduling performs best in the evaluations of the acceptance ratio, other effects have to be taken into account when choosing a scheduler for real embedded systems. Our experiments have shown that global scheduling algorithms are suboptimal for fine-grained parallel applications due to their high thread counts which leads to a large number of preemptions and presumably to significant scheduling overhead when executed on real hardware. Also semi-federated scheduling, which to the best of our knowledge has one of the best performing schedulability analysis [12], [14] in the current state-of-the-art, suffers from huge amounts of preemptions on the shared processors that execute the fractional parts of the high density



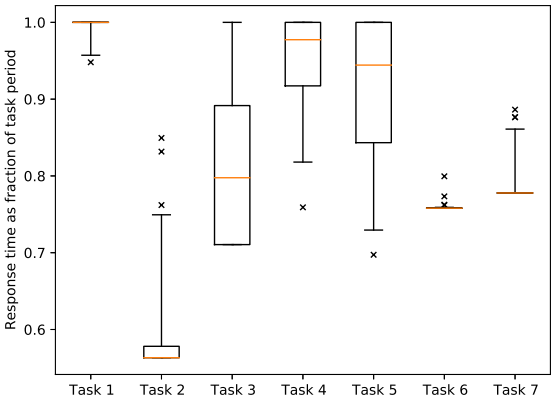
(a) GlobalDM.TP



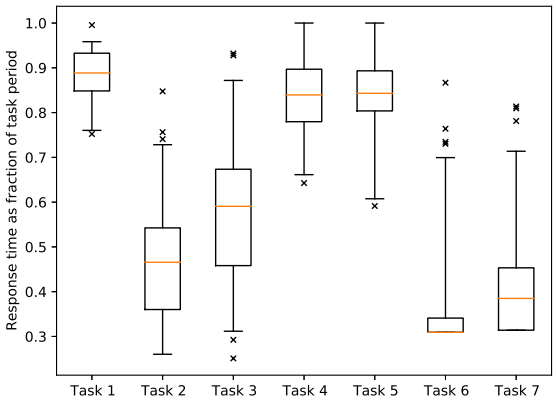
(b) GlobalDM.ITP



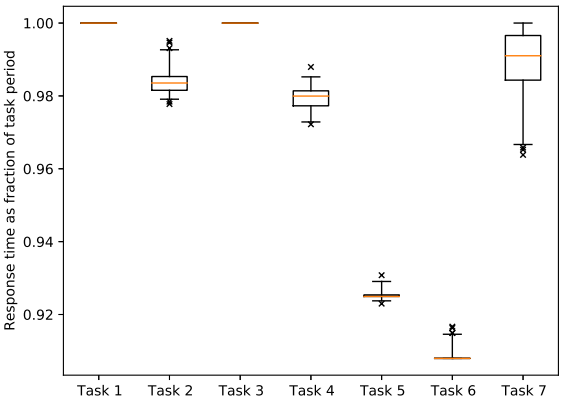
(c) GlobalDM.Classic



(d) GlobalEDF.TP



(e) GlobalEDF.Classic



(f) SemiFederated.Partitioned

Fig. 4. Evaluation of single task set with utilization $U_{tot} = 8$

tasks.

On this matter, the thread pool task model model offers the possibility of reducing the number of threads in the system and the experiments demonstrate the desired effect: The number of preemptions is drastically reduced compared to the classical variant, especially in the implicit deadline experiments. Another advantage of the thread pool model is that it works in conjunction with global schedulers. For this reason, it is possible to combine the classic DAG task model and the thread pool model in the same system: the classic task model is used for applications with low rates of parallelism while the task pool model is employed for large-scale parallel applications. However, better thread assignment strategies have to be derived in order to combine the classic and thread pool model in a system with global EDF scheduling.

While schedulability and preemptions are relevant metrics to consider, the criticality of the tasks has sometimes to be taken into account. Looking at the response time distributions of Figure 4, global EDF scheduling (on both, the classical and thread pool model) might not be a suitable candidate for this matter as priorities are computed dynamically which leads to widely varying response times and deadline misses might therefore occur on most tasks. Semi-federated scheduling favors heavy task whereas the light tasks suffer from lots of preemptions and as a result, miss a lot of deadlines. To this end, the fixed-priority assignment algorithms are the best choice when the consideration of the criticality of tasks is necessary.

A short summary of our findings is presented in Table I where the feasibility, number of preemptions and criticality are evaluated for the different investigated scheduling algorithms.

TABLE I
SUMMARY OF RESULTS (WORST: $--$ & BEST: $++$).

Algorithm	Feasibility	Preemptions	Criticality
Implicit deadlines			
GDM.TP	$--$	$++$	$++$
GDM.ITP	$+$	$+$	$++$
GDM.Classic	$-$	$--$	$++$
GEDF.TP	$--$	$++$	$--$
GEDF.Classic	$++$	$--$	$--$
SF.Partitioned	$+$	$-$	$-$
Constrained deadlines			
GDM.TP	$--$	$++$	$++$
GDM.ITP	$+$	$-$	$++$
GDM.Classic	$+$	$--$	$++$
GEDF.TP	$--$	$++$	$--$
GEDF.Classic	$++$	$--$	$--$
SF.Partitioned	$--$	$--$	$-$

VII. CONCLUSION

In this paper, we presented the evaluations of scheduling algorithms for the classic DAG task model using simulations. In the evaluations, we generated random task sets and measured their performance according to their feasibility, number of preemptions and response time distributions. We furthermore discussed our findings of the evaluations to help with future implementations of real-time systems that need to correctly schedule fine-grained parallel applications.

We showed that global EDF scheduling of the classical DAG task model yields great results in the feasibility analysis, however, may not be a suitable fit for real systems due to the large number of preemptions. For this reason, we suggest the use of the thread pool task model for large-scale parallel applications. In the thread pool task model, each task has a limited maximum parallelism adjustable by the number of threads assigned to each task individually. Both task models can be easily used concurrently in the same system which allows to exploit the benefits of both models.

ACKNOWLEDGMENTS

The research leading to these results has received funding from the Federal Ministry for Education and Research (BMBF) under Grant 01IS18047D in the context of the ITEA3 EU-Project PANORAMA as well as from the Center Digitization.Bavaria under Grant 16-1541.

REFERENCES

[1] P. Axer, S. Quinton, M. Neukirchner, R. Ernst, B. Döbel, and H. Härtig, "Response-Time Analysis of Parallel Fork-Join Workloads with Real-Time Constraints," in *25th Euromicro Conference on Real-Time Systems*, Jul 2013.

[2] K. Lakshmanan, S. Kato, and R. Rajkumar, "Scheduling Parallel Real-Time Tasks on Multi-core Processors," in *31st IEEE Real-Time Systems Symposium*, Nov 2010.

[3] F. Fauberteau, S. Midonnet, and M. Qamhieh, "Partitioned scheduling of parallel real-time tasks on multiprocessor systems," *ACM SIGBED Review*, vol. 8, no. 3, p. 28–31, Sep 2011.

[4] C. Maia, M. Bertogna, L. Nogueira, and L. M. Pinho, "Response-Time Analysis of Synchronous Parallel Tasks in Multiprocessor Systems," in *Proceedings of the 22nd International Conference on Real-Time Networks and Systems*, 2014.

[5] H. S. Chwa, J. Lee, K. Phan, A. Easwaran, and I. Shin, "Global EDF Schedulability Analysis for Synchronous Parallel Tasks on Multicore Platforms," in *25th Euromicro Conference on Real-Time Systems*, Jul 2013.

[6] A. Melani, M. Bertogna, V. Bonifaci, A. Marchetti-Spaccamela, and G. C. Buttazzo, "Response-Time Analysis of Conditional DAG Tasks in Multiprocessor Systems," in *27th Euromicro Conference on Real-Time Systems*, Jul 2015, p. 211–221.

[7] J. Fonseca, G. Nelissen, and V. Nélis, "Improved Response Time Analysis of Sporadic DAG Tasks for Global FP Scheduling," in *Proceedings of the 25th International Conference on Real-Time Networks and Systems*, 2017.

[8] A. Parri, A. Biondi, and M. Marinoni, "Response time analysis for G-EDF and G-DM scheduling of sporadic DAG-tasks with arbitrary deadline," in *Proceedings of the 23rd International Conference on Real Time and Networks Systems*, 2015.

[9] V. Bonifaci, A. Marchetti-Spaccamela, S. Stiller, and A. Wiese, "Feasibility analysis in the sporadic dag task model," in *2013 25th Euromicro Conference on Real-Time Systems*, Jul 2013, p. 225–233.

[10] S. Baruah, "Improved multiprocessor global schedulability analysis of sporadic dag task systems," in *2014 26th Euromicro Conference on Real-Time Systems*, Jul 2014, p. 97–105.

[11] D. Casini, A. Biondi, and G. Buttazzo, "Analyzing Parallel Real-Time Tasks Implemented with Thread Pools," in *Proceedings of the 56th Annual Design Automation Conference*, Jun 2019.

[12] M. Schmid and J. Mottok, "Response Time Analysis of Parallel Real-Time DAG Tasks Scheduled by Thread Pools," in *Proceedings of the 29th International Conference on Real-Time Networks and Systems*. Association for Computing Machinery, 2021.

[13] J. Li, J. J. Chen, K. Agrawal, C. Lu, C. Gill, and A. Saifullah, "Analysis of Federated and Global Scheduling for Parallel Real-Time Tasks," in *2014 26th Euromicro Conference on Real-Time Systems*. IEEE, Jul 2014.

[14] X. Jiang, N. Guan, X. Long, and W. Yi, "Semi-Federated Scheduling of Parallel Real-Time Tasks on Multiprocessors," in *2017 IEEE Real-Time Systems Symposium*, Dec 2017.

[15] J. Li, K. Agrawal, C. Lu, and C. Gill, "Analysis of global edf for parallel tasks," in *2013 25th Euromicro Conference on Real-Time Systems*, Jul 2013, p. 3–13.

[16] J. Sun, N. Guan, X. Jiang, S. Chang, Z. Guo, Q. Deng, and W. Yi, "A capacity augmentation bound for real-time constrained-deadline parallel tasks under gedf," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, p. 2200–2211, Nov 2018.

[17] M. Qamhieh and S. Midonnet, "An experimental analysis of dag scheduling methods in hard real-time multiprocessor systems," in *Proceedings of the 2014 Conference on Research in Adaptive and Convergent Systems*, ser. RACS '14. Association for Computing Machinery, Oct 2014, p. 284–290.

[18] ——, "Simulation-based evaluations of dag scheduling in hard real-time multiprocessor systems," *ACM SIGAPP Applied Computing Review*, vol. 14, no. 4, p. 27–39, Jan 2015.

[19] E. Bini and G. C. Buttazzo, "Measuring the performance of schedulability tests," *Real-Time Systems*, vol. 30, no. 1–2, p. 129–154, May 2005.

[20] A. Saifullah, D. Ferry, J. Li, K. Agrawal, C. Lu, and C. Gill, "Parallel real-time scheduling of dags," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 12, p. 3242–3252, Dec 2014.

[21] P. Erdös and A. Renyi, "On Random Graphs I." 1959.

SytHIL: A System Level Hardware-in-the-Loop Framework for FPGA, SystemC and QEMU-based Virtual Platforms

A RISC-V example using AWS cloud servers

Lukas Jünger *, Tobias Röhmel †, Mark Burton †, and Rainer Leupers *,
*RWTH Aachen University, {juenger, leupers}@ice.rwth-aachen.de
†GreenSocs SAS, {tobias.roehmel, mark.burton}@greensocs.com

***Index Terms*—ESL, SystemC TLM, Virtual Platforms, FPGA, Simulation, Emulation, QEMU, Hardware-in-the-loop, HIL**

***Abstract*—Using FPGAs to accelerate Virtual Platforms (VPs) has bent the rule within simulation community that you cannot have both accuracy and speed. Having the ability to run a VP at considerable speed, and being sure that the critical IP in question is being emulated totally accurately opens up the scope of functional verification to include the full software stack. Only hitherto this has typically been limited to very specialized and expensive emulators that are often difficult to scale. Being able to make use of this technology in a fashion conducive to continuous integration and test would be a game changer in many embedded systems groups.**

Recently this has become more possible, due to several developments: First, Amazon has made available their FPGA-enabled cloud instances. They are not the only ones to go this route, and using FPGA cards within an in-house server farm is also becoming feasible. Second, the technology needed to make use of these devices within the context of standard simulation frameworks is now better understood.

In this work an activity in this domain is presented: the SytHIL framework. Using SytHIL RTL descriptions emulated on FPGAs, SystemC TLM models and QEMU models can be combined into one unified simulation. This allows to accurately emulate hardware IPs such as processors or accelerators on the FPGA while other system components are executed at rapid speed as simulation models. Through the SystemC layer, the SytHIL simulation can also be connected to the physical world or other simulated environments. In our case study, we demonstrate the capabilities of SytHIL by emulating different RISC-V processors on Amazon FPGA-enabled cloud instances in combination with SystemC TLM and QEMU models via which the simulation is connected to the host network and thus to the internet.

I. Introduction

Over the past decades, the complexity of HW/SW systems has been steadily increasing. Also, embedded systems have become ubiquitous in our every day life in many application. To verify the correct functionality of everything from planes to disc-drives early in the design cycle, full system simulators, so called Virtual Platforms (VPs), are the de facto standard tool.

While VPs are mainly used for functional software verification, the Register Transfer Level (RTL) description of the components is usually verified using either RTL simulation within complex test benches, Field Programmable Gate Array (FPGA) emulation using special hardware or FPGA prototyping. The advantage of RTL simulation is that it is, by construction, entirely faithful to the design, not only in functionality but in measured performance. However, typical RTL simulators are many orders of magnitude slower than is required for typical software development or reasonable functional verification. Emulation using special hardware or FPGA prototyping bridges this gap by enabling the RTL description to be executed at near real-time speeds. While this seems to be an ideal solution, there are two main problems: The availability of the RTL description itself early in the design cycle, and the availability of the FPGA hardware on which to execute the RTL. To this end, standard VP modeling techniques are typically deployed using languages such as SystemC to implement a model of the proposed design, sufficient for the needs of functional verification. But again, this solution is not without its difficulties. Models are often complex to construct, or may not run as fast as the functional verification engineers would like. Hence, often a hybrid approach is preferred, re-using elements of available RTL designs while also using some standard simulation models of components. The degree of flexibility with which different model components can be used in such an environment is critical, but equally important is the availability of the FPGA hardware that will be used.

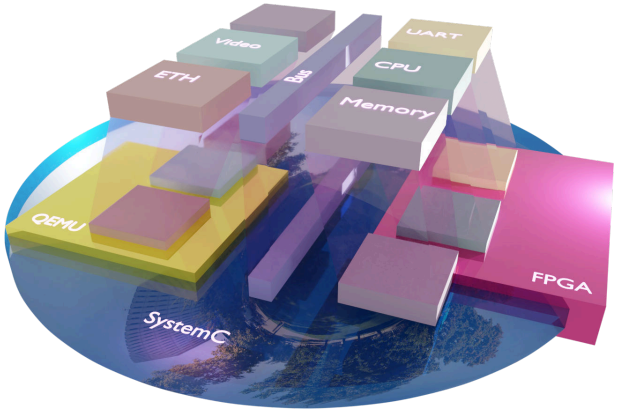


Fig. 1. Overview of the SytHIL framework.

This work aims to unite the software and hardware testing strategies into a hybrid approach. It uses the IEEE standard language for VPs: SystemC TLM 2.0, the ubiquitous open-source hypervisor/emulator QEMU [17], and FPGAs. Specifically, and critically, the FPGAs used are available in the cloud using the Amazon Web Services (AWS). This means

Paper	Goal	Emulation	Simulation type	Synchronization	Flow	SCE-MI
SytHIL	Func. hardware & software verification	FPGA	SytemC	Transaction only	both	No
[1]	Hardware verification	Emulator	C-Program	Clock control	sw→hw	No
[2]	Func. co-verification	FPGA	C-Program / ISS	Transaction only	sw→hw	No
[3]	Func. hardware & software verification	FPGA	SytemC	Cycle accurate	hw→sw	No
[4]	Hardware verification	FPGA	C-Program	Transaction only	sw→hw	No
[5]	Func. hardware verification	FPGA	C-Program / ISS	Transaction only	sw→hw	Yes
[6]	Mixed-signal hardware verification	FPGA	Any HLA	Clock control	both	No
[7]	Hardware verification	HDL simulation	C++ Program	Transaction only	sw→hw	No
[8]	Hardware verification	FPGA	C-Program	Transaction only	sw→hw	No
[9]	Hardware verification	FPGA	SytemC	Clock control	sw→hw	Yes
[10]	Algorithm verification co-design	FPGA	C model/ISS	Transaction only	sw→hw	No
[11]	Co-design	HDL simulation	ISS	Clock control	both	No
[12]	Hardware & Software verification	FPGA	C/C++ code	Clock control	sw→hw	No
[13]	Functional verification	phy. hardware	C-Code	Transaction only	sw→hw	No
[14]	Simulation speed up	FPGA	Simple Scalar	Transaction only	sw→hw	No
[15]	Hardware verification	FPGA	C-Program	Transaction only	sw→hw	No
[16]	Hardware & Software verification	FPGA	C++ Program	Clock control	hw→sw	No

Fig. 2. SytHIL comparison with related work.

that the methodology can be deployed at low cost, and scaled to cover the needs of large teams without the need to buy dedicated hardware. The integrated methodology that has been developed in this work, which allows model components from each of these three environments to work together, will be referred to as the *System Level Hardware-in-the-Loop (SytHIL)* framework.

The diagram in Fig. I aims to give an overview of the technology. At the top, a VP is shown consisting of different components, such as a CPU, a video card, an Ethernet MAC, memory and an UART. With SytHIL this VP can be partitioned into parts for execution inside the QEMU domain, the SystemC domain or the FPGA domain. The components are mapped onto these supports. Also, both the SystemC and FPGA environment themselves are hosted within SystemC. The FPGA environment is represented at the edge of the SystemC environment, as there are certain aspects of the FPGA which are not handled within SystemC, and physically the FPGA allows to interface to the real world. The diagram also shows a memory element that is shared between SystemC and the FPGA domain, the extent to which this is implemented will be examined further in this paper, as it is a critical feature which improves simulation speed. Finally, SystemC provides a view of the world to the simulation environment as SystemC is in control of the notion of time, and inputs/outputs to and from the VP. The purpose of the SytHIL framework is to enable all of this partitioning to be carried out as simply as possible, while all the necessary transactors and adapters are deployed under the hood within the framework.

II. Related Work

Functional hardware-software-co-verification was first proposed in 1998 in [13]. In that paper the emphasis is put on running software in the physical target environment. To do that, a functional software simulator is augmented with an FPGA that can interact with physical devices. The simulator can run target software and the interactions with the outside world are handled by the FPGA by toggling the respective pins. The exact mechanism that allows the transfer of data from the software simulation to the hardware simulation is the core technology in all of the hardware-software-co-verification approaches and will be categorized in the following. Shortly after the first paper, [7] introduced a transaction based scheme which increases abstraction by only keeping track of logical transaction information, e.g. address and data to be transferred. Compared to the older approach, where every logical level of the physical bus had to be accounted for, the new approach can increase throughput significantly. [7] also shifted the focus from functional verification to hardware verification. This means that the C simulation was used to write an abstract test bench that would have previously been written in an Hardware Description Language (HDL). This allowed greater productivity for the engineers and re-usability was increased since the test bench could be used for both a software and an RTL model of the design. Since then most research effort has been focused on the transaction based approach because of the increased simulation performance. Several papers improved the general mechanism in various ways like improved abstraction [1], addition of mixed-signal simulations [6] or exploration of the communication architecture [3] [9]. All these approaches rely on clock cycle accurate synchronization between the simulation and the FPGA which restricts the overall simulation performance.

A popular approach to increase simulation performance is to relax the timing synchronization between the software and the hardware simulation. A simple way to implement this idea is to omit explicit alignment of timing information and only rely on stalling while one part of the simulation depends on the other. To illustrate this concept, imagine a VP with an Instruction Set Simulator (ISS) executing target software and an FPGA attached in some way. Assume the FPGA contains the RTL description of a peripheral that will at some point be accessed by the target software. From the point of view of the target software it will just access a memory mapped register. At this point the software simulation will be stalled until the interaction with the FPGA is completed. The ISS will get the result of the memory access as if no time had passed.

The gain in simulation performance lies in the fact that while the FPGA is calculating the result of the memory access it is not synchronized with the software simulation. This approach lets both simulation run at full speed and only stall when they interact. This approach has been utilized often in literature ([2], [4], [5], [7], [8], [10], [14], [15]).

Another important distinction in this space is whether or not the design can handle transactions that are initiated by the hardware simulation. For now only the case where data bus masters or bus initiators are in the software simulation was considered. This is a considerable disadvantage if peripherals that have a Direct Memory Access (DMA) port or even whole CPUs are to be verified. The concept of having a device on the FPGA that can write data back through a DMA port has been explored in [12] and [11]. [12] describes a mechanism that allows for data to be transferred in both directions by synchronizing hardware and software on every clock cycle. While this approach allows the desired DMA transfer, it suffers from low simulation performance because of the synchronization overhead. In [6] and [3] an entire CPU was instantiated on the FPGA to verify the interplay between target software and RTL design. [3] ran target software for a NIOS 2 softcore in a custom scheduler environment and connected SystemC to simulate other peripherals. Cycle accurate synchronization is taken care of by the scheduler that runs underneath the target software, which ensures accuracy but incurs a performance penalty as mentioned previously. The authors of [6] rely on the HLA/RTI standard to connect all simulations which implement it. The FPGA is connected by controlling the Design Under Test (DUT) with a debugging subsystem and a software layer that implements the required HLA/RTI interfaces. This approach is also cycle accurate. A different angle is approached by [16], which puts emphasis on CPU design and architecture exploration. In their paper the RTL description of the CPU is transformed in a way that allows fine grained control of the hardware simulation while every module's clock is decoupled. They also provide software adapters that transfer the logic level of peripheral outputs to their software simulation. There is also the Accelera Standard Co-Emulation Modeling Interface (SCE-MI) which standardizes a modeling interface that is supposed to enable transactor models to be easily migrated from simulation to emulation [18]. Few of the mentioned approaches implement this standard.

With SytHIL we propose a framework that allows the connection of all types of peripherals and CPUs in whichever domain, hardware or software simulation, in a transaction based and decoupled manner. Fig. 2 provides a table which gives a comparison of our work and previous publications.

III. The SytHIL Framework

Our framework consists of two major parts, first the QEMU integration into SystemC and second the SystemC-FPGA integration. The QEMU domain is handled using GreenSocs's Qbox [19]. This allows wrapping QEMU CPUs into SystemC modules that have standard Transaction Level Modeling (TLM) ports that connect to the rest of the system. It also allows non-CPU QEMU devices to be wrapped and similarly exposed as SystemC modules so that it is possible to reuse all the devices that QEMU provides. For the FPGA-SystemC integration Xilinx's libsystemctlm-soc library was extended [20]. There are two types of connections, one to transfer SystemC initiator requests to the FPGA and the other to transfer RTL initiator requests from the FPGA to SystemC. The main difference between the two is in which domain the transaction initiator lies. The basic structure of both connections is similar, there is an RTL part, which is called the RTL bridge, and a software module interacting with it. The RTL bridge is connected to the user RTL design on the FPGA and the SystemC module, that acts as a user space driver for the RTL bridge, provides a TLM interface to it. In principle, the RTL bridge acts as a mailbox that the initiator side can write into and the target side will read out of, once it is ready or as soon as all relevant information has arrived. The RTL bridge also supports the forwarding of interrupts in both directions.

Xilinx's library supports the RTL design to be simulated in software with Verilator or to run it on an FPGA that is connected via Peripheral Component Interconnect Express (PCIe) port. To also support different bus protocols in the RTL design, such as AXI, ACE, and CHI, the software has to account for protocol-specific parameters. The back-end and the protocol-specific configuration are encapsulated in different modules to increase abstraction. The same back-end is used in both connection directions. An overview of the general architecture can be seen in Fig. 3. Notice that the AWS shell is specific to the AWS setup and facilitates the conversion from PCIe to AXI. In non-AWS setups this block can be substituted by Xilinx's PCIe to AXI converter which does not pose any issues since no other features of the AWS shell are being used.

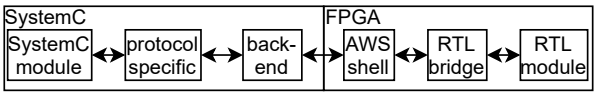


Fig. 3. FPGA to SystemC transaction state machine.

As an example the following paragraph will explain the sequence of a transaction from a functional point of view. To simplify the explanation details of the back-end interactions are neglected. The steps involved in a SystemC to FPGA transaction are depicted in Fig. 4. Generally, the transaction starts with a SystemC initiator that calls the b_transport function. The meta information of the transaction is programmed into the memory mapped registers of the RTL bridge on the FPGA. Once all information arrives, the RTL bridge carries out the AXI transaction according to the configuration and makes the result available in its registers. The software counterpart reads the results, converts them to a TLM payload, and returns them to the TLM initiator.

The FPGA to SystemC transaction works very similarly but is initiated on the FPGA. When a transaction arrives at the RTL

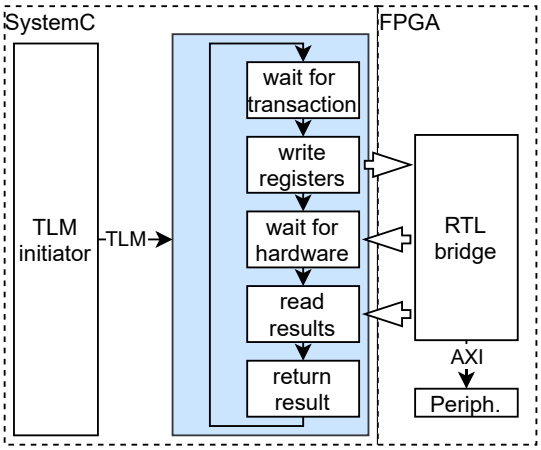


Fig. 4. SystemC to FPGA transaction state machine.

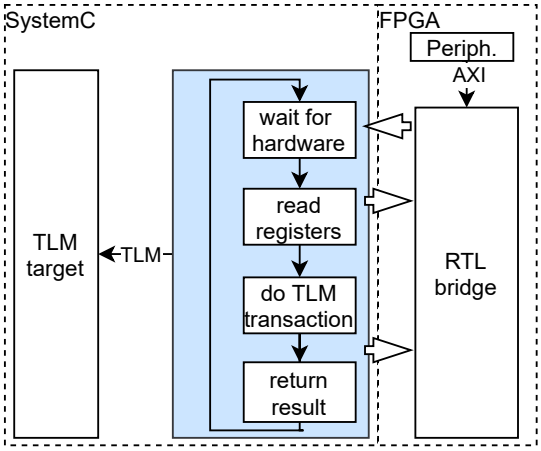


Fig. 5. FPGA to SystemC transaction state machine.

bridge, it notifies the SystemC module. This modules reads the transaction information from the RTL bridge registers and carries out the respective TLM transaction. The results are again written to the RTL bridge registers. This is shown in Fig. 5.

To increase the throughput of the SystemC to FPGA connection a mechanism that can optionally bypass all transaction related modules was implemented. The idea is to give the initiator module direct access to the FPGA through a memory pointer that is mapped to the FPGA. All accesses to that pointer result in AXI accesses on the FPGA. This pointer is initialized by the back-end by interacting with an AWS specific library and is then passed on to the initiator by using SystemC's Direct Memory Interface (DMI) mechanism. On the FPGA, Xilinx AXI interconnects are used to create two paths to the RTL peripheral. The first goes through the RTL bridge as before, while the second one bypasses the RTL bridge and directly connects to the target RTL module. A graphical representation of this setup is shown in Fig. 6. Since

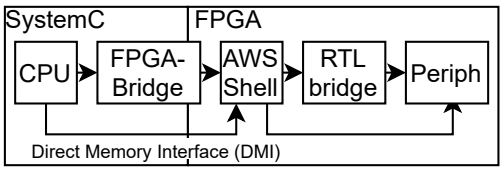


Fig. 6. AWS DMI setup.

the RTL bridges are still in the memory space the RTL target peripheral needs to be mapped in a special address range. The driving software expects the RTL bridges to occupy the first addresses up to 0x20_0000 which means the peripheral address needs to be bigger than that.

IV. Results

In this section, an overview of the experimental evaluation results is presented. An AWS f1.2xlarge instance was used as the simulation host for all benchmarks. An overview of its technical specification is provided in Table I.

First the speed of data movement between the FPGA and the SystemC framework was evaluated using different methodologies to identify the performance bottlenecks and boundaries. In the first experiment, a VP consisting of a RISC-V QEMU instance, a SystemC UART model and a memory is partitioned using our SytHIL framework. QEMU and the SystemC UART model are executed on the simulation host, while the memory is placed on the FPGA and connected using SytHIL transactors. Linux is booted on the RISC-V core running on QEMU. A benchmark program is executed in the RISC-V Linux that reads and writes data to the memory located in the FPGA.

Transfer throughput is measured against wall-clock time. A second partitioning in which the memory is placed in the SystemC domain instead of the FPGA is used for comparison. Two different methodologies for the connection between the host and the FPGA were evaluated. In the first, which is the standard approach used by e.g. [20], data is transferred via a system of letterboxes and interrupts using SystemC's blocking transport interface (b_transport). Note that both read and write operations require signaling from SystemC to the FPGA and back. Even in the case of a write transaction, the return path is used to indicate the completion and success or failure of the operation. In the second approach, which has been implemented in SytHIL, a bridge between the PCIe interface and the AXI internal bus fabric on the FPGA is used. This is implemented using the previously described DMI mechanism,

TABLE I
F1.2XLARGE INSTANCE SPECIFICATION.

CPU	Intel Xeon E5-2686 v4 (8 vCPUs)
RAM	122 GB
OS	CentOS 7
FPGA	1 Xilinx Virtex UltraScale+ VU9P
Price	1.65 USD/h

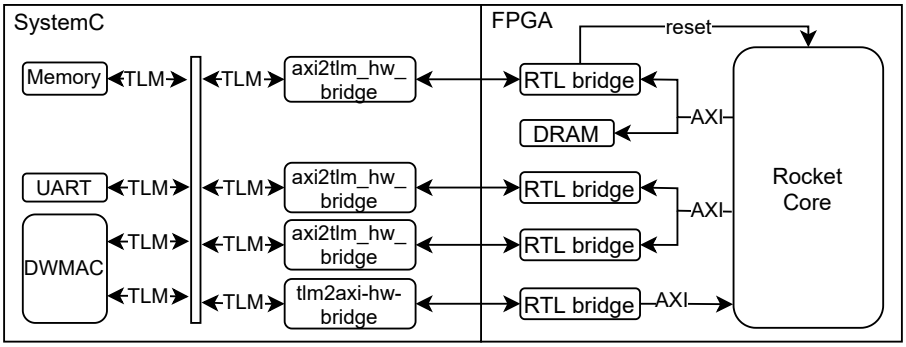


Fig. 7. Platform with Rocket Core and DWMAC.

which is part of the SystemC TLM2.0 standard, and also supported by the QEMU-based CPU model. The measured values are summarized in Table II. While the initial 'NON DMI'-based approach is slower than the standard SystemC TLM2.0 'b_transport' methodology, the DMI approach is 1 order of magnitude faster. The difference between the non-DMI and DMI approaches is stark. This improvement in bandwidth enables SytHIL to offer a more flexible distribution of components between the FPGA and SystemC environment, and it does so with no loss of simulation quality. However, when executing in 'NON DMI' mode, all accesses to memory from the FPGA domain can be traced, while this is not possible in 'DMI' mode. In terms of the DMI approach, it can be observed that writes to the FPGA memory are $4.4x$ faster than reads. This is a property of the PCIe and DRAM memory system itself. Even though the improvements are considerable, a pure SystemC platform outperforms the FPGA mechanism by a factor of 18-154 depending on the access type as moving data to and from the FPGA incurs an unavoidable overhead. One has to keep in mind that DMI is mostly used for memory accesses while SytHIL can perform DMI accesses to peripherals that update their state between transactions.

Next, the results of a more complex benchmarks are examined, specifically looking at the flexibility that the framework gives to place CPUs on the FPGA. In this case Linux boot time is used as a measure of the simulation performance. The first case consists of a VP containing a Rocket core [21] RISC-V CPU and a memory, both placed on the FPGA, and an UART in the SystemC domain, all connected using SytHIL transactors. In the second benchmark the Rocket core was replaced with the BOOM core [22] highlighting the adaptability of the framework. In both cases Linux boot time was measured. Results are shown in Table III, which indicates the boot time for the BOOM core is marginally faster than that of the Rocket core. These preliminary result of approximately 20 s compare favourably with those found by [16] who report equivalent BOOM core boot times of 3.68 minutes. It has to be taken into account that their FPGA platform (ZC706) can only run an unmodified BOOM core at 50 MHz while the AWS FPGA runs at 125 MHz.

TABLE II
BANDWIDTH BENCHMARK RESULTS

Configuration	Result
QEMU & FPGA memory NON DMI	67.1 kB/s (read/write perf.)
QEMU & FPGA memory DMI	3.796 MB/s (Read perf.) 16.6 MB/s(Write perf.)
QEMU & SystemC memory (b_transport)	0.097 MB/s(Read perf.) 0.097 MB/s(Write perf.)
SystemC memory (DMI)	276.2 MB/s(Read perf.) 276.2 MB/s(Write perf.)

TABLE III
LINUX BOOT BENCHMARK RESULTS

Configuration	Result
RocketCore VP	19.0 s (Linux boot time)
BOOM core VP	19.14 s (Linux boot time)

To test the framework in a more complex context a network benchmark was conducted. To make this possible a new platform was built that has the Rocket core on the FPGA and a Synopsys DWMAC model in SystemC. The platform is shown in Fig. 7.

The other components like UART and memory are used to interact with the platform and boot a Linux kernel that has device drivers for the DWMAC. After setting up the TAP interface that is used by the DWMAC model the simulation can be started and networking can take place between the host and the Linux running on the FPGA. iPerf3 was used to measure the network throughput in both transaction directions. The results are shown in Table IV. The throughput from the simulation host to the platform amounted to 29.6 kB/s while the other direction was measured at 9.69 kB/s.

TABLE IV
IPERF3 BENCHMARK RESULTS

Direction	Result
Host to Platform	29.6 kB/s
Platform to Host	9.69 kB/s

V. Conclusion

In this work the SytHIL framework was presented. The framework enables the integration of QEMU models, SystemC models and RTL hardware descriptions into one unified VP. Therefore, SytHIL simplifies the combined testing and verification of both the target software and the target hardware RTL descriptions. To instantiate the RTL descriptions FPGAs are used, as they are faster than RTL simulators. To exhibit the capabilities of our framework relevant use case, such as Linux boot and data transfer via network, were demonstrated and benchmarked. All measurements were carried out using AWS FPGA cloud servers as simulation hosts, as they provide an inexpensive, standardized platform that is both powerful and easily scalable.

Overall, our results show good performance, but care must be taken when designing the VP to ensure that the distribution of components between the various supports is optimal and reflects the use case. One aspect of this is moving data between the FPGA and the host which incurs an overhead. We have shown a significant improvement on data throughput, adopting the SystemC DMI mechanism, which provides more flexibility and performance.

Eventhough, the SytHIL framework is already in use industrially we plan to extend it. In future work we will improve the performance of the SytHIL framework by enabling DMI access from the FPGA to the SystemC domain. In addition, we will address timing synchronization between the three simulation domains, as this is currently not handled by SytHIL in an efficient fashion.

References

[1] S. Hassoun, M. Kudlugi, D. Pryor, and C. Selvidge, "A transaction-based unified architecture for simulation and emulation," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 13, pp. 278–287, 2 2005.

[2] X. Zhang, F. Hui, Q. Wang, and X. Shen, "Integrated iss and fpga soc hw/sw co-verification environment design," vol. 2, 2008, pp. 1071–1075.

[3] M. B. Ayed and M. Abid, "A fast hardware/software co-verification method using a real hardware acceleration," 2012.

[4] C. Q. Li, H. C. Huang, C. Y. Xiang, A. W. Ruan, and W. Tang, "A novel methodology for hardware acceleration and emulation," in *2011 International Symposium on Integrated Circuits*, 2011, pp. 597–600.

[5] Y. B. Liao, P. Li, A. W. Ruan, Y. W. Wang, W. C. Li, and W. Li, "Hierarchy communication channel in transaction-level hardware/software co-emulation system." Institute of Electrical and Electronics Engineers Inc., 2008, pp. 94–99.

[9] Y.-I. Kim, K.-Y. Aim, H. Shim, W. Yang, Y.-S. Kwon, A. Ki, and C.-M. Kyung, "Automatic generation of software/hardware co-emulation interface for transaction-level communication," in *2005 IEEE VLSI-TSA International Symposium on VLSI Design, Automation and Test, 2005. (VLSI-TSA-DAT).*, 2005, pp. 196–199.

[6] M. G. Seok, T. G. Kim, and D. Park, "An hla-based formal co-simulation approach for rapid prototyping of heterogeneous mixed-signal socs," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E100A, pp. 1374–1383, 7 2017.

[7] D. S. Brahme, S. Cox, J. Gallo, M. Glasser, W. Grundmann, C. N. Ip, W. Paulsen, J. L. Pierce, J. Rose, D. Shea, and K. Whiting, "The transaction-based verification methodology," Cadence Berkeley Labs, Tech. Rep., 2000.

[8] F. Borlenghi, D. Auras, E. M. Witte, T. Kempf, G. Ascheid, R. Leupers, and H. Meyr, "An fpga-accelerated testbed for hardware component development in mimo wireless communication systems," in *2012 International Conference on Embedded Computer Systems (SAMOS)*, 2012, pp. 278–285.

[10] S. Lee, M.-K. Jung, I.-C. Park, and C.-M. Kyung, "isave: a behavioral emulator for in-system algorithm verification," in *Proceedings of Second IEEE Asia Pacific Conference on ASICs. AP-ASIC 2000 (Cat. No.00EX434)*, 2000, pp. 303–306.

[11] H. Shim, S.-H. Lee, Y.-S. Woo, M.-K. Chung, J.-G. Lee, and C.-M. Kyung, "Cycle-accurate verification of ahb-based rtl ip with transaction-level system environment," in *2006 International Symposium on VLSI Design, Automation and Test*. IEEE, 2006, pp. 1–4.

[12] Y. Nakamura, "Software verification for system on a chip using a c/c++ simulator and fpga emulator," in *2006 International Symposium on VLSI Design, Automation and Test*, 2006, pp. 1–4.

[13] N. Kim, H. Choi, S. Lee, S. Lee, I.-C. Park, and C.-M. Kyung, "Virtual chip: making functional models work on real target systems," in *Proceedings 1998 Design and Automation Conference. 35th DAC. (Cat. No.98CH36175)*, 1998, pp. 170–173.

[14] J. Shen, T. Suh, H.-H. S. Lee, S.-L. Lu, and J. Shen, "Initial observations of hardware/software co-simulation using fpga in architecture research," 2006. [Online]. Available: https://www.researchgate.net/publication/228578868

[15] P. Schumacher, M. Mattavelli, A. Chirila-Rus, and R. Turney, "A virtual socket framework for rapid emulation of video and multimedia designs," in *2005 IEEE International Conference on Multimedia and Expo*, 2005, pp. 872–875.

[16] D. Kim, A. Izraelevitz, C. Celio, H. Kim, B. Zimmer, Y. Lee, J. Bachrach, and K. Asanovicc, "Strober: Fast and accurate sample-based energy simulation for arbitrary rtl," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016, pp. 128–139.

[17] F. Bellard, "Qemu, a fast and portable dynamic translator." in *USENIX annual technical conference, FREENIX Track*, vol. 41. Califor-nia, USA, 2005, p. 46.

[18] "Standard Co-Emulation Modeling Interface (SCE-MI) Reference Manual Version 2.4," 11 2016.

[19] G. Delbergue, M. Burton, F. Konrad, B. Le Gal, and C. Jego, "QBox: an industrial solution for virtual platform simulation using QEMU and SystemC TLM-2.0," in *8th European Congress on Embedded Real Time Software and Systems (ERTS 2016)*, TOULOUSE, France, Jan. 2016. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01292317

[20] E. Iglesias, "libsystemctlm-soc," https://github.com/Xilinx/libsystemctlm-soc, 2017.

[21] K. Asanovic, R. Avizienis, J. Bachrach, S. Beamer, D. Biancolin, C. Celio, H. Cook, D. Dabbelt, J. Hauser, A. Izraelevitz *et al.*, "The rocket chip generator," *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-17*, 2016.

[22] C. Celio, D. A. Patterson, and K. Asanovic, "The berkeley out-of-order machine (boom): An industry-competitive, synthesizable, parameterized risc-v processor," *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2015-167*, 2015.

Session We.4.C

Network

Wednesday 1st June

17:00

–

Room Pastel

Checking validity of the min-plus operations involved in the analysis of a real-time embedded network

Marc Boyer[1], Pierre Roux[1],
and Hugo Daigmorte[2]

[1] ONERA / DTIS – Université de Toulouse
F-31055 Toulouse – France
[2] RealTime-at-Work
F-54600 Villers-lès-Nancy – France

Abstract. The network calculus theory is widely used to check that a network satisfies its real-time requirements. Such checking involves a lot of computations in the min-plus dioid. This paper shows how such computations can be formally checked using the Coq proof assistant in a realistic industrial context.

1. Introduction

Network calculus is a theory widely used in industry to check that a real-time network (like AFDX or TSN) provides guaranteed latency bounds to the real-time data flow [7, 8, 15]. When these networks are embedded in a critical system, where faults can lead to severe damages or injuries, a very high level of confidence on these bounds must be provided. Since such bounds are commonly computed by a dedicated tool [21], the correctness of the bounds depends both on the correctness of the algorithms and the correctness of their implementation.

The correctness of the algorithms is commonly ensured by open publication and peer-reviewing while the correctness of the implementation is mainly ensured by code review and tests. More confidence in the implementation can be insured by developing several pieces of software and comparing on the fly that all programs give the same result. Nevertheless, such approach can not detect an error in the algorithm itself [11]. Using proof assistants, such as Coq or Isabelle/HOL, is a way to increase the confidence in a software: one may prove the correctness of the algorithms and derive an implementation from the proof [13]. One may also use a *skeptical approach*, where the proof assistant is not able to check an algorithm or to compute a result, but still able to check that a result is correct. Indeed, checking a solution is commonly much easier that solving a problem (for example, finding vs checking the roots of a polynomial, the decomposition into prime factors, matrix inversion,...). Of course, this approach can be used only for software used at design: if there is a mistake in the algorithm or a bug in the implementation, it will only be detected at runtime, while a completely proved approach will ensure that the software is conform to its specification. Nevertheless, the skeptical approach often requires significantly less effort.

Network calculus is based on the min-plus dioid theory [3], and analyzing a network involves a lot of operations in this theory (like physics involves a lot of matrix manipulations). In a recent work [17], this skeptical approach has been applied to the min-plus dioid of real functions. This paper shows how this approach can be used in an industrial context.

2. Network calculus

Network calculus is a theory designed to compute upper bounds on delay an memory usage in networks. Data transiting through the network are called flows. A flow is modeled by *cumulative curves* at each point in the considered network, $A : \mathbb{R}^+ \to \mathbb{R}^+$ where $A(t)$ represents the cumulative amount of data observed in the flow up to time t at a given point of the network. Possible cumulative curves are specified by envelopes called *arrival functions*. A flow A satisfies an arrival function α when: $\forall t, d \geq 0 : A(t+d) - A(t) \leq \alpha(d)$. For instance, a periodic flow sending frames of size L every T time unit admits as arrival curve $\nu_{L,T} : d \mapsto L \left\lceil \frac{d}{T} \right\rceil$, where $\lceil \cdot \rceil : \mathbb{R}^+ \to \mathbb{N}$ is the ceiling function. All network elements are modeled by *servers*. A n-server S transforms n input flows $(A_1, \ldots, A_n)$ into n output flows $(D_1, \ldots, D_n)$, where all A_i and D_i are cumulative curves. The performance of these servers is specified by *service curves*. A n-server S admits a strict service curve β when, for all busy interval $(t, t+d]$, the aggregate output is at least $\beta(d)$, i.e., $\sum_{i=1}^{n} D_i(t+d) - D_i(t) \geq \beta(d)$.

Among others, a result of network calculus states that, if a server S uses a FIFO policy, if it admits a strict service curve β and each of its incoming flow

A_i admits as arrival curve a function α_i, then the delay experienced by each flow in the server is upper bounded by

$$hDev\left(\sum_{i=1}^{n}\alpha_i, \beta\right) \tag{1}$$

(2)
$$\text{with } hDev(f,g) = \sup_{t\geq 0}\{\inf\{d \mid f(t) \leq g(t+d)\}\}.$$

Network calculus also uses other operators, like the the min-plus convolution $*$ and deconvolution $\oslash$, defined by: $\forall f, g : \mathbb{R}^+ \to \mathbb{R}^+, \forall t \in \mathbb{R}^+$,

$$(f * g)(t) = \inf_{0\leq s\leq t}\{f(t-s) + g(s)\}, \tag{3}$$

$$(f \oslash g)(t) = \sup_{s\geq 0}\{f(t+s) - g(s)\}. \tag{4}$$

3. Related work

To perform actual min-plus computations, one has to settle on a given class of real functions. Two main classes of functions are used in network calculus: the set of concave or convex piecewise linear functions, C[x]PL [18], and the, strictly larger, set of ultimately pseudo-periodic piecewise linear functions, commonly known as UPP [6]. The data structure and algorithms for the CPL class are so simple that they, to our knowledge, have never been published. Nevertheless, they cannot accurately model packetized traffic, whereas the UPP class gives more precise results at the expense of higher computation times [7]. The algorithms of the operators on the UPP class are given in [6].

An open source implementation of the operators on the C[x]PL class can be found in the DISCO network calculus tool [2]. An open source implementation of the UPP class has been developed [4] but is no longer maintained to our knowledge. The Real-Time Calculus toolbox (RTC) does performance analysis of distributed real-time systems [19, 20]. Its kernel implements Variability Characterization Curves (VCC's), a class very close to UPP. None of these implementations has been formally proved correct.

The first work on the formal verification of network calculus computation were presented in [14]. The aim was to verify that a tool was correctly using the network calculus theory. An Isabelle/HOL library was developed, providing the main objects of network calculus and the statement of the main theorems, but not their proofs. They were assumed to be correct, since they have been established in the literature for long. Then, the tool was extended to provide not only a result, but also a proof on how network calculus has been used to produce this result. Then, Isabelle/HOL was in charge of checking the correctness of this proof. The result was taking the form of an algebraic min-plus expression, yet to be computed.

Another piece of work, presented in [16], consists in proving, in Coq, the network calculus results themselves: building the min-plus dioid of functions, the main objects of network calculus and the main theorems (statements and proofs).

The tool CertiCAN [12] is able to produce Coq proofs for some real-time analyses of the CAN protocol. These analyses are not based on the network calculus theory.

The PROSA library also provides proofs of correctness for the response time of real-time systems, but focuses on scheduling tasks for processors [10].

4. Existing tools

4.1. RTaW-Pegase. RTaW-Pegase is a proprietary tool written in Java and developed by the company RealTime-at-Work that can provide network analyses and optimization. It can support many technology types and networks such as: automotive, aerospace and industrial Ethernet TSN, CAN (FD,XL), LIN, Arinc as well as wireless networks for off-board communication.

Its first functionality is to model a network, visualize it and set configuration parameters: priorities, offsets, routing, shapers, transmission schedule, preemption,... The graphical interface allows to manipulate the configurations as shown in Figure 1. In this example, the network is made of 5 switches (in blue) and 14 end systems (in orange) and one of the flows is shown, between "CAM1" and "ECU1".

In addition to timing-accurate simulation, RTaW-Pegase can compute guaranteed upper bounds on message delays using a state-of-the-art analysis in network calculus.

RTaW-Pegase can also generate complete traces of its analyses. These traces contain all the computations performed during the analysis in a format both readable by a human and that can be interpreted by an additional RTaW tool, the Network calculus interpreter. Figure 2 shows a simplified example of such a trace. This trace gives the calculations for two periodic flows (Flow1, Flow2) crossing the first link of a network, arbitrated with FIFO policy. The `assert` expression at last line checks that the computed value is not greater than the value 156/5 computed by RTaW-Pegase.

Thereby, a network calculus expert can read this trace and check that the mathematical operations performed correspond to the result of the network calculus theory. Using the calculation capability of the interpreter, it can perform other computations,

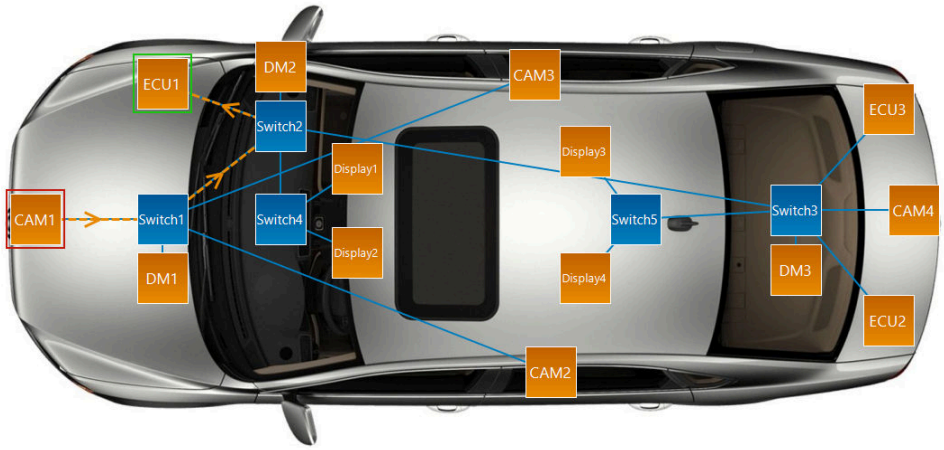


FIGURE 1. Example of visualization in RTaW-Pegase.

```
1 ##################################
2 # Time unit: microsecond
3 # Frame size unit: bit
4 ##################################
5 # Input flows
6 ##################################
7 # Flow1 entering at Node1>port-P1
8 Flow1 := stair(0,10000,1360)
9
10 # Flow2 entering at Node1>port-P1
11 Flow2 := stair(0,5000,1760)
12
13 ##################################
14 # EndSystem : Node1>port-P1
15 ##################################
16 # Computations at priority level 0
17 cumA_Node1 := zero
18
19 # Flow1 in Node1 -> P1
20 cumA_Node1 := cumA_Node1 + Flow1
21
22 # Flow2 in Node1 -> P1
23 cumA_Node1 := cumA_Node1 + Flow2
24
25 # Common service at level 0
26 S_Node1 := affine(100, 0)
27
28 # Common delay
29 d_Node1 := hDev(cumA_Node1, S_Node1)
30
31 assert(d_Node1 <= 156/5)
```

FIGURE 2. Example of a trace that can be produced by RTaW-Pegase

get more values and also plot the functions. An online version is available for non-commercial academic use [1].

4.2. Minerve. Let's look at the last `assert` on Figure 2. To perform a formal proof of this result within the Coq proof assistant, one needs to define the three functions `Flow1`, `Flow2` and `S_Node1`, state the property $hDev(Flow1 + Flow2, S\_Node1) \leq \frac{156}{5}$ and prove this property. Such a proof is given in Figure 3.

First, one loads our tool Minerve [17] (for MIN-plus ExpRession VErification) providing a formalization of the UPP class of functions and an automatic proof tactic to check min-plus computations on those functions. The next line simply instructs Coq to interpret all subsequent numeric constants as arbitrary precision rationals.

Then one needs to define the considered UPP functions. The first function `Flow1` is, as seen on line 8 of Figure 2, a stair function that is incremented by 1360 every 10000, starting from 0. The second function `Flow2` is also a stair function, as seen on line 11 of Figure 2, whereas the third function `S_Node1` is, still according to Figure 2 (line 26), a linear function of slope 100. These functions are encoded by their period `sequpp_d`, increment `sequpp_c` and an initial segment `sequpp_T` (to allow a specific initial behavior before the regular periodic one) and a list of linear segments following the pattern $(x, (y, (\rho, \sigma)))$ where (x, y) are the coordinates of the leftmost point of the segment, ρ is its slope and σ its limit on the right of x (discontinuities are allowed on the right of x, as seen for instance on `Flow1` where the function is $y = 0$ at $x = 0$ and immediately jumps to $\sigma = 1360$ just after 0). Not all combinations of parameters are valid. For instance the period must be positive. Such validity of the parameters is checked by the `F_of_sequpp` function.

Finally, the property to prove is expressed after the `Goal` keyword and automatically proved by our

```
Require Import minerve.tactic.
Local Open Scope bigQ.

Definition Flow1_js : @seqjs bigQ := [::
  (0, (0%:E, (0, 1360%:E)));
  (5000, (1360%:E, (0, 1360%:E)));
  (10000, (1360%:E, (0, 2720%:E)))].
Definition Flow1 := F_of_sequpp {|
  sequpp_T := 5000;
  sequpp_d := 10000;
  sequpp_c := 1360;
  sequpp_js := Flow1_js
|}.

Definition Flow2_js : @seqjs bigQ := [::
  (0, (0%:E, (0, 1760%:E)));
  (2500, (1760%:E, (0, 1760%:E)));
  (5000, (1760%:E, (0, 3520%:E)))].
Definition Flow2 := F_of_sequpp {|
  sequpp_T := 2500;
  sequpp_d := 5000;
  sequpp_c := 1760;
  sequpp_js := Flow2_js
|}.

Definition S_Node1_js : @seqjs bigQ := [::
  (0, (0%:E, (100, 0%:E)))].
Definition S_Node1 := F_of_sequpp {|
  sequpp_T := 0;
  sequpp_d := 1;
  sequpp_c := 100;
  sequpp_js := S_Node1_js
|}.

Goal hDev_bounded (Flow1 + Flow2) S_Node1 (156/5).
Proof. nccoq. Qed.
```

FIGURE 3. Example of Coq proof

`nccoq` tactic [17]. This tactic is a reflexive tactic, which means it makes use of Coq efficient computation capabilities to perform proofs. This is key in getting a very pervasive tactic being entirely developed in Coq. Of course, the proofs automatically performed by the tactic offer the same strong correctness guarantees as any other handmade Coq proof.

It is worth noting the application of the *skeptical approach* here, the horizontal deviation $\frac{156}{5}$ is computed by RTaW-Pegase but only checked with Coq.

4.3. Reading the Coq Specification. To fully trust a Coq proof, one must inspect its statement in order to agree that the proof is indeed proving what the user is expecting. Indeed, whereas Coq can automatically check proofs, it cannot read the user mind to ensure the formal statement match its understanding by the user.

So lets inspect our freshly proved theorem. If one types

```
Unset Printing Notations.
Check hDev_bounded (Flow1 + Flow2) S_Node1 (156/5).
Set Printing Notations.
```

Coq reprints the statement with all notations expanded

```
UPP_PA_refinement.hDev_bounded (F_plus Flow1
  Flow2) S_Node1 (bigQ2rat (BigQ.div (BigQ.Qz
    (BigZ.Pos (BigN.N0 156))) (BigQ.Qz
    (BigZ.Pos (BigN.N0 5)))))
```

in particular, one can see that + was a notation for *F_plus*. One can for instance inspect the latter by asking Coq to print its definition

```
Print F_plus.
```

and Coq answers

```
F_plus = fun f g : F ⇒ f \+ g
```

By deactivating printing of notations again, one could see that \+ is the pointwise addition of functions.

Proceeding with such investigations, one can check that `hDev_bounded` indeed states that the horizontal deviation is bounded and that the definitions of functions introduced with `F_of_sequpp` perfectly match what one is expecting.

Digging further, one could even look at the definition of the field of real numbers $\mathbb{R}$ given by the library we are using.

5. Checking RTaW-Pegase traces with Minerve

The RTaW-Pegase tool allows to compute upper bounds on the delays on the flows crossing a network. As presented in Section 4.1, the engineer enters into the tool the network description, the flow characteristics, and the tool can apply network calculus to check that the system satisfies its latency requirements. The correctness of these bounds depends on a chain of responsibilities. The first link of the chain is the correctness of the network calculus theorems (for example, the result presented in (1)). In case of mistake in the proof, the confidence in the result collapses. Such issue have been addressed in [16]. A second link is the correct application of network calculus results by the tool: if the tool applies a result whose hypotheses are not satisfied by the system, the confidence also collapses. The trace generated by the tool, illustrated in Figure 2, has been designed to allow proofreading by a human expert. A formal proof can also be generated [14]. The last link is the exactness of the computation: the basic operations (sum,

convolution, deconvolution, *hDev*...) are too complex to be checked by humans. This is the aim of Minerve, presented in Section 4.2.

The approach presented in the current work boils down to checking that all computations done in a given RTaW-Pegase trace have provided a sound result, i.e., verifying that the transformations of expressions into values have been performed correctly.

But the traces generated by RTaW-Pegase have been designed to be read by a network calculus expert, to increase confidence in the operations, whereas Minerve has been designed for proof simplicity.

To make both tools run together, we had to solve a few issues:

(1) *Function syntax:* Both tools do not have the same syntax to represent functions. In particular, the domain of the non-periodic part (the initial prefix) of a function in Minerve is always a right open interval $[0, T)$, whereas it can be open $[0, T)$ or closed $[0, T]$ in the syntax of RTaW-Pegase.
(2) *Single assignment:* The RTaW-Pegase trace is a script in an imperative programming language, in which identifiers are variables, whose value can be updated along the script lines. On the opposite, Coq is designed to state mathematical definitions, where a given identifier represents the same value all along the proof. There are several ways to solve this problem, and we have basically chosen to resort on a single static assignment transformation.
(3) *Performance tricks:* While going from handmade examples, with dozens of operations, to realistic examples with thousands of operations, we faced some performance problems, not related to the core of the algorithms but related to some details in implementations.

5.1. Function syntax. One problems is that, although both tools represent functions as sequences of segments and spots, RTaW-Pegase's input language accepts segments that can be open or closed on both ends, whereas NCCoq always considers a sequence of spots and open segments.

Consider a function $f : \mathbb{R}^+ \to \mathbb{R}^+$, defined by

$$f(t) = \begin{cases} 1 & \text{if } t \leq 2, \\ t & \text{if } t > 2. \end{cases} \tag{5}$$

In RTaW-Pegase, such a function can be represented as a sequence of two segments, the first one going from point (0,0) to (2,0) with slope 0 (printed as `[(0,0)0(2,0)]`) and a second one going from (2,2), excluded, up to infinity with slope 1 (then printed as `](2,2)1(+Inf,+Inf)[`). In Minerve input, such a function is represented as a sequence of two pairs (spot, open segment), the first one having a spot (0,0) and a segment with origin 0 and slope 0, and the second one having a spot (2,0) and a segment with origin 0 and slope 1. Then, the translation from RTaW-Pegase to Minerve has to split the closed segment into a spot and an open segment, and push the right extremity as a spot of the second segment. Eventually, the translation can be done on a per-segment basis, pushing spots from one segment to another.

A second problem was that RTaW-Pegase may have no periodic part when the function ends with an infinite segment. The transformation into Minerve has to chose an arbitrary period value (we chose 1).

The third problem was that RTaW-Pegase accepts two expressions of the periodicity, whereas Minerve admits only one. In RTaW-Pegase, one may either state that a function is pseudo-periodic when it exists T, d, c such that: $\forall t > T, f(t + d) = f(t) + c$ or: $\forall t \geq T, f(t + d) = f(t) + c$, whereas Minerve only knows about the latter definition. Both conditions are equally expressive, but one has to increase the value of T in order to cast a definition from the former format into the latter one.

Consider, for instance, the function

$$g : t \mapsto \max\left(t + 1, \left\lfloor \frac{t}{2} \right\rfloor\right)$$

plotted in Figure 4. In RTaW-Pegase, it can be represented by a the closed first segment $[(0, 2) - 1/2(2, 0)]$ followed by the left-open segment $](2, 1)0(4, 0)]$ repeated with periodicity 2 and increment 1, as shown in the upper part of Figure 4. In Minerve, such a representation is impossible, and one has to extend the non-periodic prefix, as illustrated in the lower part of Figure 4.

5.2. Static Single assignment. Looking at the trace presented in Figure 2, the correctness of the trace relies on a correct computation of the variable `cumA_Node`. But since this variable is assigned three times in the trace, there is no single value to check. We may consider the line number of the expression as a tie-breaker, but it is somehow fragile. Then, RTaW-Pegase can generate an enhanced trace, augmented with comments giving an number identifier to each expression to check. This enhanced trace also contains as assertions the expected values of the operands and results for each operation. For instance, the line 20 in the listing of Figure 2 is replaced by the set of lines presented in Figure 5.2.

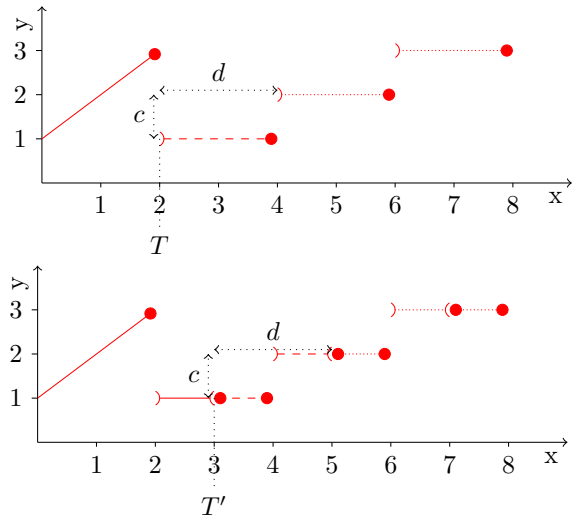


FIGURE 4. Prefix extention due to translation from RTaW-Pegase to NCCoq

```
# MP-Coq-Check 1
assert( cumA_Node1 = uaf([(0,0)0(+Infinity,0)
    []))
assert( Flow1 = upp([(0,0)], period(](0,1360)
    0(10000,1360)]), 1360, 10000))
cumA_Node1 := cumA_Node1 + Flow1
assert( cumA_Node1 = upp([(0,0)], period
    (](0,1360)0(10000,1360)]), 1360, 10000))
# Coq Check 1: cumA_Node1 = (cumA_Node1 +
    Flow1)
```

FIGURE 5. Example of a trace that can be produced by RTaW-Pegase

Then a Coq file is generated with a list of definitions and proof obligations. For instance, from the part of the trace presented in Figure the Coq code presented in Figure 6 is generated. The `idtac` and `Time` commands are there to log the time taken by Coq to perform the proof.

5.3. Performance tricks. Going from simple hand-made examples, with dozens of operations, to realistic examples with thousands of operations, we encountered some performance issues, not related to the core of the algorithms but related to some details in implementations. We sum them up here.

In a first version of the files we used `Let` instead of `Definition`. While this was apparently inocuous with only a few occurrences, this became a major issue with hundreds or thousands of occurences as this means the terms computed by Coq for the last check was embedding all previous definitions, dramatically slowing down computations. Replacing the keyword `Let` by `Definition` was enough to fix the issue.

```
(* Proof for Check 1*)
(* Conversion of upp([(0,0)0(1,0)[, period
    ([(1,0)0(2,0)[), 0, 1) *)
Definition lop_1_js : @seqjs bigQ := [::
  (0, ( (0)%:E, (0, (0)%:E)));
  (1, ( (0)%:E, (0, (0)%:E)))]
%bigQ.
Definition lop_1 := F_of_sequpp {|
  sequpp_T := 1;
  sequpp_d := 1;
  sequpp_c := 0;
  sequpp_js := lop_1_js
|}%bigQ.
(* Conversion of upp([(0,0)0(0,0)], period
    (](0,1360)0(10000,1360)]), 1360, 10000) *)
Definition rop_1_js : @seqjs bigQ := [::
  (0, ( (0)%:E, (0, (1360)%:E)));
  (5000, ( (1360)%:E, (0, (1360)%:E)));
  (10000, ( (1360)%:E, (0, (2720)%:E)))]
%bigQ.
Definition rop_1 := F_of_sequpp {|
  sequpp_T := 5000;
  sequpp_d := 10000;
  sequpp_c := 1360;
  sequpp_js := rop_1_js
|}%bigQ.
(* Conversion of upp([(0,0)0(0,0)], period
    (](0,1360)0(10000,1360)]), 1360, 10000) *)
Definition res_1_js : @seqjs bigQ := [::
  (0, ( (0)%:E, (0, (1360)%:E)));
  (5000, ( (1360)%:E, (0, (1360)%:E)));
  (10000, ( (1360)%:E, (0, (2720)%:E)))]
%bigQ.
Definition res_1 := F_of_sequpp {|
  sequpp_T := 5000;
  sequpp_d := 10000;
  sequpp_c := 1360;
  sequpp_js := res_1_js
|}%bigQ.
Goal res_1 = (lop_1 + rop_1).
Proof. idtac "Check 1". Time nccoq. Qed.
```

FIGURE 6. Coq proof generated from listing in Figure 5.2

As a first step, our automatic tactic `nccoq` maps each min-plus operation, like $+$ or $*$ introduced in Section 2, on arbitrary functions to effective operators on UPP functions, our tactic uses the typeclass resolution mechanism of Coq. This mechanism can perform exponential searches and is known to easily lead to serious slow downs. After doing some profiling, we discovered that more time was spent in this resolution than actually performing the computation of the reflexive tactic. Putting the definitions of the sequences `sequpp_js` in a separate definition

(they were originally inlined) solved the issue by enabling the type class search procedure of Coq to find the expected solution much earlier.

Finally, the largest computations first appeared much slower than expected. Inspecting the intermediate values computed, we discovered rational numbers such as $\frac{17498164897827\times10^{5000}}{38753975857\times10^{5000}}$. Further investigation revealed that we were using non normalizing operations on arbitrary precision rational numbers. Replacing them with the normalizing operations solved the issue.

In Minerve, all functions are periodic with a given period (c.f. d in Figure 4). For affine functions, this means that an arbitrary period must be chosen. A bad choice can be an issue when computing an operator whose operands have wildly diferent periods. To avoid such issues, a preprocessing was added to change the (fake) period of affine functions in accordance to the period of the other operand before any binary operation.

When performing a proof with Coq, the proof is first elaborated with tactics (everything between `Proof` and `Qed`) then rechecked by the kernel of Coq at `Qed` time. This means any expensive computation performed by the tactics will be performed a second time by the `Qed`. To avoid such duplications, a trick using the `abstract` tactic of Coq is used[1].

6. Benchmarks

We evaluated our approach on three midsize to large case studies representative of actual industrial use cases.

The first case study, is a medium size network made of 8 end systems and 2 switches. It is crossed by 57 flows, each having 1 to 5 receivers. The links data rate is set to 100Mb/s except for the link between the two switches which is at 1000Mb/s. For the service policy used, all the flows are distributed between 5 priority levels, at the same level of priority flows respect the FIFO rule. The analysis of such a network with RTaW-Pegase as well as the writing of the trace takes about 1 second.

In the second and third case studies, the considered network is much larger. Comprising 104 end systems and 8 switches, it is crossed by a thousand flows. The links data rate is set to 100Mb/s. For the second case study, all the flows have the same level of priority and are processed in a single FIFO queue. For the third case study, the flows are now distributed in different priority levels and are processed by 5 FIFO queues. RTaW-Pegase analyzes these configurations in about 4 and 8 seconds.

Checking each of these benchmarks involved verifying thousands of operations in each case. We ran the benchmarks on an average few years old laptop with 4 GiB of RAM. The total run time where kept within a couple of hours thanks to checking time per operation well below the second in most cases, with only a handful of operations requiring a few dozen of seconds to be proved correct by Coq. All timings are summarized in Table 1. Note that the last benchmark had to be divided in eight separate Coq files to avoid Coq running out of memory on a huge file.

Although much slower than the initial runs of RTaW-Pegase, we consider these runtimes for verification to be perfectly acceptable considering that they would constitute the last part of the development or certification process of an embedded network. The fat RTaW-Pegase can still be used without extra verification for dimensioning or development purposes.

Last but not least, no bug was found in RTaW-Pegase during the experiments.

The benchmarks, as well as detailed instructions to reproduce those results, are available at `https://doi.org/10.5281/zenodo.5849594`.

7. Conclusion

The skeptical approach (that formally proves the correctness of a result given by a program), is an efficient way to get a high level of confidence in the results of a program. It has been applied in the context of the real-time performances of embedded networks: the correctness of the min-plus operations involved in the computation of real-time bounds can now be checked using the Coq proof assistant. The theoretical part have been presented in [17]. This paper presents how it can be used in an industrial context.

This research experiment can be seen as a success both in terms of development effort and scalability of the verification. Developing a min-plus toolbox requires about one man-year of development [5, 9], developing our min-plus checker required one PhD year of development [17] and plugging the min-plus checker and RTaW-Pegase required about 1 month of development. On run-time, analyzing a network with the RTaW-Pegase requires typically a few minutes using a common laptop whereas checking the results requires a few hours on the same hardware. For software with such a confidence level, the overhead in development time and running time is fully acceptable.

As a future work, one could consider unifying the previous works in [16], [14] and the current work in

[1] This tactic basically seals a computation into an auxiliary lemma (with its own `Qed`), then the final `Qed` just checks the auxiliary lemma statement rather than completely rechecking it.

benchmark	#op	time	time / op	max time
MEDIUM_SP	1923	4:44	0.15	15.23
BIG_FIFO	34121	47:35	0.08	7.61
BIG_SP	81333	4:19:38	0.19	20.28

TABLE 1. Benchmarks: "#op" is the number of operations, "Time" the total time for Coq to check all operations (hours:minutes:seconds), "time / op" the average time per operations (in seconds), "max time" the maximum time to check a single operation (in seconds).

a single Coq proof. This would greatly reduce the trusted code base and avoid having a Network Calculus expert check the output trace of RTaW-Pegase and its mapping to the verified Coq file.

References

[1] RealTime-at-Work online Min-Plus interpreter for Network Calculus. https://www.realtimeatwork.com/minplus-playground.

[2] Steffen Bondorf and Jens B. Schmitt. The DiscoDNC v2 – A Comprehensive Tool for Deterministic Network Calculus. In *Proc. of the International Conference on Performance Evaluation Methodologies and Tools*, ValueTools '14, pages 44–49, December 2014.

[3] Anne Bouillard, Marc Boyer, and Euriell Le Corronc. *Deterministic Network Calculus – From theory to practical implementation*. Number ISBN: 978-1-119-56341-9. Wiley, 2018.

[4] Anne Bouillard, Bertrand Cottenceau, Bruno Gaujal, Laurent Hardouin, Sebastien Lagrange, Mehdi Lhommeau, and Eric Thierry. COINC library: a toolbox for the network calculus. In *Proc. of the 4th int. conf. on performance evaluation methodologies and tools (ValueTools)*, volume 9, 2009.

[5] Anne Bouillard and Éric Thierry. An algorithmic toolbox for network calculus. *Discrete Event Dynamic Systems*, 18(1):3–49, october 2008. http://www.springerlink.com/content/876x51r6647r8g68/.

[6] Anne Bouillard and Eric Thierry. An Algorithmic Toolbox for Network Calculus. *Discrete Event Dynamic Systems: Theory and Applications*, 18, 03 2008.

[7] Marc Boyer, Jörn Migge, and Marc Fumey. PEGASE, a robust and efficient tool for worst case network traversal time. In *Proc. of the SAE 2011 AeroTech Congress & Exhibition*, Toulouse, France, 2011. SAE International.

[8] Marc Boyer, Nicolas Navet, and Marc Fumey. Experimental assessment of timing verification techniques for AFDX. In *Proc. of the 6th Int. Congress on Embedded Real Time Software and Systems*, Toulouse, France, February 2012.

[9] Marc Boyer, Nicolas Navet, Xavier Olive, and Eric Thierry. The PEGASE project: precise and scalable temporal analysis for aerospace communication systems with network calculus. In T. Margaria and B. Steffen, editors, *Proceedings of the 4th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation (ISoLA 2010)*, LNCS. Springer, 2010.

[10] Felipe Cerqueira, Felix Stutz, and Björn B Brandenburg. PROSA: A case for readable mechanized schedulability analysis. In *2016 28th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 273–284. IEEE, 2016.

[11] Robert Davis, Alan Burns, Reinder Bril, and Johan Lukkien. Controller area network (CAN) schedulability analysis: Refuted, revisited and revised. *Real-Time Systems*, 35:239–272, 2007. 10.1007/s11241-007-9012-7.

[12] Pascal Fradet, Xiaojie Guo, Jean-François Monin, and Sophie Quinton. Certican: A tool for the coq certification of CAN analysis results. In Björn B. Brandenburg, editor, *25th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2019, Montreal, QC, Canada, April 16-18, 2019*, pages 182–191. IEEE, 2019.

[13] Pierre Letouzey. Extraction in coq: An overview. In Arnold Beckmann, Costas Dimitracopoulos, and Benedikt Löwe, editors, *Logic and Theory of Algorithms, 4th Conference on Computability in Europe, CiE 2008, Athens, Greece, June 15-20, 2008, Proceedings*, volume 5028 of *Lecture Notes in Computer Science*, pages 359–369. Springer, 2008.

[14] Etienne Mabille, Marc Boyer, Loïc Fejoz, and Stephan Merz. Towards certifying network calculus. In *Proc. of the 4th Conference on Interactive Theorem Proving (ITP 2013)*, Rennes, France, July 2013.

[15] Lisa Maile, Kai-Steffen Hielscher, and Reinhard German. Network calculus results for tsn: An introduction. In *Proc. of the Information Communication Technologies Conference (ICTC 2020)*, pages 131–140. IEEE, 2020.

[16] Lucien Rakotomalala, Marc Boyer, and Pierre Roux. Formal Verification of Real-time Networks. In *JRWRTC 2019, Junior Workshop RTNS 2019*, Toulouse, France, November 2019.

[17] Lucien Rakotomalala, Marc Boyer, and Pierre Roux. Verifying min-plus computations with coq. In *Proc. of the 13th NASA Formal Methods Symposium (NFM 2021)*, May 24-28 2021.

[18] Hanrijanto Sariowan, Rene L. Cruz, and George C. Polyzos. SCED: A generalized scheduling policy for guaranteeing quality-of-service. *IEEE/ACM transactions on networking*, 7(5):669–684, October 1999.

[19] E. Wandeler. Modular performance analysis and interface based design for embedded real time systems. 2006.

[20] Ernesto Wandeler and Lothar Thiele. Real-Time Calculus (RTC) Toolbox, 2006.

[21] Boyang Zhou, Isaac Howenstine, Siraphob Limprapaipong, and Liang Cheng. A survey on network calculus tools for network infrastructure in real-time systems. *IEEE Access*, 8:223588–223605, 2020.

Assessing a precise gPTP simulator with IEEE802.1AS hardware measurements

Quentin Bailleul
IRT Saint Exupéry
Toulouse, France
quentin.bailleul@irt-saintexupery.com

Katia Jaffrès-Runser, Jean-Luc Scharbarg
IRIT, Université de Toulouse, CNRS, Toulouse INP, UT3
Toulouse, France
{kjr, jean-luc.scharbarg}@enseeiht.fr

Philippe Cuenot
IRT Saint Exupery, Seconded from Continental Automotive France
Toulouse, France
philippe.cuenot@continental-corporation.com

***Abstract*—TSN (Time Sensitive Networking) standards are arousing growing interest in the critical on-board network community because of their promise of deterministic Ethernet networking. Among these standards IEEE802.1AS allows the synchronization of network devices. This protocol achieves much greater precision than other Ethernet synchronization standard such as NTP or PTP. Thus, it paves the way for new network mechanisms, such as the Time Aware Shaper, and allows the use of applications that are more constrained in terms of synchronization. In order to study the new mechanisms allowing to reach this precision, precise simulations are mandatory. In this paper, we review the different existing tools, extend the most promising one to incorporate the most advanced features of IEEE802.1AS and assess its behavior with respect to measurements. More specifically, we extend an OMNeT++ simulation library, calibrate its results following an extensive measurement campaign of switched Ethernet devices supporting IEEE 802.1AS to assess its performance in terms of precision.**

***Keywords*—IEEE802.1AS, gPTP, TSN, Synchronization, Simulation**

I. Introduction

Critical on-board network architecture cannot cope with the diversification of flows and their constraints. Thus, real-time Ethernet solutions have been designed to bring much higher bandwidth and advanced quality of service policies. Promising solutions are the standards proposed by the IEEE Time Sensitive Networking (TSN) working group. Among these standards, central synchronous shapers like the Time Aware Shaper (TAS) allow the transmission of zero-jitter time-bounded low latency flows. Such shapers rely on a network-wide precise and accurate synchronisation of the devices. The IEEE TSN group has standardized the IEEE 802.1AS synchronization protocol [1] we are investigating in this paper.

The IEEE 802.1AS protocol is a profile of the IEEE 1588 [2] synchronization standard already in use in non-critical systems. IEEE 802.1AS has been designed with the goal of reaching a precision of less than 1 microsecond in a linear network setting where a master clock and an ordinary clock are separated by 7 hops. The precision is defined as the difference between the clock under test and the reference clock. Experimental measurements [3] as well as simulation and worst-case analysis [4] have assessed such precision. Simulation allows the evaluation of this synchronization protocol at a lower cost, provided that simulation assumptions are derived from measurements. To the best of our knowledge, existing simulation libraries have not been compared to real devices supporting IEEE 802.1AS.

In this paper, we first review the already available simulation tools for IEEE802.1AS and select the most advanced open source OMNET++ simulation library. Then, we extend this library to incorporate the most advanced features of IEEE802.1AS and compare its performance to experimental measurements of real IEEE802.1AS devices. From this step, we propose a calibration step of the simulator and assess, after calibration, its precision to validate its behavior in relation to reality. Our contribution is a realistic open source simulation model for IEEE802.1AS that reproduces the behavior of real devices. Thanks to the improvements made to the simulation model and the calibration of the jitter linked to the PHY layer, the drift and the granularity of the clock using measurements on real devices, we achieve a Root Mean Squared Error (RMSE) of approximately 3 ns between the measured and simulated sliding average of the synchronization precision.

This paper is organized as follows. First, we present IEEE 802.1AS and the related work on IEEE802.1AS simulation in Section II. In Section III, we discuss our changes to an existing library. Then in section IV, we present our experimental protocols and discuss our results. And finally, we conclude and present future work in section V.

II. Context

A. IEEE 802.1AS overview

IEEE 802.1AS [1] is a profile of IEEE 1588 Precision Timing Protocol (PTP) [2] for Time Sensitive Networking. Sometimes called generalized Precision Time Protocol (gPTP), this protocol is used to synchronize clocks across a network using the master slave paradigm. Each port is in one of the following three states: master, slave, passive. Master ports

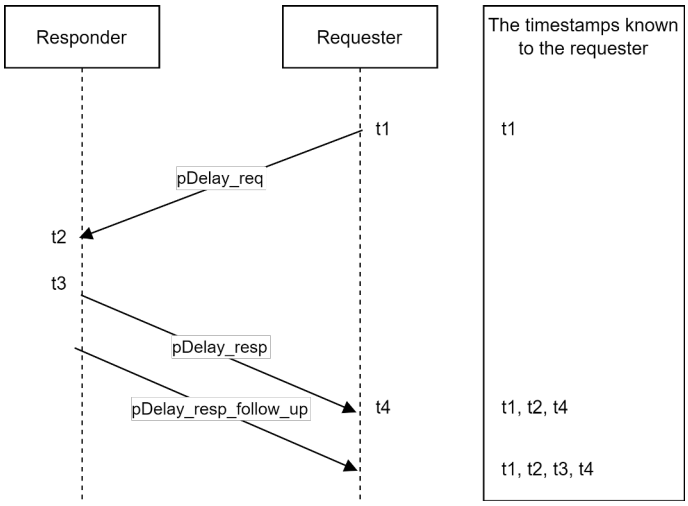


Fig. 1: Peer-to-Peer delay mechanism.

periodically broadcast time synchronization messages. Slave ports process these messages on reception, while passive ports ignore them to avoid loops in the distribution. The time-aware system with all its ports in master state is called Grandmaster and it is the time source of the network. It can be synchronized by an external time source (GPS, NTP, ...) or use its own clock.

To determine port states, IEEE 802.1AS provides two methods. The first one is the external port state configuration. It statically defines the state of the ports for all the devices involved in the synchronization. The second method is the Best Master Clock Algorithm (BMCA). This distributed algorithm is executed on each time-aware system to eventually determine the state of the local ports and to elect the Grandmaster by comparing the information received from each Grandmaster candidate.

Synchronization itself is based on two core mechanisms: *i)* the measurement of the link propagation delay using the Peer-to-Peer delay mechanism and *ii)* the distribution of synchronization information.

The Peer-to-Peer delay mechanism defined in IEEE 802.1AS measures the link propagation delay between two time-aware systems that are separated by one hop. `Pdelay` messages carrying timestamps are exchanged every *Pdelay interval* (1s by default). Measurement of propagation delay needs four timestamps as depicted in Fig. 1. $t1$ is measured when the `Pdelay_req` is issued. $t2$ is obtained upon reception of this message. $t3$ is measured when the `Pdelay_resp` is sent. Finally, $t4$ is measured upon reception of `Pdelay_resp`. Formula (1) calculates the delay of the link, T_{prop}, using the timestamps. Moreover, with the $t3$ and $t4$ timestamps of two consecutive Pdelay procedures, the requester can extract the so-called *neighbor rate ratio* nr of Eq. (2) in order to compensate the relative clock drift in Eq. (1).

$$T_{prop} = \frac{(t2 - t3) + nr.(t4 - t1)}{2} \tag{1}$$

$$nr = \frac{f_{req}}{f_{resp}} = \frac{t3_i - t3_{i-1}}{t4_i - t4_{i-1}} \tag{2}$$

Equation (1) assumes that the link is symetric. Existing asymmetries can be compensated if they can be estimated, typically in a calibration step. To smooth the effects of sources of inaccuracies, some devices include filters for T_{prop}.

The distribution mechanism is based on the transmission of `Sync` and `Follow_Up` messages that allow each time-aware system to synchronize to the Grandmaster clock. Every synchronization interval (typically 125ms), the Grandmaster sends a `Sync` message out of its master ports, followed by a `Follow_Up` message containing t_0, the exact transmission time of the `Sync` message, as pictured in Fig. 2. These two messages are received via the slave ports of the device connected to the Grandmaster. If the receiving device has at least one port in the master state, it forwards both messages to the next time-aware system. The `Follow_Up` message carries t_0, and updated values of the *rate ratio* r and the *correction field* C. These two fields are detailed next.

The rate ratio r allows for logical syntonization of a time-aware system to the Grandmaster rate. It is the product of the neighbor rate ratio calculated by the receiver ports on the path going from the Grandmaster to the time-aware system of interest. It is initialized to 1 by the Grandmaster and is updated on each hop with the equation (3), where i is the receiving node and $i-1$ the sending node.

$$r_i = r_{i-1} * nr \tag{3}$$

The *correction field* C carries the time elapsed in the time-aware systems and on the links on the path between the Grandmaster and the time-aware system preceding the last hop. At hop i, C_i is calculated using the previous correction field C_{i-1}, the previous rate ratio r_{i-1}, the neighbor rate ratio nr, the propagation time undergone during the last hop T_{prop} and the residence time T_{res} of the `Sync` in time aware system as given in (4) :

$$C_i = C_{i-1} + r_{i-1} * T_{prop} + r_{i-1} * nr * T_{res} \tag{4}$$

These operations are repeated at each hop, until the complete set of time-aware systems is reached.

Using the two mechanisms described above, each device can calculate the difference between its local clock and the Grandmaster clock in order to deduce the correction to be applied. Indeed, to deduce the Grandmaster time at the `Sync` reception time, the device adds to t_0, the original time of transmission of the `Sync` by the Grandmaster, the correction field carried in the `Follow_Up` message and the propagation delay of the last hop T_{prop} measured with the Peer-to-Peer delay mechanism.

B. Related Work

Since the first version of 802.1AS in 2011, several simulation tools have been created. Garner et al. [3] proposed a simulation model to evaluate a draft of 802.1AS. In order to reduce its complexity, this discrete event simulation library only models events linked to `Sync`, `Pdelay_Req` and `Pdelay_Resp` messages. It approximates operation in one-step mode. Using this simulation, they showed that the proposed synchronization mechanism can cope with Audio/Video application constraints with a network having up to 7 hops. Lim et al. [5] built a simulation model for IEEE802.1AS using OMNeT ++/INET

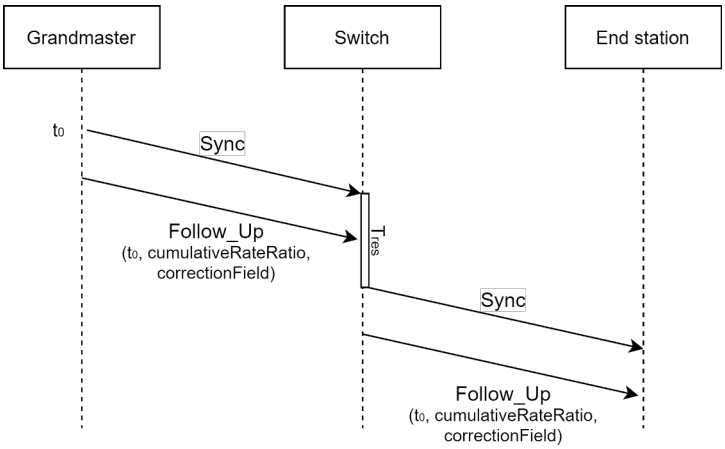


Fig. 2: Synchronization distribution mechanism.

in order to evaluate the performance of the standard on an automotive Ethernet topology. This simulation model uses passive clocks and a static configuration. They highlighted the importance of the choice of filter to smooth out errors in the propagation delay measurement and the improvement in precision when a lower syncInterval is used in the first hop. Gutiérrez et al. [4] developed a simulation library in order to compare probabilistic and worst case precision achievable in a 100 hop network. Their simulation model is based on OMNET ++ / INET with a clock model using a drift that varies over time at a bounded rate. Nevertheless, the neighbor rate ratio is calculated from the perfect neighbor rate ratio and the maximum error allowed by the standard, which makes the Pdelay calculation pessimistic. However, these different simulation libraries are not open source and haven't been assessed by real measurements.

A few works propose open source libraries, e.g. the one proposed by Puttnies et al. [6]. They developed a IEEE802.1AS simulation model, using OMNeT++ [7] with the INET framework [8], containing the core time synchronization and propagation delay measurements operations. The model uses a simple clock model with constant drift. The BMCA is not implemented because they focus on achievable precision. Obtained results have been compared with the simulation results of Lim et al. [5]. Wallner et al [9] also built an open source simulation framework for IEEE1588-2008, also based on OMNeT++/INET, with complex clock models using realistic noises. This very complete simulator allows the use of many PTP mechanisms such as End-to-End or Peer-to-Peer delay measurement, BMCA and transparent clock mechanisms.

The simulation library of Wallner et al. [9] is of higher complexity than the one of Puttnies et al. [6] due to the implementation of many PTP mechanisms that are not part of IEEE802.1AS. As such, we decided to carry out this work on the basis of the library of Puttnies et al. [6] that supports the core functions of IEEE802.1AS by design. In order to compare the simulation accuracy with real measurements, we had to extend this library with new features to better capture all mechanisms of IEEE802.1AS. Resulting extensions are presented next.

III. gPTP Simulation model extensions

In order to improve the simulation accuracy of [6], we had to extend the simulation model to account for clock syntonization on the one hand, and to better capture core platform-dependant features such as clock granularity and jitter of the link delay due to the PHY layer on the other hand.

A. Logical syntonization

As described above, logical syntonization is essential to reach higher levels of synchronization precision. Indeed, it allows to take into account the local drift compared to the Grandmaster one when updating the *correction field*. Without this mechanism, the relative drift between the Grandmaster and the time aware system is not compensated for, causing a wrong estimate of the correction to be applied to the clock. This error is also propagated to the devices on slave ports with the *correction field*. We have added this syntonization step following the IEEE802.1AS standard to the simulator of [6].

B. Towards a more realistic model

To better capture the real behavior of devices supporting IEEE802.1AS, the following sources of imprecision have been added to the simulator.

The first one is the granularity of the clock. This is the step at which the clock ticks, for instance 10ns. Thus, when a duration is measured between two timestamps, e.g. a duration of the propagation delay computation or the residence time of a `Sync` message, the measurement error varies between -10ns and + 10ns. In the simulator, we round the duration to the immediately lower multiple of 10ns to capture clock granularity.

The second source of imprecision is the inaccuracy related to the PHY layer that occurs at the reception device. Despite the hardware timestamping used in devices supporting IEEE 802.1AS, which eliminates software-related jitter, the PHY layer causes an implementation dependent jitter. As shown by Loschmidt et al. in [10], the jitter linked to the PHY layer depends on the protocol used. They show for instance that the jitter is higher with 10Base-T than with 100Base-T. Moreover, a propagation delay asymmetry may exist whose magnitude depends on the initialization of the link PHY layer. These two PHY layer phenomenons have significant impact on the synchronization results since they change the propagation delay statistics and values.

In the following we consider a 100Base-T PHY. Based on [10], a random jitter is added to the reception timestamp of a message. It follows a normal distribution with zero mean and 0.286ns standard deviation. Additionally, a link delay asymmetry may be introduced at link initialization by constant latency. This asymmetry is rooted in the Phased Lock Loop (PLL) system that may lock on dissimilar edges of the signal at the RX interface on both ends of the link. If the PLLs of the two RX interfaces at both ends of the same link lock on dissimilar edges, then the propagation delay in both ways is asymmetric, which causes an error in the estimation of the propagation delay. The 5 possible edges being 8ns apart, the worst asymmetry is therefore of 32ns which causes an error

Protocol Parameters	Value
Sync Interval	0,125s
Pdelay Interval	1s
SyncLocked	True

TABLE I: Protocol parameters

of 16ns when calculating the propagation delay by the time aware system. This error leads to errors in the estimation of the time spent by the `Sync` messages on the link and therefore inaccuracies in the synchronization.

There are other sources of inaccuracy such as PLL or oscillator noise. They can be neglected since their impact on the delay is less than one nanosecond [10].

IV. Experiments

In the previous section, we detailed the changes made to improve the simulation library's realism. This section aims to calibrate and validate the choices made during the implementation and to identify calibration steps to improve simulation accuracy. For this, we will measure and analyse first, on the real TSN switch, the behavior of the switches clock. Second, we challenge the results obtained by the link delay measurement mechanism with the simulated ones. And finally we compare the precision of the synchronization IEEE802.1AS to the simulation results to validate the simulator's accuracy. From the first two steps, we extract calibration steps that can be performed to adjust the simulator to a specific 802.1 PHY technology.

A. Experimental setup

The goal is to calibrate and validate the behavior of the simulation library using real devices. Configuration parameters of IEEE802.1AS are given in Table I. The library is configured with the help of the values determined in the rest of this paper to get as close as possible to the behavior of real device.

Our experimental testbed, pictured in Fig. 3 consists of:

- 4 Fraunhofer IPMS TSN Multiport Switch Core - TSN-SW v0.5.0 on Netleap boards,
- a netTimeLogic PPS analyer,
- 100Base-T Ethernet link.

As described in Fig. 3, the four switches are connected in a chain topology. We capture the progress of the different clocks using the PPS Analyzer. One of our experiments described later requiring greater accuracy required the use of a better quality reference clock as a reference for the PPS analyzer. For this we used a Meinberg microSync HR slaved on GNSS time connected to the reference input of the PPS analyzer with a PPS link. And finally, to configure and retrieve internal switch values, such as the result of the delay propagation calculation, we use the serial port of the switches to communicate with a computer.

Simulations also consider the same topology. Unlike a one-hop topology, this multi-hop topology allows an evaluation of the impact of the *correction field*. The measurements are

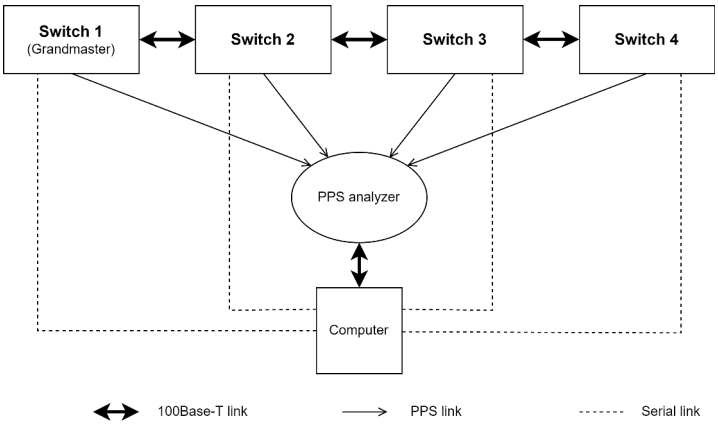


Fig. 3: Experimental measurement topology.

carried out by OMNeT ++ at instants following precise events, e.g. the end of the propagation delay measurement or just before and just after the correction of the clock.

To determine the duration of experimentation and simulation necessary to obtain significant results, we studied the evolution of the mean squared error (MSE) between the normalized distribution of value obtain by the Pdelay mechanism of a given experiment duration compared to a experiment duration of 32h. A duration of 3600s corresponds repeatedly to an MSE of around 10^{-5} % of the number of occurrences as shown in the fig 4. Fig 5 makes it possible to observe the small difference between the distribution of value obtain by the Pdelay mechanism obtained after 1 hour of experience and the distribution obtained after 32 hours. The sources of variability, such as the granularity and the PHY jitter, do not depend on the AS mechanism which is studied, we will therefore use this duration of experimentation for the other mechanisms and not only for the Pdelay mechanism. Thus, for the rest of this study, we take measurements for 3600s.

The following sections present the three experiments that we carried out to compare the operation of the simulator with the operation of real devices in order to calibrate and validate this simulation model.

B. Clock calibration

In [6], Puttnies et al. have chosen to implement a simple constant drift clock. In order to validate this choice and adjust the parameters of the simulator clock deviation in relation to our real devices, we have studied the behavior of the clocks of our switches in freerunning during one hour. To do this, we measured with the help of the PPS analyzer the evolution of the drift for each switch compared to the Meinberg microSync HR slaved on GNSS when the synchronization is deactivated. We use the Meinberg microSync HR on this experiment for its clock quality.

Figure 6 and table II shows the results of this experiment. These measurements were taken 10 minutes after starting the devices, so that the temperature of the electronic components is stable and without attempting to control the room temperature. As indicated in the figure 7, presenting multiple results of the

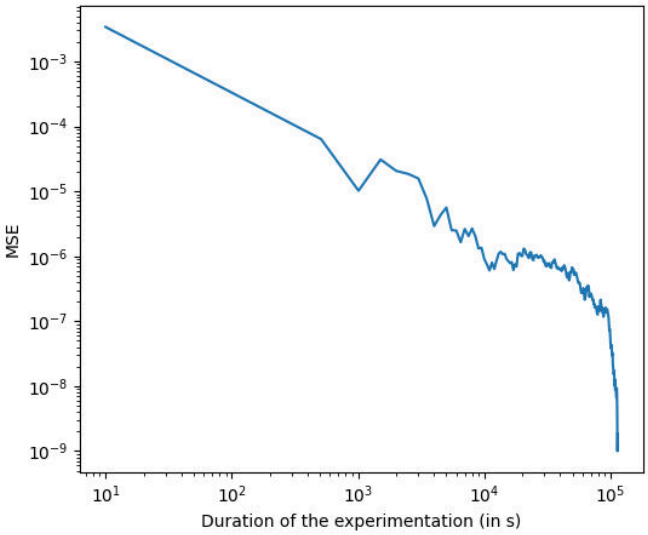


Fig. 4: MSE of the distribution of the results of the Pdelay mechanism according to the duration of the experience compared to a 32 hours experience with the real switches

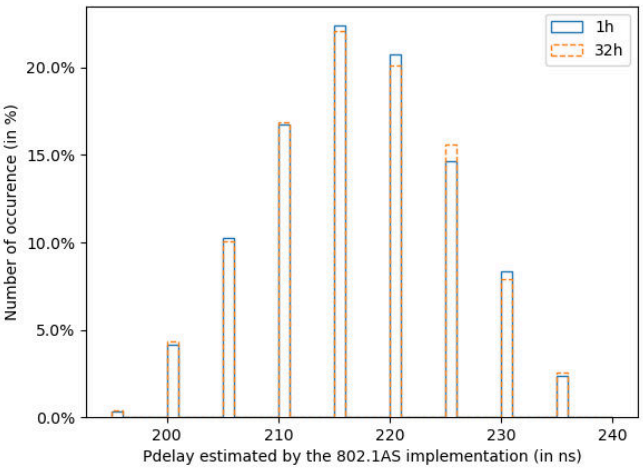


Fig. 5: Pdelay distribution for an 1-hour experience and a 32-hour experience

linear regressions carried out, similar results are obtained for the different repetitions of the measurements during 24h.

Using Fig 6 and the table II, we observe that an implementation of a constant drift clock makes it possible to simulate the behavior of a real clock in freerunning for one hour in an environment where the temperature variations are small. However, by repeating the experiment during 24h, we observe small variations in ppm within a range of 0.009 ppm , as shown in Fig 7 caused by the temperature variation in the room. In order to use up to date ppm value in the simulator, in the rest of our comparison between simulation and reality, we perform a ppm measurement in freeruning compared to the Grandmaster

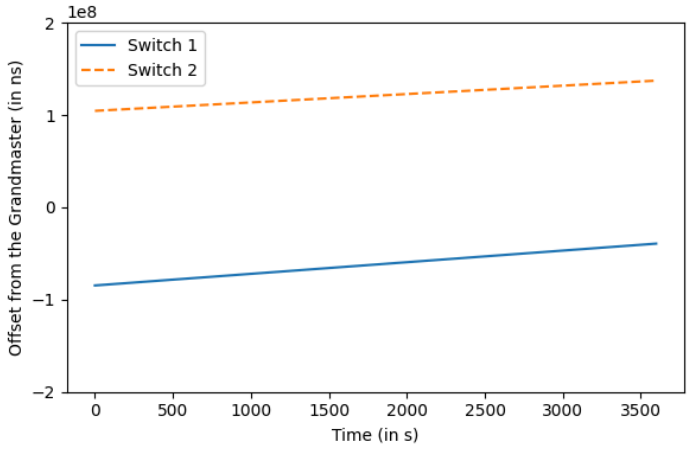


Fig. 6: Grandmaster clock offset of two switches during 3600s

	Switch 1	Switch 2
ppm	12.593	9.094
pvalue	$< 10^{-9}$	$< 10^{-9}$
rvalue	0.99999995	0.99999999

TABLE II: Results of linear regression of the data presented in Fig 6

of the experiment before performing any other measurements on real devices.

C. Canal calibration

In order to adjust and validate the different sources of inaccuracy that we have added to the simulator, we measure the distribution of the values obtained by the Peer-to-Peer delay measurement mechanism without any filter and compare them to the value obtained with the simulator. To configure

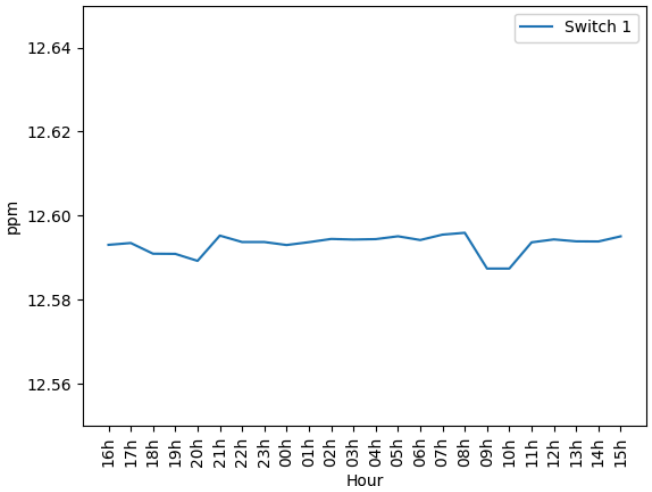


Fig. 7: Result of linear regression of switch 1 clock drift measured for 3600s and repeated for 24 hours.

the simulator, we use the relative drift measured before the experiment and the mean link delay measured during the experiment.

In order to determine the standard deviation to parameterize the normal distribution which adds jitter to the link crossing time in the simulator, we compared the distributions obtained using the simulated and real Peer-to-Peer delay mechanism. Figure 8 shows the MSE obtained between the simulated and measured distribution of the Pdelay as a function of the standard deviation used to parameterize the normal law which causes the PHY jitter when crossing the link. By minizing the MSE, we find a standard deviation of 12.5ns, which is quite different from the 0.286ns measured by Loschmidt et al in [10]. Indeed, our switch embeds a different PHY chip from the one used for their measurements. In addition, our measurement takes place much further in the chain of message transmission. Indeed, our measurement is based on timestamps which takes place between the MAC layer and the PHY layer of the OSI model while their measurement takes place directly at the PHY level using the RX_DV and TX_EN PINs of the MII. Thus other source of jitter can be found between their point of measurement and ours.

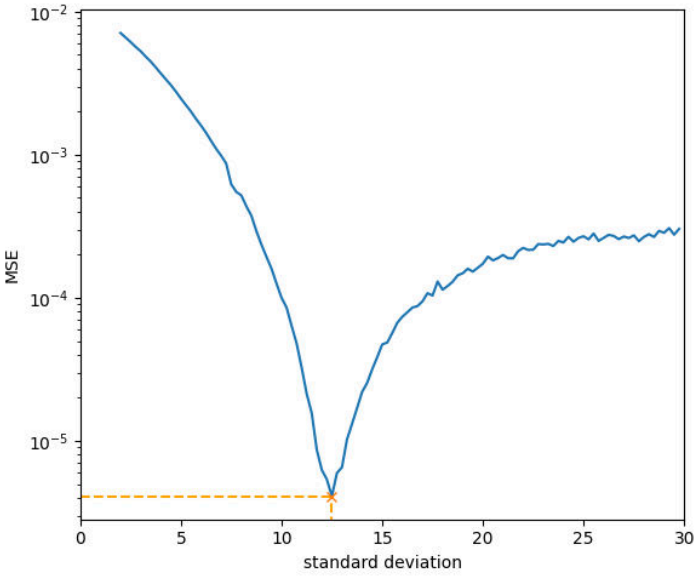


Fig. 8: MSE between the simulated and measured distribution of the Pdelay during 32h according to the standard deviation used to simulate the PHY jitter in the simulator

The figure 9 presents the results obtained where switches are synchronized and exchange Pdelay messages every seconds. We find there the distribution of the different values obtained by the measurement mechanisms of the link delay of one of the switches and of the simulation library obtained during 32h. For this measurement, we use a measurement time much longer than what we have previously determined to be sure that the simulated distribution also covers rare events. The simulation parameters such as mean link delay, granularity and PHY standard deviation were chosen to match the experimental distribution. By analyzing the distribution of the propagation delays obtained by the real device, in Fig 9, we can deduce the granularity of the clocks used by observing the difference between the two consecutive values. Here, the difference being 5ns, the granularity is therefore 10ns, because of the division by two in the formula (1). As can be seen with this figure, our simulation gives a distribution very close to reality although a little more pessimistic than reality but which does not bother us in view of our on-board scope of application.

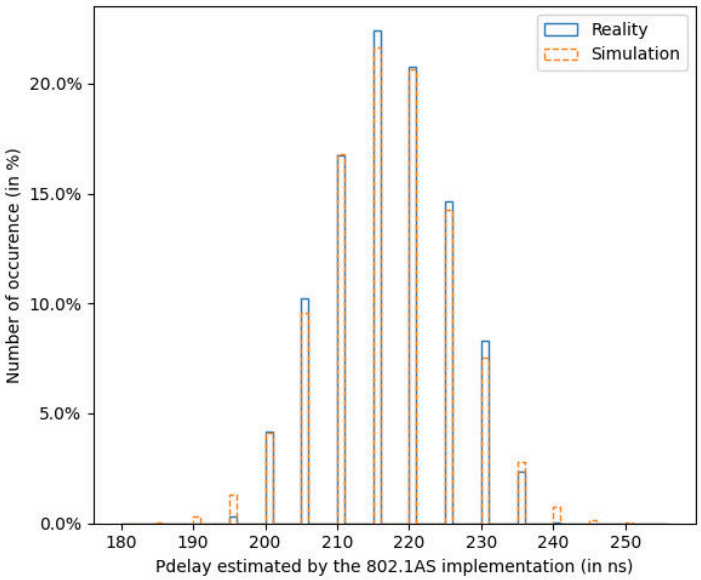


Fig. 9: Distribution of the results obtained by the simulated and real Pdelay mechanisms

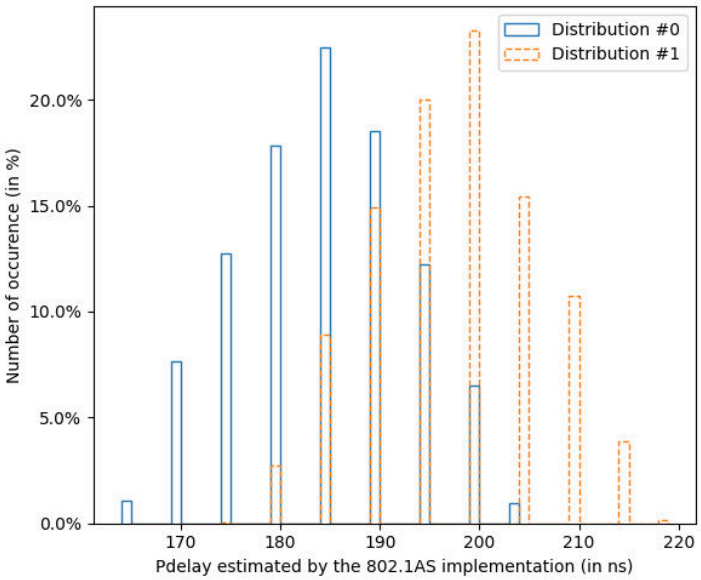


Fig. 10: Distribution of the results obtained during two experiments with the real switches.

When the link between two measurements is reinitialized, a variation of the mean Pdelay, as shown in fig 10, is observed. These two distributions originate from a periodic measurement

of the Pdelay on real devices for which we trigger a link reinitialization every hour. As described by Loschmidt et al. in [11], the difference between the two distributions is a consequence of link delay asymmetry. This asymmetry is induced by the edges on which the PLLs of each physical interface lock during link initialization. If the two PLLs don't lock on the same edge, the various messages of the Pdelay mechanism undergo a different link propagation delay depending on their direction on the link. As such, there is an error in the estimation of the link delay by the Pdelay mechanism since it is based on a symmetrical channel assumption. With the library, we reproduce this behavior by randomly setting the edge on which each interface is locked for each simulation.

D. Validation

In this last part, we compare the synchronisation precision measured with the real TSN switches to the one computed by the OMNeT ++ simulator. This steps validates the different enhancements of the simulation library and the calibration steps proposed in this paper.

1) Bounding the precision.: The measurements are triggered by the PPS analyser every second. The `SYNC` messages are sent about every 125ms. The PPS measurements aren't synchronized with the `SYNC` dates and as such, the measurements may be taken at various times of the synchronization cycle. For instance, if the previous synchronization stage takes place a few nanoseconds before the measurement time, the precision measured using the PPS signal is much better than if the last synchronization stage takes place a few tens of milliseconds before.

Unlike real measurements based on the PPS signal, we can measure in simulation the synchronization precision at specific times which are related to the main protocol execution steps. The worst precision can thus be measured just before the synchronization procedure takes place and the best one just after it. Since we can't compare measurements and simulation at exactly the same dates, we leverage the track the best and the worst synchronization by simulation, and plot the real measurements in the same figure to validate whether measurements lie in between best and worst simulation precision. Figure 11 shows the precision measured on the first hop, as well as the simulated precision before and after sync for 3600s. With this figure, we observe that the simulator allows to bound the result obtained with the real device on the first hop, despite the limitations of the PPS signal. There results are reasonable, even though they don't allow us to judge the accuracy of the bounds obtained with the simulator.

2) Accuracy of simulated precision bounds: By repeating the previous experiment, we observed gaps in the precision measured using the PPS signal as shown in the figures 12, 13 and 14. These figures show the measured precision, as well as the simulated precision before and after synchronization respectively at the first, second and third hops. These PPS gaps represent the situation where we move from measuring precision just after synchronization takes place to measuring precision in reality just before synchronization takes place. Over a campaign of 200 one-hour measurements, we have observes these PPS gaps 19 times.

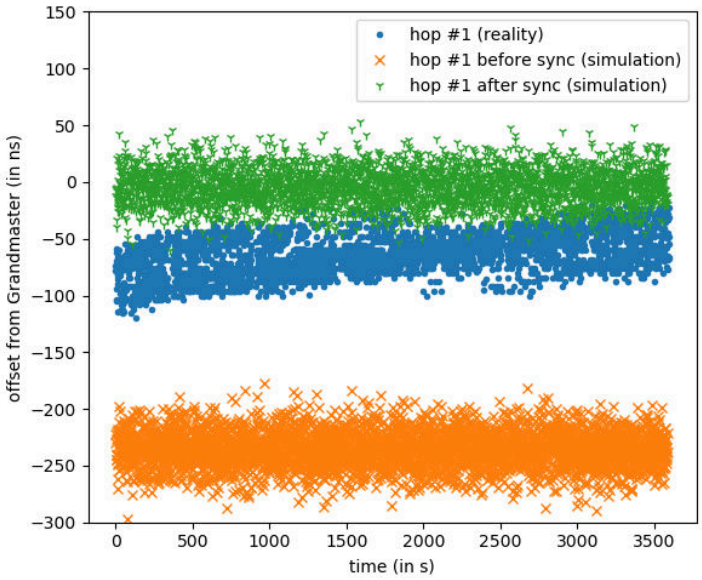


Fig. 11: Offset between the Grandmaster clock and the switch clock at hop #1 in reality and the simulator. The simulator is configured with the granularity, mean link delay and the standard deviation estimated in the previous experiments. For the clock drift, we use the drift measured before each experiment.

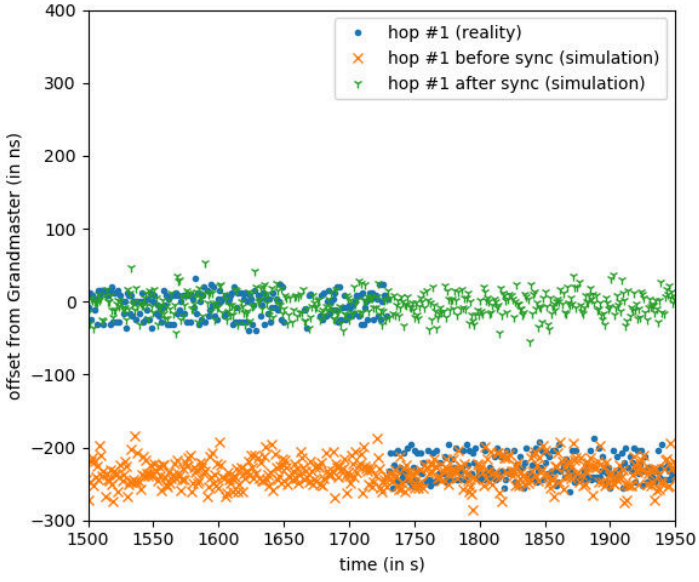


Fig. 12: Identification of a PPS gap in the offset measurement between the Grandmaster clock and the switch clock at hop #1 in reality. Simulated best and worst offset are plotted as well.

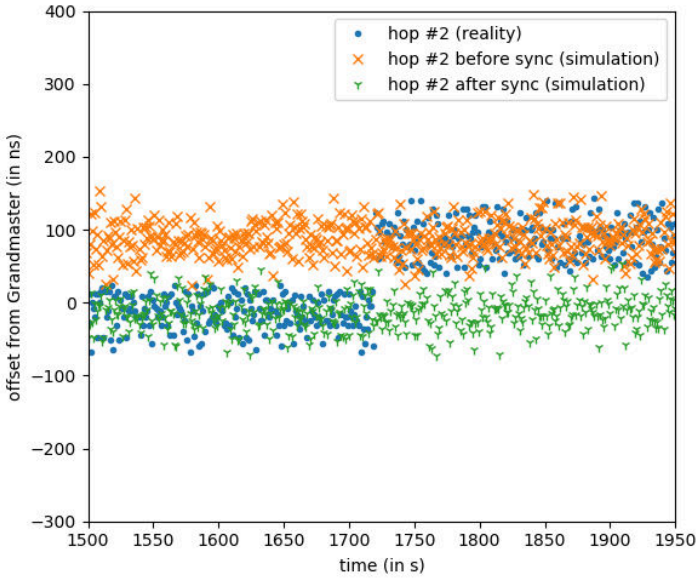


Fig. 13: Identification of a PPS gap in the offset measurement between the Grandmaster clock and the switch clock at hop #2 in reality. Simulated best and worst offset are plotted as well.

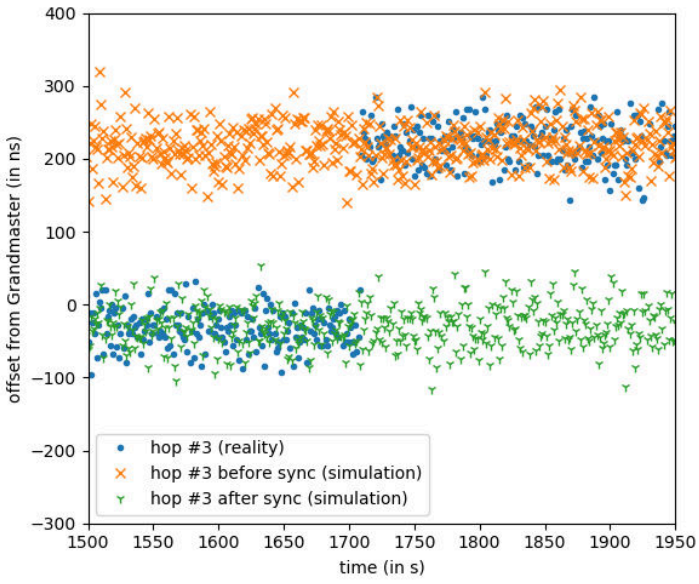


Fig. 14: Identification of a PPS gap in the offset measurement between the Grandmaster clock and the switch clock at hop #3 in reality. Simulated best and worst offset are plotted as well.

	Hop 1	Hop 2	Hop 3
Minimal RMSE (ns)	0.89	1.1	2.1
Maximal RMSE (ns)	7.1	4.8	6.2
Median RMSE (ns)	2.3	2.8	3.9
Mean RMSE (ns)	3.4	2.9	4.2

TABLE III: Result of the RMSE calculation using 10 different simulations

	Switch 2	Switch 3	Switch 4
ppm	-1.850	0.817	1.997

TABLE IV: Clock drift of the different devices measured before the simulator validation experiment

Thus, close to the PPS gap, it's possible to determine when the synchronization cycle takes place. In this situation, it becomes possible to compare the bounds obtained with the simulator with the worst and best precision measured around the PPS gap. With the figures 12, 13 and 14, it can be seen that the simulator makes it possible to precisely limit the synchronization precision reachable with this Ethernet switch. Indeed, we observe the best case, just after the synchronization, and the worst case, just before the synchronization, in a single measurement for the three hops. Furthermore, the measurements and comparisons with the simulation for hop #2 and #3 make it possible to validate the implementation of the calculation of the *correction field* and thus the measurement of the residence time. We also observe that the dispersion of the precision values remains similar despite our pessimistic simulation of the PHY jitter.

These observations are validated by the calculation of the RMSE between the measured and simulated sliding average of the synchronization precision. Sliding average is computed over a window of 150 samples for the 500 samples preceding the PPS gap for each one of the three hops. Due to the progressive shift of the synchronization cycle relatively to the PPS measurement time, we are bound to use a small window. This small window isn't large enough to capture all possible variation in simulation. To compensate for this variability, we perform an RMSE calculation over 10 different simulations. Results are are presented in the table III. As shown in this table, the average RMSE is approximately 3 ns. There is also an increase in the median RMSE as the number of hops increases. This increase in the differences between reality and simulation is mainly caused by the pessimistic estimation of the PHY jitter, introduced during the calibration phase of the delay measurement mechanism.

Using the three figures, we also observe a large variation in the precision before synchronization as a function of the number of hops. This variation is due to the relative drift between the Grandmaster and the device in question. Indeed, before synchronization, the precision error is mainly caused by the drift which has taken effect since the last synchronization, here since 125ms. Before this measurement, we measured the relative drift between the Grandmaster and the different devices and observed a lower drift for switch 3 (hop #2) than for the other devices as shown in the table IV.

V. Conclusion

This paper presents the enhancement of an open source simulation library of the IEEE802.1AS synchronization protocol available here [12]. The integration of logical syntonization and real hardware inaccuracies brings the simulation library closer to reality. Our tests show the fidelity of the simulator after calibration compared to the result obtained with real TSN switches, as evidenced by the MSEs of the order of $4 * 10^{-6}$ % of occurrence between the distribution of values obtained by the simulated and measured link delay measurement mechanisms, as well as the RMSEs of approximately 3 ns between sliding average of the precision measured and simulated. This library now allows to study the precision of the synchronization and its impact on the clients according to parameters such as topology, protocol configuration, filters for measuring propagation delay and clock servo algorithm. In addition, we also propose a method which is repeatable to calibrate the simulator for other switches or end stations.

Future works will investigate the impact of other PHY layers like 1000Base-T and 10Base-T1S on precision using this library. This work will also allow us to focus on the calculation of the worst case precision to optimize the protocol parameters according to the intended use.

Acknowledgment

The authors thank all people and industrial partners involved in the EDEN project. This work is supported by the French Research Agency (ANR) and the partners of IRT Saint-Exupéry Scientific Cooperation Foundation: Airbus Operation, Airbus Defence and Space, CNES, Continental Automotive, INPT/IRIT, ISAE-SUPAERO, ONERA, Safran Electronics and Defense, Thales Alenia Space and Thales Avionics.

References

[1] "IEEE Standard for Local and Metropolitan Area Networks–Timing and Synchronization for Time-Sensitive Applications," *IEEE Std 802.1AS-2020 (Revision of IEEE Std 802.1AS-2011)*, pp. 1–421, 2020.

[2] "IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems," *IEEE Std 1588-2019 (Revision ofIEEE Std 1588-2008)*, pp. 1–499, 2020.

[3] G. M. Garner, A. Gelter, and M. J. Teener, "New simulation and test results for IEEE 802.1 AS timing performance," in *2009 International Symposium on Precision Clock Synchronization for Measurement, Control and Communication*. IEEE, 2009, pp. 1–7.

[4] M. Gutiérrez, W. Steiner, R. Dobrin, and S. Punnekkat, "Synchronization quality of IEEE 802.1 AS in large-scale industrial automation networks," in *2017 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2017, pp. 273–282.

[5] H.-T. Lim, D. Herrscher, L. Völker, and M. J. Waltl, "IEEE 802.1 AS time synchronization in a switched Ethernet based in-car network," in *2011 IEEE Vehicular Networking Conference (VNC)*. IEEE, 2011, pp. 147–154.

[6] H. Puttnies, P. Danielis, E. Janchivnyambuu, and D. Timmermann, "A Simulation Model of IEEE 802.1 AS gPTP for Clock Synchronization in OMNeT++." in *OMNeT++*, 2018, pp. 63–72.

[7] (2021) Omnet++ simulator version 5.2. [Online]. Available: https://omnetpp.org/

[8] (2021) Inet framework version 3.6.3. [Online]. Available: https://inet.omnetpp.org/

[9] W. Wallner, A. Wasicek, and R. Grosu, "A simulation framework for IEEE 1588," in *2016 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS)*. IEEE, 2016, pp. 1–6.

[10] P. Loschmidt, R. Exel, and G. Gaderer, "Highly accurate timestamping for ethernet-based clock synchronization," *Journal of Computer Networks and Communications*, vol. 2012, 2012.

[11] P. Loschmidt, "On enhanced clock synchronization performance through dedicated ethernet hardware support," Ph.D. dissertation, 2010.

[12] (2021) The simulation library. [Online]. Available: https://gitlab.amd.e-technik.uni-rostock.de/peter.danielis/gptp-implementation

Smart Management of Virtualized Network Service Chains in 5G Infrastructure

T. Djemai*, P. Berthou†, O. Gremillet*, A. Al Sheikh*, Y. Drif*, F. Arnal*, A. Bergaoui*
*IRT Lab, 3 rue Tarfaya, 31000 Toulouse, France
†LAAS-CNRS, 7 avenue Colonel Roche, 31000 Toulouse, France

***Abstract*—Future 5G infrastructures promise to deliver unprecedented Quality of Services (QoS) guarantees through ultra-low latency and high data rate specifications that lead to handle many critical applications and services.**

Network Function Virtualization (NFV) is the paradigm that enables the implementation of network functions and capabilities as software components executed in virtual entities provisioned in general-purpose hardware. This paradigm plays a pivotal role to achieve management flexibility of 5G services and reduce infrastructures' investment and operational costs.

Recent advances in artificial intelligence (AI) and the large amounts of data collected on orchestration platforms offer new perspectives. Many recent publications advocate the use of AI in the orchestration of virtualized networks and many algorithms are proposed. Though, today small amount of literature works implement or test these concepts in a real platform that considers network service chains data.

In this work, we present a framework for Service Function Chains' (SFCs) profiling and management through Machine-learning approaches. We propose an extended network orchestration platform based on Openstack, Kubernetes (K8s), and Open Source Mano (OSM) orchestrator. The benefits of this architecture are shown by an algorithm that realizes proactive auto-scaling procedure for service chains. It considers features' importance per service type while achieving a trade-off between services' stringent QoS requirements and the cost of resources usage.

***Index Terms*—Quality of Service, NFV, Adaptive network, Machine learning.**

I. Introduction

The infrastructure of 5th generation mobile networks is expected to offer distributed computation, storage and communication capacities with smart and flexible management services. This specificity will help to cope with stringent Quality of Service (QoS) requirements and support critical and ultra-reliable low-latency services such as autonomous vehicle's control and augmented reality applications.

Network Function Virtualization (NFV) [1] (along with Software Defined Network (SDN)) is an IT-based network paradigm aiming to replace the purpose-built proprietary network equipment with software network functions consolidated on commodity hardware. This paradigm eases the transition to a more agile and open service provisioning and brings flexibility of network services' operation/management. Moreover, the NFV allows to reuse efficiently the shared physical resources between services and handle the variation of services demand and theirs performance with much lower capital expenditure (CapEx) and operational expenditure (OpEx).

A survey conducted by [2] on Communications Service Providers (CSPs), identifies the NFV/SDN usage for 5G as the use case with the greatest revenue potential for the next few years.

However, the NFV usage in general and more specifically in 5G faces some issues that impact the commitment of CSPs for concrete NFV deployment.

It has been reported in the same study that half of the surveyed CSPs have virtualized less than 10% of their target network functions and that 77% of the virtualized functions do not use orchestration to automate their network and services. The orchestration procedure aims to plan the life cycle of the virtual functions and to control/manage their internal/external interactions.

We can summarize the NFV drawbacks as follows:

- **Operational complexity:** the operational complexity is considered as the biggest obstacle to an effective SDN/NFV deployment. Which is due to the distributed and heterogeneous aspect of the physical and the virtual infrastructure, the high dynamicity of workload and, the mobile aspect of users and services.
- **Services performance degradation and unsteady Virtual Network Functions (VNFs) behavior:** the NFV inherits the critical issue of IT virtualization reducing services processing capacities. Indeed, the virtualized network functions are easily impacted by various environmental factors such as workloads and resource contentions. This could lead to unsteady behavior and unpredictable performance degradation of the deployed VNFs. This drawback is amplified, if we consider and ordered combination of VNFs, known as Service Function chain (SFC). A Service Function chain is used to describe the definition and instantiation of an ordered or partially ordered set of virtual service functions, and the subsequent steering of traffic flows through those service functions.
- **Lack of smart orchestration tools and full NFV platforms:** It is highlighted that without orchestration, CSPs will not experience the levels of automation they want and will not be ready for the 5G market. The fact that today's implementations are often silo projects and not well integrated with existing operational support systems and SDN implementations.

We believe that one efficient approach to handle the potential VNFs/SFCs performance degradation and cope with the complexity of a NFV/5G environment shall pass by the usage of cognitive AI-based management systems combined with VNFs/SFCs profiling approaches as source data. The usage of

AI-based management will pave the way to future zero-touch network architectures.

However, current orchestration/management platforms do not support AI reasoning modules. Moreover, they can handle only simple VNFs and does not consider the SFCs related aspects such as traffic flow policies. A smart Management and orchestration (MANO) requires detailed information about Network Service (NS) behavior to react proactively and maintain its performances while manage efficiently its resources. This is why the integration of profiling and AI is necessary.

The contributions of this work can be summarized as follows:

- Extending the existing standardized OSM-based orchestration platform, Super-G, with a smart reasoning module.
- Extending the Super-G platform with Service function chain (SFC) related concepts.
- Proposing cognitive auto-scaling algorithm for SFCs.

The remainder of this paper is structured as follows: In section II, we present a state of the art on NFV management platforms and proposed AI-based scaling approaches for NFV/SFC. Section III presents the cognitive Super-G SFC platform architecture and functionalities. Section IV presents the proposed auto-scaling algorithm while the performance of this last are shown through experiments in V. Finally, VI presents some future works.

II. Related work

The following related work is divided in two parts. We first give an overview of existent platforms for NFV orchestration with their management services. Then, we focus on VNF/SFC elasticity problem through auto-scaling methods proposed in the literature.

A. SFC Orchestration and Management Platforms

The primary building blocks of an NFV platform are the Management and Orchestration (MANO) plane, the service plane, and the NFV Infrastructure (NFVI). In this work we focus on the MANO plane. The MANO plane offers a centralized control for service provision and management. The NFVI offers the pool of virtualized computation, storage and network resources that are managed by a cloud operating system such as Openstack. The service plane is a collection of VNFs that are combined to form an end to end service chain deployed in the NFVI and monitored via the MANO plane.

Numerous efforts have been devoted for the deployment of NFV orchestration platforms. Some of them follow the ETSI (European Telecommunications Standards Institute) standardized requirements while others are non-standardized propositions. Those platforms still in their infancy and are far away from proposing smart orchestration and management services. In accordance with work [3], we divide current MANO platforms into holistic platform that offers end to end deployment services and specific management issue oriented one.

Many of the existent holistic platforms focus only on the basic elements of the NFV architecture and do not consider more complex services like Service Function chains' management. In addition, the collected data are insufficient to study the behavior of SFCs. While there is a lack of holistic platforms that propose NFV/SFC profiling services, the design choice of the second category of MANO platforms is often based on distinct motivations and use cases such as scheduling, scaling, load-balancing, fail-over and can not be used for a full end to end deployment.

For the holistic MANO platform we can cite ETSI [4], OpenMANO [5], Open Baton [6], vConductor [7], TeNOR [8]. All of the cited technologies have the same elementary operation such as: manual creation and deployment of network services through graphic interface or cli. However, to the best of our knowledge none of them propose and AI-based module for SFC profiling and management. Our work is based on the ETSI Open Source MANO (OSM) platform that is extended with an AI-based SFC management module.

B. Service Function chains' Auto-Scaling Approaches

The management of Service Function Chains (SFCs) is a non trivial endeavor and in contrast with single VNF management, relatively few works try to tackle this problem. In this work we focus on the SFC scaling operation integrated within an AI-based extension of the ETSI OSM Platform. The proposed AI module can be used for other management tasks such as services' scheduling and placement.

VNFs scaling is a fundamental management task that helps tackling performance variations of VNFs/SFCs. In the literature, several approaches propose scaling mechanisms that differ in the utilized techniques. Works on VNF/SFC auto-scaling can be divided into reactive and proactive approaches. The reactive approaches consist on defining thresholds statically or dynamically to trigger the scaling (in/out) of services. The proactive scaling allows to scale VNFs resources based on their future behavior or on future system state (e.g. future workload) [9].

Most of the scaling work focuses on single network function data to scale one particular type of VNF such as Firewall (FW) or Deep Packet Inspection (DPI) functions.

Authors in [10] and [11] propose scaling algorithms based on static thresholds. The reactive approach, based on fixed values, allows performance degradation then trigger the scaling. This approach is not suitable for some applications of the 5G system that are considered as critical ones and can not afford any performance degradation. Moreover, the identification of the optimum threshold for scaling (in and out) is a non straightforward task and may depends on various parameters. Even if these approaches are simple to implement and do not cost a lot of computation resources, they may induce to services oscillating behavior affecting the overall system performances.

For the proactive approaches, work in [12] proposes an hybrid offline/online algorithm to forecast CPU usage based on historical data set for a time series model. While authors in [13] and [14] addresses the problem of managing VNF resources fluctuations by predicting resource requirements

using ML techniques and thereby enhancing the performance of the resource allocation algorithm. In the first part of their work, authors of [15] used the supervised learning approach and propose two neural-networks based on Multi-Layer Perceptron (a regressor and a classifier) to perform auto-scaling by predicting the required number of virtual network function instances based on the traffic demand. They focused on the User Plan Network Function (UPF) use case to identify the required number of UPFs to process incoming traffic in base stations with an objective to either maximize QoS or minimize cost.

We have noticed that most of the cited works focus only on data coming from one VNF and do not consider the impact of VNFs from the same chain or from other chains deployed on same nodes. In addition, almost every forecasting algorithm consider one dimension of the system at a time (such as CPU usage, or workload) but none has multi-decision criteria orientation. Furthermore, one of forecasting challenges in such dynamic and heterogeneous system is to define, according to the state of the system, on which level we should act. Which VNFs of the SFC should be scaled and which instances of the VNFs should be considered for that?

For works that consider SFCs, authors in [16] propose a method for finding optimal SFC path considering the resource utilization of virtual network function (VNF) and VNF placement.Work in [17] introduces a VNF resource prediction model that maximizes the benefits of using SFCs. The model exploits SFCs data to help predicting resource usage patterns of VNFs.

Authors in [18] define a set of critical performance features for each element of an NFV architecture. They introduce Probius, an analysis system. Probius collects most possible VNF performance related features, analyzes their behaviors and try to detect abnormal behaviors. The work in [19] proposes a dynamic auto-scaling algorithm called ElasticSFC. The algorithm aims to minimize the cost of resources usage while meeting the end-to-end latency of the service chain.

The second part of our work investigates the scaling decisions of the deployed Service Function chains (the moment of scaling, the number of replicas and the entities to scale) to ensure the stability of their performance and reduce the scaling and monitoring costs.

III. Super-G Architecture overview and Cognitive Management Framework

In this section, we present the initial platform developed within IRT labs [20] and its new functional components added within the AI-based management module.

The Super-G platform is a virtualized infrastructure deployed as a test-bed to study various subjects related to the integration of Satcoms with 5G networks.

Figure 1 depicts the high level architecture of the platform where the added AI-related modules are highlighted in green.

The physical infrastructure groups eight Dell servers and four HPGen compute nodes with 16 CPUs, 96 GB of RAM,

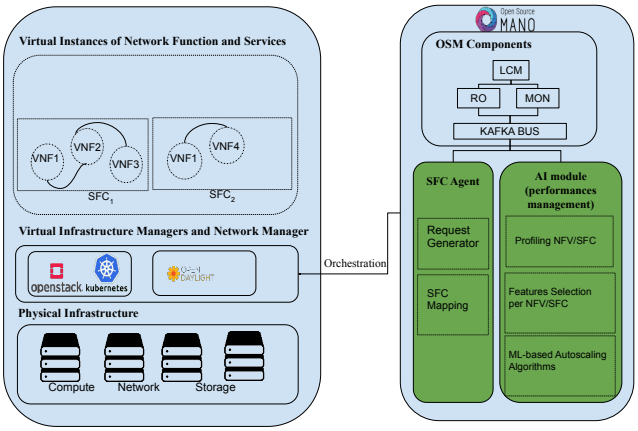


Fig. 1: High level view of functional architecture of Super-G platform.

and a shared storage of 34 TB. This infrastructure is aggregated behind a Proxmox hypervisor. On top of this last, we build Openstack private clouds and Kubernetes (K8s) clusters with their respective lightweight versions, microstack and microK8s (mK8s). Those technologies manage the physical resources allocated to the virtual instances of VNFs/SFCs. The platform has also an SDN controller for network resource management.

The orchestration part and high level management decisions are ensured by the OpenSource MANO (OSM) components. Relevant components for our work are: (1) the Life Cycle Manager (LCM) that is the core component of the tool and ensure inter operation with other modules, (2) the Resource Orchestrator (RO) that interacts with the Virtual Infrastructure Managers (VIMs) (openstack and K8s) for resources requests, and (3) the Monitoring collector (MON) that is responsible to get VIMs and VNF related metrics.

In order to handle SFCs related concepts, that are not supported by the OSM yet, we add the SFC agent that extends the Network Services concept and objects of OSM with SFCs specificity, such as client flow policies and Forwarding graph. In order to generate and transmit data to the profiling module, we use the SFC requests generator module.

The AI module has three main components: (1) the VNF/SFC Profiler, (2) the Feature Selections and, (3) the ML-based auto-scaling algorithms.

1) VNFs and SFCs Profiling: the VNF Profiling is the act of acquiring deep knowledge about the behavior of a VNF (resource consumption over time and performances metrics). The SFC profiling is more complex than just making the separate VNF profiling, as we need to establish the correlation between VNFs behaviors of the same chain. The main objective of the VNFs/SFCs profiling module is to cover the relationship between resources configuration, services demand and performances targets. This will help to estimate the optimum network load a VNF can support and estimate its resources (processor, memory and network) requirements to

meet its performance targets with ML techniques.

2) Data Collection, Metrics Granularity and Features Extraction: this module is in charge of collecting, pre-processing data, study relationship between various collected data and their relevance for specific VNF/SFC. It is also in charge of adjusting the granularity of the metrics according to the system state, in order to avoid unnecessary features and data collection. Preliminary experiments have shown that the computation, network and time overhead of the monitoring part is non negligible.

In this work, we consider two types of data: (1) Resource Usage (RU) metrics per VNF instance and VNF type, and (2) QoS metric per SFC. The Resource usage data collection interval is set according to the response time requirement of each SFC, while the QoS metric is collected whenever it is available. Then, data cleaning operations are applied in order to map the QoS values of an SFC with the resource usage of its VNFs.

3) ML-based Auto-scaling Algorithms: this module is in charge of studying various autoscaling algorithms (ML-based) and comparing their accuracy. It chooses the adequate model according to the context and the models are enhanced over time by online learning.

We have implemented a set of linear, nonlinear, ensemble and neural network based machine learning algorithms that were proven to be efficient and adequate for time series prediction problems. The implemented algorithms, well defined in work [21], are: (1) Passive Aggressive Regression (PA), (2) ElasticNet (EN), (3) RANdom SAmple Consensus (RANSAC), (4) Huber, (5) Ridge, (6) Lasso, (7) least-angle regression (LARS) (8) LassoLars (LLARS), (9) Long Short Term Memory (LSTM).

We consider a multivariate multi-step time series forecasting problem for QoS and Resource Usage (RU) predictions. Each method (except the LSTM) gives a set of prediction intervals (upper predicted values, lower predicted values and medium predicted one). Some modifications were necessary to adapt data for supervised learning models and to make multiple predictions.

There are three possible approaches to make multiple steps predictions in the literature: direct, recursive ,and multiple outputs. The first one involves developing a separate model for each forecast time step while the second one, used in models (1) to (8) of this work, involves making a prediction for one time step, taking the prediction, and feeding it into the model as an input to predict the subsequent time step. This process is repeated until the desired number of steps is reached.

The multiple outputs strategy, used in the LSTM model, involves developing one model that is capable of predicting the entire forecast sequence in a one-shot manner.

The Root Mean Squared Error (RMSE) is the metric used to compare the performance of the deployed algorithms.

The most efficient models were LSTM and GBR that we can briefly described as follows:

- Long short-term memory (LSTM) is a deep learning algorithm based on artificial recurrent neural network (RNN) architecture. Unlike standard feedforward neural networks, LSTM has feedback connections. LSTMs were developed to deal with the vanishing gradient problem that can be encountered when training traditional RNNs. Relative insensitivity to gap length is an advantage of LSTM over RNNs, hidden Markov models and other sequence learning methods in numerous applications.
- Gradient Boosting Regressor (GBR): Gradient boosting is a machine learning technique used in regression and classification tasks, among others. It gives a prediction model in the form of an ensemble of weak prediction models (decision trees). A gradient-boosted trees model is built in a stage-wise fashion as in other boosting methods, but it generalizes the other methods by allowing optimization of an arbitrary differentiable loss function.

More details about data preparation, models parameters selection, model performance comparison and their online training will be given in the experiment section V.

IV. SFCs Elasticity through ML-based Auto-scaling in Super-G

In this section, we present the mathematical formulation of the SFC system, the considered problem and the proposed resolution algorithm, Super-G Autoscaling Algorithm (SAA).

A. SFC System model and Problem statement

1) A Service Function Chain: At each time step t, a service function chain, SFC_i, is represented by $\mathbb{V}_i$, the set of its VNFs types and their data dependency $\mathbb{DP}_i^t$.

Each SFC_i has a QoS requirement vector $\vec{QoS}_{it}$ presenting the QoS metrics and their objectives values at instant time t, as depicted by Eq (1).

This work considers only the maximum response time metric in seconds (s) RT_i^{max} and the effective RT_i^t which is the response time of the last processing flow in seconds (s) (data flow from the user equipment to its final application and its callback response).

$$\vec{QoS}_{it} =< (RT_i^{max}, RT_i^t) > \tag{1}$$

From the response requirement we establish a priority between SFCs. Each SFC has a priority γ_i, where the most prior SFC is the one with the biggest value and is computed according to Eq (2) as follows:

$$\gamma_i = \frac{1}{RT_i^{max}} \tag{2}$$

According to the applications' functional sensitivity, we define two type of SFCs: (1) critical context SFCs, that serves application related to safety and security and (2) non-critical context SFCs that fall in all other areas such as advertisement and gaming.

We define the binary variable δ_i, presented by Eq (3) as follows:

$$\delta_i = \begin{cases} 0, & \text{non-critical context } SFC_i \\ 1, & \text{critical context } SFC_i \end{cases} \tag{3}$$

2) A VNF and a VNF instance: A VNF is the intermediary computation entity that has input data, make some functional processing and then sends its results to the next VNF, the final application or the end user (UE).

A VNF has a type p that identifies its functional behavior. Some examples can be given such as Firewall and Deep Packet inspection services.

A VNF_i^p has $\mathbb{V}_i^p$, its set of VNF instances at time t, where each element VNF_{ij}^{tp} represents the j^{th} instance of the VNF of type p from the i^{th} SFC in the system.

Each VNF instance is identified by its maximum and effective resources consumption (network and computation) vector $\vec{RU}_{ijpt}$ as described in Eq (4), and a set of their predecessors VNFs $\mathbb{D}_{ijp}$ within SFC_i.

A resource usage vector at instant time t gives us the processing (CPU in m), memory (RAM in MB) and bandwidth (BW in MBps) usage of the VNF.

$$\vec{RU}_{ijpt} =< (CPU_{ijp}^{max}, CPU_{ijp}^{t}), (RAM_{ijp}^{max}, RAM_{ijp}^{t}), (BW_{ijp}^{max}, BW_{ijp}^{t}) > \quad (4)$$

3) SFC autoscaling problem: Given a time slotted system, we consider a set of deployed Service Function Chains $\mathbb{S}$ and their response time requirements.

We define $AD_{t+\tau}(SFC_i)$, the SFC autoscaling decision from instant time t to $t+\tau$, as a list of pairs $\{(VNF_i^p, nb_{ip})\}$, where VNF_i^p is the VNF type to be scaled and nb_{ip} is the number of replicas to be created.

This work aims to provide a set of SFCs autoscaling decisions for the next τ time steps to anticipate SFCs' QoS degradation and maintain their response time performances.

Before explaining the proposed scaling procedure we define some prediction related variables.

We define β_i, presented in Eq (5), as the data collection interval of an SFC_i.

$$\beta_i = \frac{RT_i^{max}}{\mathbf{card}(\mathbb{V}_i) + \mathbf{card}(\mathbb{DP}_i)} \quad (5)$$

The τ defines the prediction horizon in seconds (s).

The variable ζ represents the number of QoS predictions as is computed according to Eq (6).

$$\zeta_i = \frac{\tau}{\beta_i} \quad (6)$$

A QoS prediction related to SFC_i is the set of predicted response time within the time period τ, as defined by Eq (7)

$$\mathbb{PR}_i^{\tau}(QoS) = \{\hat{\vec{QoS}}_{ik}\}_{k\in[t,t+\tau]} = \{\hat{RT}_{ik}\}_{k\in[t,t+\tau]} \quad (7)$$

Similarly for each $VNF_{ij}^p \in \mathbb{V}_i^p$ a RU prediction is a set of predicted resources usage vectors that is defined by Eq (8).

$$\mathbb{PR}_{ijp}^{\tau}(RU) = \{\hat{\vec{RU}}_{ijpk}\}_{k\in[t,t+\tau]} \quad (8)$$

The RU prediction per VNF type, as defined by Eq (9), is the average value for each resource prediction at each time step between all the instances of VNF_i^p

$$\mathbb{PR}_{ip}^{\tau}(RU) = avg_{k\in[t,t+\tau]}(\{\hat{\vec{RU}}_{ijpk}\}_{k\in[t,t+\tau]}) \quad (9)$$

A resources usage prediction for an SFC, $\mathbb{PR}_i^{\tau}(RU)$,is the set of the resources usage prediction for each of its VNF type

B. Super-G Auto-scaling algorithm for SFCs (SAA)

The SAA algorithm, described in Algorithm 1, is divided into two main procedures: (1) the Prediction_procedure and (2) the VNFs_selection_replicas_procedure, described respectively in Algorithms 2 and 3.

The SAA is continuously executed and it takes as input the set of current deployed SFCs, their QoS requirements and the prediction horizon τ (the duration of prediction time).

At each iteration the prediction procedure is called for all the SFCs and it returns a set of QoS predictions. If the SFC_i is non-critical and it its is possible, it returns the lower predicted values. Otherwise it returns the upper predicted values.

The resultant predictions are then given as input to the VNFs_selection_replicas_procedure to: (1) identify SFCs that need scaling operations and (2) choose the VNFs to be scaled and their number of replicas.

A SFC is identified as a SFC to be scaled if one of its predicted QoS values is greater than its maximum tolerated response time.

A scaling ratio (SR) is computed for each SFC to be scaled. This last is computed as the ratio between the maximum tolerated value and the maximum predicted value.

This choice aims to reduce the number of scaling operations while anticipating future QoS degradation as early as possible.

In order to avoid resources usage inefficiency (over-usage of resources), the procedure does not scale all the VNFs of the SFCs. But instead, it gets the resource usage predictions of each VNFs of the SFCs that need to be scaled, and then select only the VNFs that have increasing series of resource usage values and their successors VNFs (to avoid the bottleneck sliding effect). Finally, the number of VNFs replicas (RN) is defined by the previously computed scaling ratio of their SFCs.

After executed the previous procedure, the SAA sends the scaling decision to the VIM agents and check if there are new SFCs and add them to the SFCs set.

We would like to draw the attention of the reader that for the online prediction aspect, we have a background procedure that evaluates the performances after each prediction iteration for all the models) and compares their average RMSE and it returns the best model to use.

All the model except LSTM are trained in online fashion and maybe updated (refit and retest) if after three consecutive prediction iterations their RMSE increases and this augmentation is greater than 30%.

V. EXPERIMENTS

This section aims to prove the benefits of using the proposed proactive auto-scaling algorithm (SAA) against the legacy K8s

Data: $\mathbb{S}$ (SFCs list), τ (prediction horizon) $\bigcup_{i=1}^{\mathbf{card}(\mathbb{S})}\{RT_i^{max}\}$ (SFCs' response time requirements list);
compute_priorities($\mathbb{S}, \bigcup_{i=1}^{\mathbf{card}(\mathbb{S})}\{RT_i^{max}\}$);
sort_by_decreasing_priority($\mathbb{S}$);
while *True* **do**
  $\mathbb{PR}^{\tau} \leftarrow \varnothing$;
  for $SFC_i \in \mathbb{S}$ **do**
    $\mathbb{PR}_i^{\tau}$ = Prediction_procedure(QoS, SFC_i, τ);
    $\mathbb{PR}^{\tau} \leftarrow \mathbb{PR}^{\tau} \cup \{\mathbb{PR}_i^{\tau}\}$;
  end
  $\mathbb{AD}_{t+\tau}$ = VNFs_selection_replicas_procedure$(\mathbb{S}, \mathbb{PR}^{\tau})$;
  send_decisions_to_VIMs($\mathbb{AD}_{t+\tau}$, @VIMs);
  update_SFCs_set(@VIMS)
end

Algorithm 1: Auto-scaling main procedure

Data: Pr_{type}, SFC_i, τ (Prediction horizon)
Result: List of QoS predictions and list or RU predictions
initialization;
if $Pr_{type} = QoS$ **then**
  $model_{QoS} \leftarrow$ best previous prediction model() ;
  $\mathbb{PR}_i^{\tau} \leftarrow model_{QoS}$.predict$(SFC_i, \tau)$;
  return $\mathbb{PR}_i^{\tau}(QoS)$;
else
  $\mathbb{PR}_i^{\tau}(RU) \leftarrow \varnothing$
  $model_{RU} \leftarrow$ best previous prediction model() ;
  for $VNF_i^p \in \mathbb{V}_i$ **do**
    $\mathbb{PR}_{ip}^{\tau} \leftarrow \varnothing$;
    for $VNF_{ij}^p \in \mathbb{V}_i^p$ **do**
      $\mathbb{PR}_{ijp}^{\tau} \leftarrow model_{RU}$.predict$(VNF_{ij}^p, \tau)$;
      $\mathbb{PR}_{ip}^{\tau} \leftarrow \mathbb{PR}_{ip}^{\tau} \cup \mathbb{PR}_{ijp}^{\tau}$
    end
    $\mathbb{PR}_{ip}^{\tau} = avg_{k\in[t,t+\tau]}(\{\hat{\vec{RU}}_{ijpk}\}_{k\in[t,t+\tau]})$
    $\{\mathbb{PR}_i^{\tau}(RU)\} \leftarrow \{\mathbb{PR}_i^{\tau}(RU)\} \cup \mathbb{PR}_{ip}^{\tau}$
  end
  return $\mathbb{PR}_i^{\tau}(RU)$
end

Algorithm 2: Prediction_procedure

HPA algorithm in the case of critical applications deployed within a K8s-based NFV infrastructure.

We first start by showing the weakness of the legacy not tuned K8s HPA with a simple experiment and its impact on services response time degradation and resource usage inefficiency. In the second part, we expose some results related to the behavior of the chosen machine learning algorithms. In the last part, we present the comparison results of the SAA and the K8s HPA algorithms.

Data: List of QoS predictions, list of SFCs, τ (prediction horizon)
Result: List of Auto-scaling decision $\mathbb{AD}_{t+\tau}$
$\mathbb{S}_{toScale} \leftarrow \varnothing$ set of pairs SFC to scale and its scaling ratio (SR);
$\mathbb{V}_{toScale} \leftarrow \varnothing$ set of pairs VNF to scale and its number of replicas (RN);
$\mathbb{AD}_{t+\tau} \leftarrow \varnothing$ set of scaling decisions ;
for $SFC_i \in \mathbb{S}$ **do**
  $pr_i^{max} \leftarrow \max_{\{pr_i^t \in \mathbb{PR}_i^{\tau}\}}(pr_i^t)$
  if $pr_i^{max} > RT_i^{max}$ **then**
    $SR_i \leftarrow \frac{pr_i^{max}}{RT_i^{max}}$;
    $\mathbb{S}_{toScale} \leftarrow \{(SFC_i, SR_i)\} \cup \mathbb{S}_{toScale}$;
  else
    continue;
  end
end
for $(SFC_i, SR_i) \in \mathbb{S}_{toScale}$ **do**
  $\mathbb{PR}_i^{\tau}(RU) \leftarrow$ prediction_procedure(RU, SFC_i, τ)
  for $VNF_i^p \in \mathbb{V}_i$ **do**
    if *(increase_serie($\mathbb{PR}_i^p$)) and* $(VNF_i^p \notin \mathbb{V}_{toScale})$ **then**
      $RN_i^p \leftarrow \lceil (SR_i * \mathbf{card}(\mathbb{V}_{ip}^t)) \rceil + \mathbf{card}(\mathbb{V}_{ip}^t)$;
      $\mathbb{V}_{toScale} \leftarrow \mathbb{V}_{toScale} \cup \{(VNF_i^p, RN_i^p)\} \bigcup_{VNF_i^l \in \mathbb{D}_{ip}} \{(VNF_i^l, RN_i^p)\}$;
    else
      continue;
    end
  end
  $AD_{t+\tau}(SFC_i) \leftarrow \mathbb{V}_{toScale}$;
  $\mathbb{AD}_{t+\tau} \leftarrow \mathbb{AD}_{t+\tau} \cup AD_{t+\tau}(SFC_i)$;
end
return $\mathbb{AD}_{t+\tau}$;

Algorithm 3: VNFs_selection_replicas_procedure

A. Limits of the HPA K8s algorithm if not used in a proactive fashion

In this section, we motivate the need of an ML-based autoscaling algorithm by focusing on existent algorithms and their failure to efficiently support an NFV/SFC context. More specifically, we focus on the Horizontal Pod Auto-scaling algorithm (K8s HP) of the Kubernetes platform. We present some preliminary results of a baseline scenario. The K8s HPA algorithm allows services to be scaled (replicated) based on user predefined values of computation resources usage such as the processor and the memory.

In our test, we consider one service function (php-apache server) deployed in microK8s cluster and one client (load generator) that queries the server. We collect CPU usage data during 20 minutes in the K8s cluster for both no scaling policy and using K8s HPA autoscaling algorithm with a CPU usage limit set to 20% and a maximum number of replicas set to 4. We also measure the average user experienced response time

for both no scale and HPA scenarios. The workload was sent from 15:30 to 15:50 to a not scaled service ad then from 16:00 to 16:20 when the HPA algorithm was enabled.

Figures 2a and 2b show CPU usage of the microK8s cluster for both no scaling and HPA scenarios. As expected, the CPU usage after scaling increased from 17.56% to 22.39%. We notice also that the monitoring part for CPU consumption is not negligible. However, from figure 2c that shows CPU usage of one of the service replicas, we can see that the replica is underutilized (so as the two others). We have noticed that replicas are created and not scaled down even if they are not used for a certain amount of time. We can conclude that there is an inefficient usage of resources with the K8s HPA (resource are overused).

For the QoS aspect of the service, experiments show that the time between creation of replicas and their ready to use state is non negligible and during this period the user experience response time is being negatively impacted. Indeed, we have noticed that the average response time of service without replicas was around 55ms while with replicas it decreases to 45ms. However,for the first minutes before all replicas are ready the response slightly increases to 58ms. Those few ms of degradation can not be tolerated if we consider critical services such as autonomous driving.

Moreover, alongside with the absence of network related policy for autoscaling in the HPA, the definition of the optimal values of CPU and memory to trigger the scale is let to the administrator and need to be thoroughly studied.

B. Experiments description

1) The SFCs and the infrastructure: We emulate the behavior of two time-critical applications, considered as pivot use cases within the 5G community: (1) the telesurgical application (similarly to remote vehicle control) has stringent response time requirements and is considered as a context critical application, (2) the augmented reality application for advertising (ARA), which is a time-sensitive application and is considered as non context-critical one. Both applications run in the same VIM and share the same physical resources.

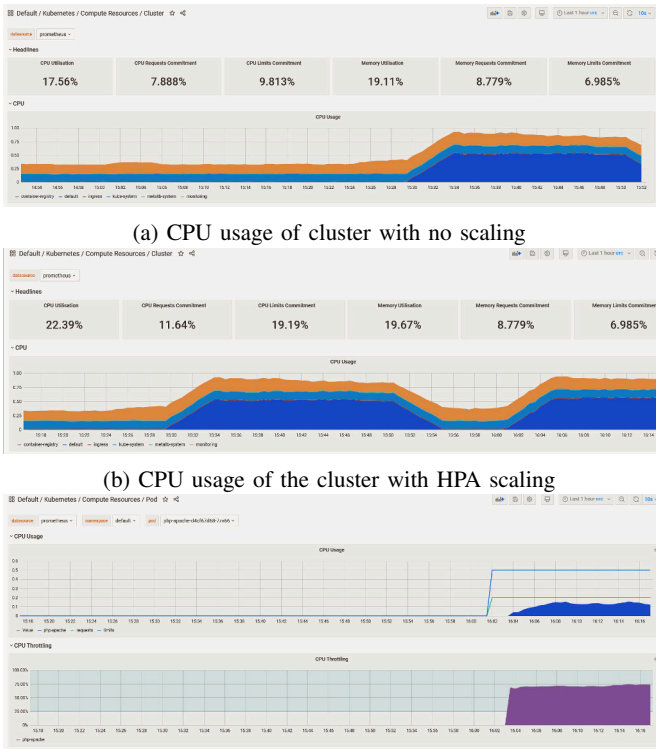
Figure 3 (a) shows the communication schema between elements of the telesurgical SFC (SFC_1).

Base on the defined telesurgical application in work [22], we model SFC_1 as a diamond shaped diagram, where VNF_1^3 and VNF_1^3 represent respectively the VNF types of management agents of the robot surgery tool and the 3D camera. The VNF_1^2 is the master decision entity that interacts directly with the surgeon.

We consider VNF_1^3 as RAM intensive as it should get access as fast as possible to the instructions and VNF_1^1 as CPU intensive procedure as it should process 3D image data. The VNF_2 is both CPU and RAM intensive procedure.

According to work [23], the maximum allowed latency for the real-time multimedia system part of the telesurgey application should fall beneath $RT_1^{max} = 150ms$.

Figure 3 (b) shows the communication schema between elements of the augmented reality advertisement (SFC_2).

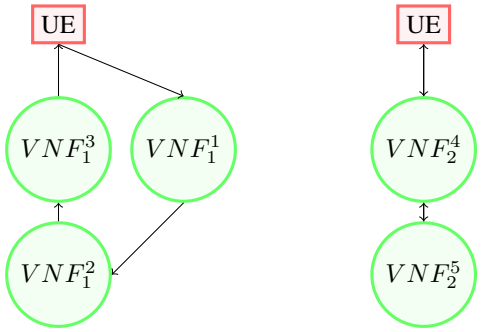


(a) CPU usage of cluster with no scaling

(b) CPU usage of the cluster with HPA scaling

(c) CPU usage of one service replica

Fig. 2: Time series data of CPU usage for no scaling and HPA scaling between 15:30 and 16:15.



(a) SFC_1 (b) SFC_2

Fig. 3: SFCs communication diagrams with UEs for (a) Telesurgery, (b) Augmented Reality Advertisement

It is composed of VNF_2^4 which is responsible of video image preprocessing and GPS tracking and VNF_2^5 which is responsible of augmented reality object creation, management decisions, and image processing.

The maximum latency for this service should fall beneath $RT_2^{max} = 500mss$.

In order to simulated the 5G communication between VNFs each communication link latency should be beneath 150ms.

The VIM infrastructure is managed by K8s agent enhanced by our algorithms. The SFCs are instantiated as argo workflows [24]. Each SFC has its namespace. A VNF type is a K8s

deployment and a VNF instance is represented by one pod of the deployment.

The VNFs do not have the functional instruction of the applications but are considered as black boxes with input and output data. In order to mimic the functional time, we consider CPU and RAM intensive programs (within the VNFs) that will stress those resources for a certain amount of time, chosen uniformly in a given interval of time at each data flow round.

For a time period of 24 hours, the SFC_1 and SFC_2 are stressed and stopped.

The User Equipment (UE) of SFC_1 (telesurgery) sends one packet of data every 0.01 seconds until the packet counter reach's n_1 packets (a value taken uniformly within $[10, 5000]$ and then it stops for time period of $rest_1$, a value taken uniformly within $[1ms, 30ms]$.

Similarly the UE of SFC_2 (ARA) sends one packet of data every 0.05 seconds until the packet counter reach's n_2 packets (a value taken uniformly within $[10, 5000]$ and then it stop for time period of $rest_2$, a value taken uniformly within $[1ms, 30ms]$. The shown results are the average results of 5 runs.

C. The collected Data

The dataset utilized in this work is generated for a period of 24 hours. For each SFC_i, the traces in the dataset are in the form of $(\vec{RU}_i^t, QoS_i^t)$ and we interpret this time series as a set of samples $\{(\vec{RU}_i^1, QoS_i^1)), (\vec{RU}_i^2, QoS_i^2)), ..., (\vec{RU}_i^n, QoS_i^n))\}$.

To avoid unnecessary data collection and storage, the resource usage traces are collected on a timescale relative to their SFC QoS requirement and defined by the β_i described in section IV. We consider a lag of $10mn$ for each prediction as the preliminary tests have shown that this value allows the algorithms to perform at their best.

D. The learning models

The implemented learning algorithms were tuned in order to make multiple output predictions and time prediction intervals. The models give upper, lower and middle values (except the LSTM). According to the critical-context status of the SFC, we will prefer to use the upper, middle or the lower values set.

After each period τ all the models (expect the LSTM) are being tested. If their average performance goes below 30% of the two previous predictions, they are retrained and refitted in background from the main algorithm.

Keras and TensorFlow are the library used for the LSTM algorithm and the scikit-learn library is used for the rest of the described algorithms.

Table I gives the used parameters for each model (chosen under experiments).

E. Train and Test sets

A walk forward validation is used to backtest the ML models. This approach gives the models the opportunity to make good forecasts for the next time step as new data are available to retrain.

F. Compared methods and evaluation criteria

The SAA and the K8s HPA algorithms are compared according to the SFCs' response time in seconds (s).

The K8s HPA algorithm has multiple possibilities of parametrization to trigger a scaling operation, and this last is left to the user.

In order to make a fair comparison with the SAA, our version of the K8s HPA that considers the response time of the SFCs can be described as follows:

- The QoS metrics are checked every β_i seconds and are compared to the defined QoS limit values for each SFC.
- If the collected QoS value is greater than the QoS limit, then all the VNFs of the related SFC are scaled with one replica.

G. Results analysis

1) Pre-selection of the prediction model for the SAA: In this section, we present some results related to the training procedure.

Figure 4a shows the RMSE values comparison for the ML algorithms for 6 consecutive predictions. The experiments show that the GBR is the best algorithm followed by the LSTM.

We notice that the RMSE remains quite stable even if it increases slightly when the number of prediction steps increases. Even if it performs better than other methods, the LSTM remains less efficient than the GBR. This could be caused by the amount of data used for training that remains insufficient for a deep learning procedure.

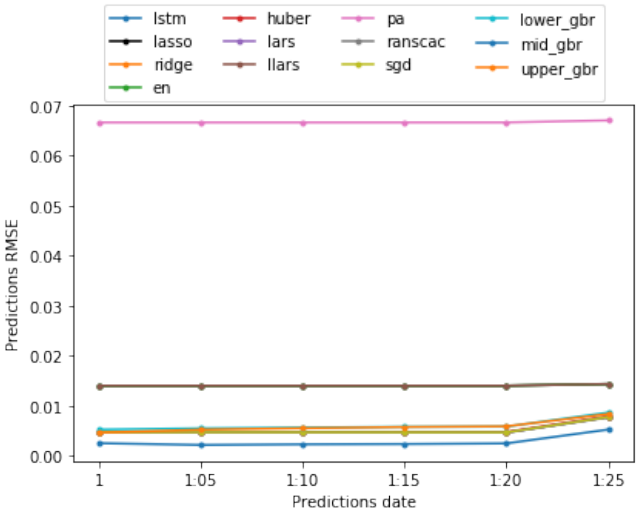
For the time aspect, we have noticed that one prediction of the most time consuming algorithm takes an average time of 0.000336 (s). Training and fitting of all algorithm (except LSTM which takes 3mn to 5mn) takes in average 13 (s).

Figure 4b shows a sample of the QoS (response time in seconds) prediction of the SFC_1 by the GBR method. The blue data presents the history data while the red line represents the QoS limit of the SFC which was set to 0.075 (s). We notice that the effective values (the green line) are included within the prediction interval of the GBR.

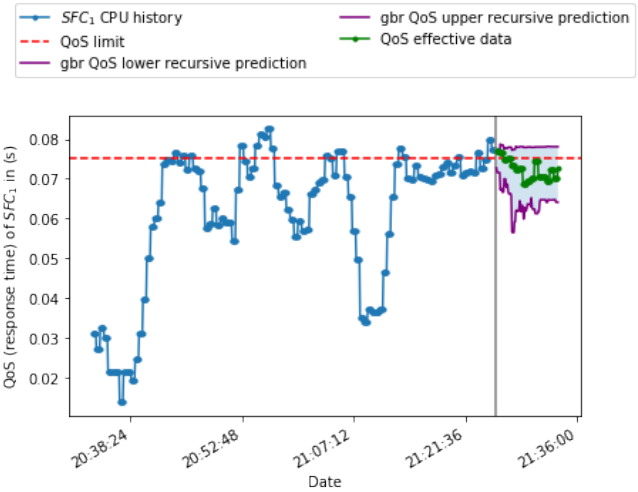
2) Comparison of K8s HPA and the SAA methods: Figure 5 show the response time of the SFC_1 over time when both K8s HPA and SAA algorithms are used. As a general behavior, we can notice that the SAA maintains the response time under the maximum response time requirement of the SFC_1. However, in some cases the threshold is crossed, which is due to the prediction accuracy. We can also notice that over all the time period, the response time with the SAA is beneath the values provided by the K8s HPA and which could be due to the overpovisioning of replicas when using the worst case prediction.

VI. Conclusion and Future Work

This work shows the importance of an AI-based orchestration of services in 5G context that will handle critical applications with stringent QoS requirements such as autonomous driving, health and augmented reality applications. We focus



(a) Performance comparison of learning algorithm through the average RMSE



(b) QoS prediction of the GBR model for $SFC1$

Fig. 4: Average ML algorithm performance and QoS prediction interval

TABLE I: Table of models parameters.

Model	**Parameters**
PA Regressor	$max_{it} = 1000, tol = 1e-3$
SGD Regressor	$max_{it} = 1000, tol = 1e-3$
GB Regressor (Upper/Lower/mid)	$loss = quantile, alpha = \{0.9, 0.2, 0.5\}$
LSTM	$neurons = 50, loss = mae, optimizer = adam$ $epochs = 50, batch\_size = 72, shuffle = False$

on the autoscaling operation of services in order to manage their unstable behavior and maintain their performance over time within a dynamic and uncertain network environment.

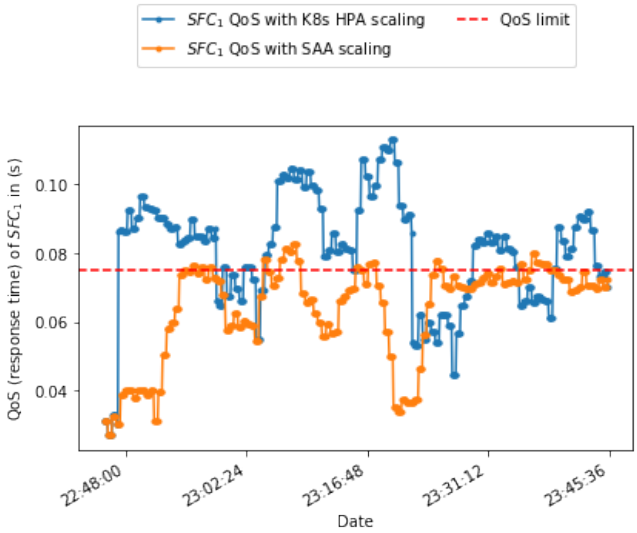


Fig. 5: Performance comparison of learning algorithm through the average RMSE

We compared the proposed SFC autoscaling algorithm (SAA) with legacy K8s HPA algorithm and have experimented its lacks of features to support the new generation of 5G critical services requirements. Results have shown that our method enhances the average response time of the tested SFCs by 26% compared to the legacy K8s HPA.

As future work, we plan to consider the vertical autoscaling operation, that allows to scale specific resources of a VNF. The scaling decision will be more complex, as it will have to choose which scale approach to use according to the context and which resource to scale. As in edge computing paradigm computation resource are limited, we must consider the problem of scaling and resource usage efficiency trade-off over time. We will also work to enhance our library of ML methods by adding and test other literature learning approaches, and track their monitoring and training cost (time and resource) with more SFCs graph types.

As a long term work, we foresee to conduct a through study on the size of the prediction window, as this last will play a pivotal role to anticipate QoS degradation that should be made as early as possible.

ACKNOWLEDGMENT

This work is supported by the Technological Research Institute IRT Saint Exupery, CS34436, 3 Rue Tarfaya, 31400 Toulouse, France.

REFERENCES

1 ETSI, "Network Function Virtualization (NFV); Terminology for Main Concepts in NFV," European Telecommunications Standards Institute, Tech. Rep., 2013.
2 Crawshaw, J., "SDN/NFV Pulse of the Industry: Commercial Realities," NetCracker Company, Tech. Rep., 2019.
3 Sousa, N., Perez, D., Rosa, R., Santos, M., and Esteve Rothenberg, C., "Network service orchestration: A survey," *Computer Communications*, vol. 142, 03 2018.

4 Mechtri, M., Ghribi, C., Soualah, O., and Zeghlache, D., "Nfv orchestration framework addressing sfc challenges," *IEEE Communications Magazine*, vol. 55, no. 6, pp. 16–23, 2017.

5 Mijumbi, R., Serrat, J., Gorricho, J.-L., Latré, S., Charalambides, M., and López, D., "Management and orchestration challenges in network functions virtualization," *IEEE Communications Magazine*, vol. 54, pp. 98–105, 2016.

6 Baton, O. Open baton: An extensible and customizable nfv mano-compliant framework. [Online]. Available: https://openbaton.github.io

7 Shen, W., Yoshida, M., Kawabata, T., Minato, K., and Imajuku, W., "vconductor: An nfv management solution for realizing end-to-end virtual network services," in *The 16th Asia-Pacific Network Operations and Management Symposium*, 2014, pp. 1–6.

8 Riera, J. F., Batallé, J., Bonnet, J., Días, M., McGrath, M., Petralia, G., Liberati, F., Giuseppi, A., Pietrabissa, A., Ceselli, A., Petrini, A., Trubian, M., Papadimitrou, P., Dietrich, D., Ramos, A., Melián, J., Xilouris, G., Kourtis, A., Kourtis, T., and Markakis, E. K., "Tenor: Steps towards an orchestration platform for multi-pop nfv deployment," in *2016 IEEE NetSoft Conference and Workshops (NetSoft)*, 2016, pp. 243–250.

9 Pande, M. Aws scaling (reactive vs proactive vs predictive). [Online]. Available: https://www.gritfeat.com/aws-scaling-reactive-vs-proactive-vs-predictive/

10 Dutta, S., Taleb, T., Frangoudis, P. A., and Ksentini, A., "On-the-fly qoe-aware transcoding in the mobile edge," in *2016 IEEE Global Communications Conference (GLOBECOM)*, 2016, pp. 1–6.

11 Carella, G. A., Pauls, M., Grebe, L., and Magedanz, T., "An extensible autoscaling engine (ae) for software-based network functions," in *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, 2016, pp. 219–225.

12 Bilal, A., Tarik, T., Vajda, A., and Miloud, B., "Dynamic cloud resource scheduling in virtualized 5g mobile systems," in *2016 IEEE Global Communications Conference (GLOBECOM)*, 2016, pp. 1–6.

13 Mijumbi, R., Hasija, S., Davy, S., Davy, A., Jennings, B., and Boutaba, R., "Topology-aware prediction of virtual network function resource requirements," *IEEE Transactions on Network and Service Management*, vol. 14, no. 1, pp. 106–120, 2017.

14 Mestres, A., Rodriguez-Natal, A., Carner, J., Barlet-Ros, P., Alarcón, E., Solé-Simó, M., Muntés-Mulero, V., Meyer, D., Barkai, S., Hibbett, M., Estrada, G., Coras, F., Ermagan, V., Latapie, H., Cassar, C., Evans, J., Maino, F., Walrand, J., and Cabellos, A., "Knowledge-defined networking," *ACM SIGCOMM Computer Communication Review*, vol. 47, 06 2016.

15 Subramanya, T., Harutyunyan, D., and Riggio, R., "Machine learning-driven service function chain placement and scaling in mec-enabled 5g networks," *Computer Networks*, vol. 166, p. 106980, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128619310254

16 Lee, D., Yoo, J.-H., and Hong, J. W.-K., "Q-learning based service function chaining using vnf resource-aware reward model," in *2020 21st Asia-Pacific Network Operations and Management Symposium (APNOMS)*, 2020, pp. 279–282.

17 Kim, H.-G., Jeong, S.-Y., Lee, D.-Y., Choi, H., Yoo, J.-H., and Hong, J. W.-K., "A deep learning approach to vnf resource prediction using correlation between vnfs," in *2019 IEEE Conference on Network Softwarization (NetSoft)*, 2019, pp. 444–449.

18 Nam, J., Seo, J., and Seungwon, S., "Probius: Automated approach for vnf and service chain analysis in software-defined nfv," 03 2018, pp. 1–13.

19 Nadjaran Toosi, A., Son, J., Chi, Q., and Buyya, R., "Elasticsfc: Auto-scaling techniques for elastic service function chaining in network functions virtualization-based clouds," *Journal of Systems and Software*, vol. 152, pp. 108–119, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121219300421

20 Institute, T. R. Accelerating science, technology research & transfers to industry. [Online]. Available: https://www.irt-saintexupery.com/

21 Sarker, I., "Machine learning: Algorithms, real-world applications and research directions," *SN Computer Science*, vol. 2, p. 160, 02 2021.

22 Thompson, J., Ottensmeyer, M., and Sheridan, T., "Human factors in telesurgery: Effects of time delay and asynchrony in video and control feedback with local manipulative assistance," *Telemedicine journal : the official journal of the American Telemedicine Association*, vol. 5, pp. 129–37, 02 1999.

23 Zhang, Q., Liu, J., and Zhao, G., "Towards 5g enabled tactile robotic telesurgery," *ArXiv*, vol. abs/1803.03586, 2018.

24 "Argo workflows - the workflow engine for kubernetes," https://argoproj.github.io/argo-workflows/fields/, accessed: 2022-01-01.

Session Th.1.A

AI: Assurance & Testing II

Thursday 2nd June

10:00

–

Amphithéâtre

A testing approach for dependable Machine Learning systems

Cyril Cappi[(1)], Camille Chapdelaine[(2,3)], Laurent Gardes[(1)], Eric Jenn[(3,4)], Baptiste Lefevre[(4)], Sylvaine Picard[(2)], Thomas Soumarmon[(3,5)]

[(1)] SNCF, Saint-Denis, [(2)] Safran Tech, Magny-les-Hameaux, [(3)] IRT Saint-Exupéry, Toulouse, [(4)] THALES AVS, Toulouse and Mérignac, [(5)] Continental, Toulouse

***Abstract* — In order to be used into a critical system, a software or an hardware component must come with strong evidences that the designer's intents have been correctly captured and implemented. This activity is already complex and expensive for classical systems despite a very large corpus of verification methods and tools. But it is even more complicated for systems embedding Machine Learning (ML) algorithms due to the very nature of the functions being implemented using ML and the very nature of the ML algorithms. This paper focuses on one specific verification technique, testing, for which we propose a four-pronged approach combining performance testing, robustness testing, worst-case testing, and bias testing.**

Keywords— machine learning, testing, performance, robustness, worst-case, bias, safety

I. MOTIVATION AND OBJECTIVES

How to engineer mission- or safety-critical systems embedding Machine-Learning (ML) is a very hot topic raising many challenges, in particular concerning verification, validation, and certification activities [1] [2] [3]. Formal verification techniques are making their way in the domain of Artificial Intelligence (AI) and ML [4], but their applicability is still limited to specific use cases and specific properties (e.g., properties around a specific input point). As of today, verification of ML system essentially relies on the only verification technique applicable on a large scale: testing. Hence, to reach the required dependability level, a rigorous testing strategy is mandatory [5]. This is particularly true for perception systems (e.g. computer vision, natural language recognition) where the dimension of the input space is huge, making the efficiency of testing highly arguable if no appropriate strategy is defined. Testing ML implies to revisit classical components of testing, such as the definition of equivalence classes or the definition of test oracles.

From these considerations, the goals of this paper are the followings:

- propose a testing strategy for systems of perception based on ML
- identify the different categories of tests supporting this strategy
- consider how test results can be used to improve the performances of the systems.

Towards those goals, we first briefly describe the target system chosen to illustrate our approach, which is a railway signal detection system based on image recognition (Section II). Then, we introduce the generic elements that constitute a test and detail our approach (Section III). In the following sections, we successively consider four categories of tests: *performance testing* (Section IV), *robustness testing* (Section V), *worst-case testing* (Section VI), and *bias testing* (Section VII). Finally, we review related works in section VIII and conclude our work.

II. USE CASE

To illustrate our approach, we consider a railway Automatic Signal Processing system (ASP). The ASP is aimed at recognizing the state of a light signal applicable to a train, a task that is currently performed by train drivers. The ASP shall remain vision-based in order to limit the impact on the infrastructure and limit the cost of its deployment.

Figure 1 gives an overview of the actions performed by a train driver. The ASP must locate the signal, check the integrity of the signaling device (e.g., it is neither broken nor maliciously modified) and determine the indication of the signal automatically. As the ASP performs a critical function with potential safety effects, it shall be certified at SIL4 level according to EN50126 and EN50128 standards. In terms of error rate, and according to studies carried out at SNCF on train drivers, a maximum error rate of 10^{-5} per signal is a sensible objective. For operational reasons, the ASP shall operate in environmental conditions and tracks contexts that are both very complex and variable. In addition, the signal can be occluded or damaged. In our case, the ASP uses ML-based algorithm.

This use case shows two interesting properties with respect to the objective of our study: it implements a simple task (recognizing a light signal) and it operates in an open and weakly structured environment.

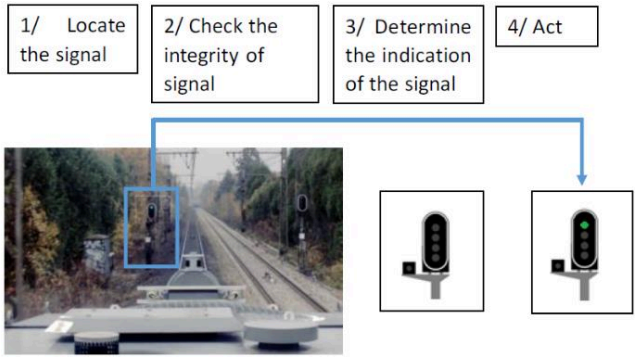


Figure 1 : Action performed by a train driver.

III. APPROACH

A. Context and hypotheses

In this study, we consider the following hypotheses.

- About the system:
  - The ML system is a neural network, even if the proposed approach may be applied to other ML models

- The learning phase is completed (offline learning)
- The architecture of the network is known
- The parameters of the model are known and are accessible.

- About the test objectives:
  - Intentional faults (cybersecurity) are not considered, even if the testing strategies proposed in this may also be applied in that case
  - Implementation faults are not considered. We estimate that those faults can be addressed using classical software testing techniques.

B. What is testing

Testing aims at demonstrating empirically that a system satisfies some properties.

A test may be used to reveal the presence of a fault in the specification or the design of a system. In that case, the test aims at *activating* some dormant fault and at *propagating* the resulting latent error up to the interface of the system to make it observable. In that case, the test may be targeted towards a specific class of faults. In the hardware domain, for instance, some tests targets stuck-at faults while, in the software domain, tests target incorrect coding of conditions or incorrect handling of input domains boundaries.

A test may also be used to verify the compliance of the system with some performance requirements. Performance may be functional (e.g., the accuracy and precision of a decision, a response time) or non-functional (e.g., the tolerance to some unintentional or intentional fault).

More generally, test is used to verify the satisfaction of some high-level properties, such as the *robustness* with respect to incorrect inputs, the *fairness* of the decisions, the *quality* of an explanation, etc.

C. Test construction

To perform testing, several elements are required: a *system to be tested* (the system under test), an *environment* to provide the inputs to the system, and an *oracle* to determine whether the system behaves correctly or not (i.e., the test passes or fails). In addition, a test-stopping criterion is usually defined in order to stop the testing activity since, except for trivial systems, the number of possible tests is usually infinite. The criterion may be structural (e.g., to cover all requirements, all lines of source code, all execution paths, etc.), statistical, or simply driven by the amount of effort deemed acceptable to perform this activity.

The quality of the test may also be evaluated. Referring to the previous definition of testing, the quality may be measured by the capability of the test to reveal errors. Traditionally, several strategies are used: fault/error injection to check if the test reveals the error, coverage analysis.

D. Operational Design Domain

The input domain depends on the purpose of the test. For performance evaluation purposes, the input domain is the "Operational Design Domain[1]" (ODD). In the automotive domain, the ODD is defined as *the specific conditions under which a given driving automation system or feature thereof is designed to function, including, but not limited to, driving modes*. In the aeronautical domain, the ODD concept is related to the concept of *foreseeable conditions*, i.e., the environment in which a system is assumed to operate, given its intended function, including operating in normal, non-normal, and emergency conditions.

Defining *precisely the ODD* is extremely difficult since it shall ideally include all the elements that may affect the function to be performed via its inputs, and all configurations (or states) of these elements. In general, the definition of the ODD cannot be formally "complete", because the environment may be too complex to characterize (i.e., it involves too many variables), or because it is simply unpredictable. Therefore, some of the operational situations remain *unknown,* and possibly unsafe [4]. As proposed in [6], the elaboration of the ODD may combine the points of view of the various actors involved in the operation and design of the system, e.g., the train driver, the image processing chain designer, etc.

Considering our use case, the ODD encompasses the state of the sensors, train, rails, signals, and, more generally, of the complete environment of the system.

Tests exercise the system on situations sampled according to the ODD. Therefore, missing a complete definition of the ODD makes testing a challenge since situation not captured by the ODD will not be considered. The next section elaborates on this challenge.

E. Why is testing ML components difficult?

First, it is important to notice that testing are the primary means to verify ML components today. For systems developed using non-ML techniques, testing may be replaced or complemented by other means such as formal techniques (model checking, abstract interpretation, formal proof, etc.). But even though some successes have already been obtained [7], those techniques are still in infancy in the ML domain.

ML components are particularly difficult to test for various reasons:

- The input space is often extremely large (for example all the train signal of France in all weather conditions), which poses the problem for the stopping criteria definition.
- ML components often address problems which specification is difficult to express in a comprehensive way.
- ML components often address problems where the environment is very complex and difficult to predict (see section on ODD).
- The test oracle is often a human, because the tasks performed with ML usually cannot be performed with classical methods.
- Fault models are unknown (yet) which imposes empirical performance and robustness evaluations.
- Test coverage metrics used for non-ML software are not useful since the behavior of the ML component depends essentially on data, not on control.

Test equivalence classes are difficult to define for ML components. A test equivalence is such that any test case taken in a class reveals the same faults as any other test in the same class. It is a fundamental rationale in software testing (see e.g. [8]).

[1] Defined in Section 3.17 of « Taxonomy and definitions for terms related to driving automation systems for on road motor vehicle. SAE recommanded practices J3016, Sept 2016 , https://www.sae.org/standards/j3016_201609/

F. Our strategy

To cope with the ML based system particularities, we propose a strategy based on 4 testing activities:

- **Performance testing** aims at verifying the performance of the ML model against its specification.
- **Robustness testing** aims at verifying the behavior of the system in the presence of invalid inputs or stressful environmental conditions" [9]. Robustness analysis can be seen as the system behavior analysis regarding any environmental or operating perturbation (known and unknown). The goal is different from performance testing in that robustness testing uses specifically *perturbed* inputs in or out the operational domain.
- **Worst-case testing** consists in exercising the system in the "worst" situations of the ODD and observe its performance.
- **Bias testing** aims at detecting that no bias is present in the decisions of the model itself, i.e. that the model has used relevant features to output the decisions which have led to the recorded performances.

All these activities bring complementary point of views on the component/system at hand. Figure 2 summarizes the testing approach proposed in this paper.

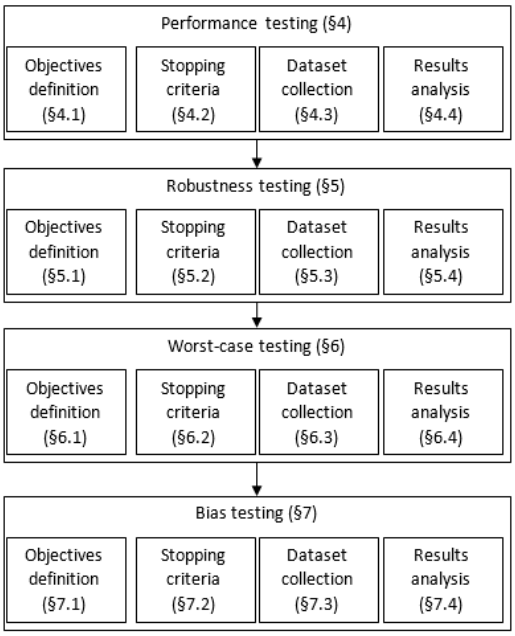


Figure 2 : Testing strategy.

G. Expected Test Results Definition

In order to check whether a test is successful or not, some pass-fail property must be expressed. The pass-fail property takes the form of a predicate involving the output produced by the system and, possibly, some reference value (expected test result, oracle, ground truth ...).

The pass-fail property normally derives from the requirements specification. The test passes if all requirements are satisfied. Unfortunately, requirements are sometimes very difficult to establish for the kind of systems concerned by ML, and so are the tests. This is why a particular care must be taken in the tests and the test data definition and creation.

The oracle may be another system possibly implemented using non-ML technologies (for testing, performance may be less important than for operations[2]) by using a back-to-back configuration or, more generally, any scheme that can be used for runtime monitoring (OOD detection, explicability-based monitoring, continuity/consistency of outputs, etc.). Here, any triggering of the monitoring is considered as a test fail condition.

A specific case is the one where the oracle is the human operator himself/herself. This applies in the situation where the system under test is embedded in an operational system (e.g., a train) and its outputs are compared with the ones of the operator. In our use case, we would for instance compare the decisions taken by the train driver with the outputs of the system under test, and check the compatibility of the decisions with the signal state reported by the system under test.

Another particular case is simulation. Here, the construction of the scenarios can be performed with respect to precise specification, and the expected output may be known a priori. Unfortunately, simulation and reality are different, and the effect of this difference are difficult to estimate. Therefore, testing based on simulation only cannot be considered as an acceptable means of verification.

Finally, some verification can still be carried out without any oracle by leveraging some invariants of the system. This is what is exploited by *metamorphic testing* techniques. See e.g., [10] [11].

H. Test Dataset quality and bias in the Dataset

As discussed in the paragraph "Test Construction", the quality of the actual tests must be verified. As we will see in the next sections, testing strongly relies on datasets. Then test validity is related to test datasets quality. Datasets quality represent different aspects like a proper dataset size, data accuracy and a small amount of untended bias.

The notion of bias in the test dataset is closely linked with the notion of representativeness. If the test dataset is biased, i.e. if some features combinations are not representative of the actual operational distribution, then the estimated performance will not hold for the general operational use. For example, if the system is only tested by day and never by night while it will be operated by night, the actual performance may significantly vary. Section IV.C proposes some strategies to ensure representativeness of the test dataset.

Biases have been identified even in well-known datasets for ML [12], and therefore, their representativeness has to be questioned [13]. The work of [12] could be used as an example to detect biases in the test dataset. Alternatively, weights for the data can be computed through an optimization problem [14]. Another way to reduce the bias is to perform a synthetic data augmentation [15].

IV. PERFORMANCE TESTING

A. Test objectives definition

The purpose of performance testing is to verify that the ML model meets some *performance requirement* expressed using some *performance metrics*.

1) Performance metrics

For classification problems, four basic performance metrics are usually considered [16].

[2] This may not be the case if a huge number of tests are required. See section on performance testing.

- **Accuracy:** $a = \frac{\text{number of inputs assigned to their correct class}}{\text{total number of inputs}}$
- **Precision:** $p = \frac{\text{number of inputs correctly assigned to class } i}{\text{number of inputs assigned to class } i}$
- **Recall:** $r = \frac{\text{number of inputs correctly assigned to class } i}{\text{number of inputs belonging to class } i}$
- **F1-score:** $f = 2pr\ /\ (p + r)$

These performance metrics allow measuring a mean performance on a given test dataset.

Obviously, performance metrics are application dependent. For example in the case of depth prediction from an RGB image by a Convolutionnal Neural Network, the performance metric is Root Mean Square Error (RMSE) on the depth predictions (with respect to a ground truth) [17]. In the following we consider only performance metrics related to a classification task.

2) Performance requirements

To specify performance requirements, the following information are necessary

- A **scope,** i.e., the conditions in which the performance objectives must be satisfied (ex: all operating conditions, low visibility operations, red lights, green lights…)
- One or several **performance metrics** (ex: accuracy, precision, recall…), as described in §1)
- One or several **performance objectives** expressed using the previous metrics (ex:$p \geq 99.9\%$, $a \geq 99.99\%$ …)
- A **confidence level**, possibly expressed as the probability for the *actual* performance of the ML component to be greater or equal to the *estimated* performance (e.g. $99{,}9\%, 99{,}99\%$).

The performance objectives may be class-dependent. For example, in our use case, classifying red lights correctly is more critical with respect to safety than classifying green lights correctly. Therefore, the performance objective for red lights will be higher than the one for green lights as far as safety is concerned.

The performance requirements also depend on the *scope*. For instance, in the railway domain, certification of systems relies on a concept called "GAME" (*Globalement Au Moins Equivalent* / Overall At Least Equivalent) that requires a new system to be at least as safe as the existing system it will replace. In our use case, this means that a misclassification may be considered acceptable if a train driver would have also misclassified the signal in similar conditions. Practically, this means that performance requirements in poor visibility environments may be lower than in normal operating conditions.

3) Examples of performance requirements

The table below provides examples of possible performance objectives for our use case:

	Example #1	**Example #2**	**Example #3**
Scope	All operating conditions All signals	All operating conditions Red signal	Poor visibility All signal
Metric	Accuracy	Accuracy	Accuracy
Obj.	99,9999%	99,99999%	99,999%
Conf. level	10^{-5}	10^{-6}	10^{-4}

B. Test stopping criteria

A test campaign must stop when the performance objective (e.g. 99,999%) is met with the target confidence level (e.g. 10^{-5}). Statistical test stopping criteria may be *distribution-independent* or *distribution-dependent*.

Distribution-independent criteria requires no assumption on the test dataset distribution with respect to the actual operational distribution, but are often intractable in practice, whereas the second one requires some assumptions.

The following notations are used in the next paragraphs:

- $\mathbb{P}(X)$ denotes the probability of an event X
- n denotes the size of the test dataset
- $\hat{p}_n$denotes the probability of an incorrect classification observed on the test dataset
- p denotes the actual unknown probability of an incorrect classification
- δ denotes the desired confidence level.

1) Distribution-independent criteria

Without assumption on the test dataset distribution, a generalization bound can be derived as explained in the course "Introduction to Statistical Learning Theory", in pages 191 and 192 in [18] : for some absolute constant C_1 and C_2 , the generalization bound reads [18]

$$\mathbb{P}\left(p \leq \hat{p}_n + C_1\sqrt{\frac{VC}{n}} + C_2\sqrt{\frac{\log(1/\delta)}{n}}\right) \geq 1 - \delta$$

In this formula, VC denotes the Vapnik and Chervonenkis dimension (VC dimension) [18].

Currently, these generalization bounds are often too loose to be usable. For example, if we take $\delta = 10^{-5}$ with $VC = 10^5$ (typical order of magnitude for a Deep Neural Network made of 10 layers and 10^4 weights [19]) the size n of the test dataset should be greater or equal to 10^{15}, which is impossible to achieve. Improving these generalization bounds is an ongoing research topic. Therefore, in this paper, this type of distribution-independent criteria is not retained.

2) Distribution-dependent criteria

If we assume that the samples of the test dataset:

(i) are independent and identically distributed (i.i.d)

(ii) have the same distribution as the operational distribution relevant to the scope of the requirement.

Note: Relevant operational distribution is defined with respect to the scope of the test. For example, if the scope is "all operating conditions, red traffic lights only", then the relevant operational distribution is the true operational distribution of red traffic lights, actually encountered by trains in operation.

Then the following relation holds, as given in lemma B.10 in page 427 in [20] :

$$\mathbb{P}\left(p \leq \hat{p}_n + \sqrt{\frac{2\hat{p}_n}{n}\ln\left(\frac{1}{\delta}\right)} + \frac{2}{n}\ln\left(\frac{1}{\delta}\right)\right) \geq 1 - \delta$$

This bound is much tighter than the previous one. For example, if we take $\hat{p}_n = 10^{-5}$ and $\delta = 10^{-5}$, the size of the test dataset should be $n \approx 10^6$, which is manageable.

But this bound requires a careful selection of the test dataset to make sure that the assumptions (i) and (ii) are met.

C. Test dataset collection and verification

In this section, we assume that the distribution-dependent stopping criterion is used. This choice implies a careful selection and verification of the test dataset in order to meet the

two assumptions: (i) the samples included in the test dataset are i.i.d and (ii) the test dataset distribution is identical to the relevant operational distribution [21].

1) Collection

In order to meet those two assumptions, we propose two approaches.

- **Random collection.** Data collected in operation are randomly sampled. Data collection and sampling are performed uniformly in order to reproduce the operational distribution. The challenge with this approach is to ensure that randomness and uniformity are effective, without bias.
  Example: for a test of "all operating conditions on a specific train line", all trains operating on the line are equipped with cameras during the whole year, and the test dataset is built by randomly sampling the collected data.
- **Planned collection.** The operational distribution is analyzed to identify all the relevant features and their frequency. Then, data collection is planned in order to gather a dataset with the same features at the same frequency as the operational distribution. The main challenge with this approach is to properly specify the operational distribution, without introducing bias in the specification.
  Example: for a test of "all operating conditions on a specific train line", the relevant features are identified: it includes weather conditions, light conditions, background type, traffic light types… Then the frequency of each situation is assessed, and collection is planned to have samples of each situation at the expected frequency.

2) Verification

The test dataset should be verified in order to check that it satisfies the assumptions. For the first assumption of independent and identically distributed (i.i.d), the verification approach differs depending on the collection approach:

- **Random collection.** The i.i.d assumption is satisfied by design, so the verification activity only aims at assuring that randomness was effective during collection.
- **Planned collection.** The i.i.d assumption may be more difficult to achieve with planned collection, and verification activities should assure that independence is effective.

In both cases, various tests exist to verify the i.i.d. assumption; such as autocorrelation plot, lag plot or turning point test [22]. None of these techniques can provide certainty, but they increase confidence.

The second assumption (consistency of the test dataset distribution with the operational distribution) is verified through an analysis of the consistency between the data collection conditions and the expected operational conditions. This analysis can be qualitative (expert judgement) and/or quantitative (comparison of features frequency).

Notes:

- *Expert judgment is used several times places in our testing process. However, this is not a way to bypass more formal and less subjective solution when available. In addition, expert selection and judgment still relies on a rigorous process, as shown for instance in* [23] *in the context of Assurance Cases.*
- *The operational distribution may evolve over time, for example when changes on the traffic signals or their environment happens. Therefore, the consistency of the test dataset distribution with the operational distribution may degrade over time. A mechanism to monitor the evolutions of the operational distribution should be implemented to identify such situation.*

D. Test results analysis

At the end of the test, two analyses are performed on the test results:

- **Performance analysis:** this first analysis is simple, as it only consists in verifying that the ML model meets its performance objectives. The distribution-dependent criteria defined in section B.2) is applied, and the result is *pass* or *fail*. If the result is *fail*, then the ML model should be retrained, and performance testing redone, with a new test dataset in order to avoid an iterative learning of the test dataset that would introduce bias in the test results. In case of reuse of the same test dataset, the distribution-dependent criteria is no longer valid, and other bounds should be applied, to account for the reuse.
- **Failure analysis:** additionally, if the ML model failed on some samples of the test dataset, and even if the overall performance is acceptable, each failure must be analyzed in order to find the root cause, and ensure that the underlying failure condition is local and not systemic. This failure analysis could be performed using explainability techniques [21].

V. ROBUSTNESS TESTING

A. Test objectives definition

According to IEEE Std 610.12-1990 , robustness is the degree to which a system or component can function correctly in the presence of *invalid inputs* or *stressful environmental conditions*.

We can see from this definition that robustness is twofold. In machine learning, among potential *stressful situations* we consider: adversarial attacks [23], worst cases or edge cases, and *invalid inputs* that are defined relatively to the distribution of the training data [24]. Edge cases will be discussed in the "Worst case section" because their study may be very specific to the application while adversarial robustness and out of distribution robustness share a lot of common point from one application to the other. Then we will focus on these two last situations in the following.

Adversarial attacks consist in transformations of the original data which are in general not visible for the human eye but which have the ability to change the response of the system. Synthesizing sophisticated adversarial examples and elaborating defense strategies are the topics of many research works [25] [26]. These invisible attacks, when used for robustness assessment, aim at providing a better understanding of the algorithm behavior in the neighbourhood of the test points. To do so, an optimization process is used to find a perturbation that provokes a decision change in the limit of a maximum perturbation radius is around a test data. If this process is often successful, the model is considered not robust.

Adversarial attacks can be visually visible and constructed to assess algorithm robustness [27] [28]. In the case of an outdoor application, like our use case, these visible perturbations can represent realistic situations like fog or damage caused to a signal.

Lastly, a technically very different approach but which goal is very similar to adversarial attack based approaches are

based on abstract interpretation of the ML component [29]. In this approach, a theoretical perturbation ball is defined around a testing point and is "forwarded" through a neural network using abstract interpretation theory. This ball is transformed by the network accordingly to its layers. At the end the transformed ball is compared with the decision boundaries. If it does not cross them, the network is robust to the tested amount of perturbation at this testing point.

The other concern about robustness corresponds to data which have not been learnt by the system, often referred as out-of-distribution (OOD) data, anomaly, outlier, or novelty [30] [31]. As a result, the behavior of the system on these data is often not predictable and needs to be evaluated. Nevertheless, it is often critical to assess that a data is in or out of the training distribution of the system. Consequently, it is necessary to develop tools to identify out-of-distribution data for observing the behavior of the system on these data only. The performance degradation should be progressive with respect to the distance between out of distribution data and in distribution ones. If it is the case the ML based system is considered as robust.

B. Test stopping criteria

As explained earlier robustness testing data are designed to explore potentially difficult situations. It is worthy to note that, benchmarks of model robustness are proposed in the literature [32]. However, in this document we consider evaluating robustness of a particular model dedicated to a particular use case. To do so, it is mandatory to be able to introduce a progression of the test cases difficulty in order to be able to measure the degree of robustness of the algorithm otherwise robustness analysis is nonsense. This graduation is easy to obtain in standard perturbation test cases:

- Noise gradation is based on noise level in dB,
- Signal frequency modification (e.g. blur) is controled by the applied filter transfer function profile,
- Geometric distortion are parametrized by geometric parameters like rotation angle.
- Percentage of occultation of a targeted object in the case of partial occultation perturbations.

It may be less obvious as for adversarial robustness testing. The natural choice is to monitor the amount of data modification through the attack radius ρ (the modified data must be comprised in a distance at most equal to ρ from an actual test set data). This idea has been extended to Wasserstein distance in [33]. Wasserstein attacks are interesting for robustness testing as they produce more realistic contents with respect to the data distribution than classical attacks; When studying the possibility of attacks in the physical world [34] and the ratio between efficiency and realization difficulty cannot be assessed by a radius criteria. However, we can draw some test stopping criteria

- Stop when: the algorithm resists to a predefined list of adversarial attacks with a radius less or equal to the natural noise amount (noise measures in the training data for instance).
- Stop when physical sticker attacks covering less than 20% (or what ever the specification) of the object of interest fail.

Concerning out of distribution (OOD) robustness, it is very difficult to establish. OODs are in the class of the "Known unknowns", that is "One knows OOD may occur, but does not know what it will be". Moreover, the notion of progressivity is very difficult to define and to obtain. However once again we consider that it is a very important characteristic that a robustness test must have. To fulfill this requirement we propose two approaches.

The first approach relies on an operational domain expert's analysis. The expert knowing well the operational domain, he should be able to express its limit and possible variations. From this analysis, out of definition data can be gathered or created through a simulation process.

The second approach computes the distance between the nominal data distribution and some gathered OOD data distributions. To do so one can estimate the Wasserstein distance between OOD sets and genuine distribution [35].

The test stopping criteria is based on the OODness measure (proposed by the expert or through the Wassertein distance). The test are stopped when the robustness limit of the system is encountered (observed through a performance drop) or the system performances are maintained for "far enough" OOD data. The rigorous gradation of all robustness tests allows to determine the ML component breaking points (similarly to mechanical testing). These breaking points can be compared to the system specification.

C. Test dataset collection

To evaluate robustness to data perturbation, a simple approach is to take all the available test data and to apply several adversarial transformations on it. A good practice is to list all the adversarial transformations that require to be tested (simple one like [23] or more complex one like [33]). A survey on adversarial attacks can be found in [26]. Then, it is necessary to tune their magnitude in order to make sure that the transformation remains realistic with respect to the application..

Building a dataset for OOD robustness assessment involves two steps. First, human experts may identify situations that should not be encountered by the system in the operational domain then that are not present in the training set. They can also imagine some shift in the operational environment of the system.

From these analysis the OOD datasets can be specified. OOD data are then gathered manually to represent these situations. Another method is to apply style transfer on the test data [36]. This allows evaluating the behavior of the system in realistic situations but with different appearance.

D. Test results analysis

It is impossible to prove that an ML algorithm is robust to every possible stressful situation.

To analyze the model robustness we observe the amount of perturbation that must be injected in the datasets before observing a significant performance drop. A finer observation of the results can enlighten in what kind of stress the system does perform most badly.

What can we do with the results? If the tests are performed at the final validation step, they should be compliant with specification to be considered as passed. If the tests are performed during algorithm development, adversarial attack can be introduced in the training data (adversarial training). If possible this approach can be done with OOD data but this will make more difficult OOD robustness assessment, new distribution of OOD data will need to be imagined and generated.

VI. WORST-CASE TESTING

A. Test objectives definition

Worst-case testing consists in observing the performance of the system in situations considered as being the "worst", or the most difficult, by experts of the operational domain. It is

important to note that this scale of difficulty is strongly related to human capabilities (processing, sensing, and actuation capabilities) and that they may not match with the actual "difficulty" for the ML system. For our use case, it means the worst cases as perceived by the train drivers, but also by the experts in Image acquisition and processing (used in the data acquisition chain).

Because performance testing only considers an average performance, and robustness testing activates the weaknesses of the ML model mostly independently from the operational domain, worst-case testing is a necessary complement to specifically observe the behavior of the model in difficult operational situations. Then, we are specifically interested here in the inputs that are likely to be generated in the real world but having the particularity of a low occurrence.

B. Test stopping criteria

Here the challenge is not to reach the exhaustiveness of the situations that could arise (i.e. the field is infinite), we focus on observing how the ML based system performs. It is clear that its average performances on these situations will be lower than what expected in the average conditions but our goal is to track to what extent of bad situation the system can go and if the operational limits we find are acceptable in practice.

Then stopping criterion is simple: Have all the worst cases identified by the experts been tested ?

C. Test dataset collection

Our use case is an outdoor perception one. Then first worst cases identified by an expert will concern weather, light conditions, scene clutter and etc. More specific situations like tunnel exit can also be identified as worst cases. Then the dataset must be defined from this comprehensive expert list of situations. These situations are frequently identifiable in the PHA (Preliminary Hazard Analysis). It is advised to also test in the neighborhood of the specified limits. Then, we propose a qualitative approach, which consists in classifying, based on an expert opinion, the most problematic situations to be managed for the train driver or the operator in general.

A non-exhaustive list could be:

- Test exceptional situations found in the ODD (with almost no sample in learning database)
- Based on expert knowledge, test critical situations (weather condition, occlusion, background, speed, vibration, line of sight, curve, etc.)
- Test safety critical scenarios (where missing the recognition leads to a critical situation)
- Test combination of already difficult cases, combination of robustness tests applied to worst-case images.

As explained earlier, these situations are rare by definition, then acquiring data representing them necessitate some dedicated effort. Real data will be acquired in the limit of feasibility and cost. To find a solution to this limitation, simulation based testing may be used [37] [38].

D. Test results analysis

The analysis of these tests allow determining the limit between the safe and unsafe domains. Beyond we could study the options of safety mitigation (e.g. failure mode, redundancy, etc.) and the way to monitor these critical situations, even redefine the operational domain. Then, the worst-case tests could provide a clue of confidence in the generalization in the real world and will highlight the limits of the safe domain of ML.

Some difficulties remains in:

- limited amount of real data representing worst cases,
- difficulty to guaranty the representativity of simulations.

VII. BIAS TESTING

A. Test objectives definition

Once it is ensured that the overall performance of the ML model is successfully tested with an unbiased test dataset, as described in section IV, it is also necessary to ensure that no bias is present in the decisions of the model itself. The model shall indeed use relevant features to output the decisions that have led to the recorded performances. In particular, the performance results shall not be obtained due to some attributes in the data (in images, color or texture, for instance) which possibly introduce a bias in the response of the system. This bias in the response corresponds to a flaw in the model's representation which is often due to a bias in the training set itself [39].

If the ML model returns biased responses, then it may be significantly more effective in some situations than others. This may not be acceptable: for example, the traffic signals detection should not be degraded on some tracks whereas it performs well on other tracks.

The absence of bias in the decisions of the model can be assessed using fairness and explicability techniques. Fairness techniques enable the detection of biased behavior of the model, whereas explainability techniques allow the analysis of the feature that most influenced the decisions made.

1) Fairness

Fairness is related to ensuring that the system applies a fair treatment to the data whatever the values of some of its attributes are [40]. The worst-case testing proposed in section VI, or the performance testing described in section IV used on subsamples of the dataset in order to compare the relative performances in different situations, may contribute to the detection of unfair data processing by the model.

2) Explainability

Due to bias in the training set, the ML model may also return its outputs based on irrelevant features of the input. A famous example is presented in [41], where a husky is classified as a wolf based on a snowy background. Using explainability techniques helps detecting such erroneous behaviors. Normally, such bias in a ML model should be detected through the performance testing approach described in section IV, with an unbiased test dataset. Additionally, explainability techniques could be used, such as LIME [41], or an occlusion sensitivity analysis as described in [42]. Nevertheless, caution should be taken with explainability techniques, since most of them have hyper parameters that greatly influence their results [43].

B. Test stopping criteria

Bias testing should be carried out on all relevant classes of bias, and stopped when the review of all these classes is completed. It is not expected that bias testing will enable the detection of unknown class of bias, because each testing techniques addresses a particular class of bias. Identification of all possible sources of bias cannot be achieved in practice. Therefore the identification of relevant classes of bias relies on expert judgement, in agreement with the certifying authorities.

Therefore, the following approach is suggested:

- **Identification of relevant classes of bias:** this activity should involve experts with various backgrounds, including data scientist, acquisition system expert, operator, etc.
- **Bias testing:** for each potential class of bias, one or several bias detection techniques should be applied to confirm or infirm the bias in the ML model. Fairness or explainability techniques, as presented in A, could be used depending on the type of bias. For each type of bias, the number of samples tested can be chosen based on the expected confidence interval, using criteria similar to the one described in section IV.B.2).
- **Stop:** bias testing stops when all known potential sources of bias have been tested on a sufficient number of samples.

C. *Test dataset collection*

The test dataset is defined by the potential sources of bias identified. For each potential source of bias, a representative test dataset should be collected, in a similar way as the one described in section IV.C.

For example, if the potential source of bias to be tested is "the ML model performs better on yellow lights than on red lights" (fairness problem), the dataset should extract two subsamples from the test dataset gathered in section 4.3: one subsample containing yellow lights, and one subsample containing red lights.

If the potential source of bias to be tested is "the ML model builds its decisions on the shape of the signal instead of its color" (explainability problem), then a dataset containing various combinations of signal shapes and colors can be used.

D. *Test results analysis*

For each confirmed source of bias, its acceptability should be checked with respect to the requirements. If the bias is confirmed and not acceptable, the ML model should be retrained to remove the identified bias(es). The analysis differs depending of the type of bias:

- **Fairness:** some degree of unfairness may be accepted. For example, it may be acceptable to have a ML model that performs better on red lights than on green lights, because it does not compromise safety. Therefore a careful safety assessment may support the acceptance of unfair ML model.
- **Explainability:** if the ML model makes decisions based on irrelevant features of the input, the ML model should not be accepted, except if the occurrence of such situation is proven to be rare enough.

VIII. RELATED WORKS

As of today, even if there is already a large number of ongoing initiatives about the *engineering* of Machine Learning in the context of critical systems, results are essentially focused on the identification of high-level challenges [1][2], or certification-level guidance [3] [45]. Our paper proposes solutions to address the challenge of verification and refines the relevant development phases down to practical engineering activities (definition of test stopping criteria, definition of dataset, etc.). However, our proposal remains partial for it only consider verification by testing and does not cover other verification means such as, for instance, formal verification.

A survey of 144 papers in Machine Learning Testing is given in [1]. This paper, which provides a comprehensive and structured analysis of the recent results concerning testing, identifies a large set of methods and tools to support this activity. Those methods and tools are some of the "building blocks" that we integrate in the testing strategy proposed in this paper. Our proposal is complementary to these papers in the sense that it proposes to organize in a sensible and practical way the various testing activities.

IX. CONCLUSION

To test safety-critical systems based on machine learning, we have presented an approach divided into four activities: performance assessment, robustness verification and worst-case testing. For each of these, we have given objectives and stopping criteria, and some solutions to collect the appropriate data.

In addition, we have illustrated our propositions in our use case of railway signal identification. As usual in machine learning domain, data are of particular importance. To ensure valid testing, we have underlined several considerations, which must be taken into account like the definition of the operational domain, or the detection of unintended biases in the datasets or in the model's decisions. Particularly for worst-case testing, due to the scarcity of worst-case situations.

As it can be seen, our approach intends to use complementary point of views of the system at hand to construct an efficient testing strategy.

REFERENCES

[1] IRT Saint-Exupéry; ANITI, "White Paper Machine Learning in Certified Systems," Toulouse, 2021.

[2] I. Stoica and et a., "A Berkeley View of Systems Challenges for AI," Berkeley, 2017.

[3] EASA, "Concepts of Design Assurance for Neural Networks (CoDANN)," 2020.

[4] 2. ISO/PAS, "Road Vehicles - Safety of the Intended Functionality (SOTIF)," Jan. 2019.

[5] F. Maleki, N. Muthukrishnan, K. Ovens, C. Reinhold and R. Forghani, "Machine Learning algorithm validation : from essentials to advanced applications and implications for regulatory certification and deployment," *Neuroimaging Clinics,* vol. 30, pp. 433-445, 2020.

[6] S. Picard, E. Jenn, C. Chapdelaine, B. Lefevre, C. Cappi and L. Gardes, "Ensuring Dataset Quality for Machine Learning Certification," in *10th IEEE International Workshop on Software Certification*, 2020.

[7] C. Liu, T. Arnon, C. Lazarus, C. Strong, C. Barret and M.-J. Kochenderfer, "Algorithms for Verifying Deep Neaul Networks," *Found. Trends. Optim,* vol. 4, 2019.

[8] I.-2. RTCA, (DO-248C) Supporting information for DO-178C and DO-278A, Dec. 2011.

[9] I. s. 6. IEEE, IEEE Standard Glossary of Software Engineering Terminoloy (IEEE 610), 1990.

[10] T. Chen, F.-C. Kuo, H. Liu and P.-L. Poon, "Metamorphic Testing: A Review of Challenges and Opportunities," *ACM Computing Surveys,* 2018.

[11] X. Xie, J. Ho, C. Murphy, G. Kaiser, B. Xy and T. Y. Chen, "Testing and Validating Machine Learning Classifiers byMetamorphic Testing," *Journal of system and sofware,* 2011.

[12] A. Torralba and A. A. Efros, "Unbiased look at dataset bia," in *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011.

[13] A. Paullada, I. D. Raji, E. M. Bender, E. Denton and A. Hanna, "Data and its (dis) contents: A survey of dataset development and use in machine learning research," in *Advances in Neural Information Processing Systems Workshop : ML Retrospectives, Surveys & Meta-analyses (ML-RSA)*, 2020.

[14] Y. Li and N. Vasconcelos, "Repair : Removing representation bias by dataset resampling," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019.

[15] N. Jaipuria, X. Zhang, R. Bhasin, M. Arafa, P. Chakravarty, S. Shrivastava, S. Manglani and V. N. Murali, "Deflating dataset bias using synthetic data augmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2020.

[16] I. J. Goodfellow, Y. Bengio and A. Courville, Deep Learning, MIT Press.

[17] M. Moukari, S. Picard, L. Simon and F. Jurie, "Deep Multi-scale architectur for monocular depth estimation," in *ICIP*, 2019.

[18] O. Bousquet, U. von Luxburg and G. Rätsch, Advanced Lectures on Machine Learning : ML Summer Schools 2003, Canberra, Australia, February 2-14, 2003, Tübingen, Germany, August 4-16, 2003, Revised Lectures, Springer, 2003.

[19] P. L. Barlett, N. Harvey, Liam, C. Liaw and A. Mehrabian, "Nearly-tight VC-dimension and Pseudodimension Bounds for Piecewise Linear Neural Networks," *Journal of Machine Learning Research,* 2017.

[20] S. Shalev-Shwartz and S. Ben-David, Understanding machine learning : From theory to algorithms, Cambridge University Press, 2014.

[21] D. Bau, B. Zhou, A. Khosla, A. Oliva and A. Torralba, "Network Dissection:Quantifying Interpretability of Deep Visual Representations," MIT, 2020. [Online]. Available: http://netdissect.csail.mit.edu/final-network-dissection.pdf.

[22] J.-Y. Le Boudec, Performance evaluation of computer and communication systems., EPFL Press, 2010.

[23] P. J. McGee and J. C. Knight, "Expert judgment in Assurance Cases," in *10th IET System Safety and Cyber-Security Conference 2015*, Bristol, UK, 2015.

[24] I. J. Goodfellow, J. Shlens and C. Szegedy, "Explaining and harnessing adversarial examples," in *International Conference on Learning Representations (ICLR)*, 2015.

[25] C. M. Bishop, "Novelty detection and neural network validation," in *IEE Proceedings-Vision, Image and Signal processing*, 1994.

[26] A. Athalye, L. Engstrom, A. Ilyas and K. Kwok, "Synthesizing Robust Adversarial Examples," in *International Conference on Machine Learning (ICML)*, 2018.

[27] X. Yuan, P. He, Q. Zhu and X. Li, "Adversarial Examples : Attacks and Defenses for Deep Learning," *IEEE Transactions on Neural Networks and Learning Systems,* vol. 30, no. 9, pp. 2805-2824, 2019.

[28] K. Pei, Y. Cao, J. Yang and S. Jana, "DeepXplore: automated whitebox testing of deep learning systems," *Commun. ACM,* vol. 62, no. 11, p. 137–145, Oct. 2019.

[29] A. Odena, C. Olsson, D. G. Andersen and I. Goodfellow, "TensorFuzz: Debugging Neural Networks with Coverage-Guided Fuzzing," 2018.

[30] T. Gehr, M. Mirman, D. Drachsler-Cohen and P. Tsankov, "AI2: Safety and Robustness Certification of Neural Networks with Abstract Interpretation," in *2018 IEEE Symposium on Security and Privacy*, 2018.

[31] R. Chalapathy and S. Chawla, "Deep learning for anomaly detection : A survey," *arXiv preprint arXiv:1901.03407,* 2019.

[32] H. Wang, M. J. Bah and M. Hammad, "Progress in Outlier Detection Techniques : A Survey," *IEEE Access,* vol. 7, pp. 107964--108000, 2019.

[33] D. Hendrycks and T. Dietterich, "Benchmarking neural network robustness to common corruptions and perturbations," in *ICLR*, 2019.

[34] K. Wu, A. H. Wang and Y. Yu, "Stronger and Faster Wasserstein Adversarial Attacks," in *ICML*, 2020.

[35] K. Eykholt, I. Evtimov,, E. Fernandes, T. Kohno, B. Li, A. Prakash, A. Rahmat and D. Song, "Robust Physical-World Attacks on Deep Learning Visual Classification," in *CVPR*, 2018.

[36] C. Villani, Optimal Transport: Old and New, Springer, 2009.

[37] L. Gatys, A. S. Ecker and M. Bethge, "Image Style Transfer Using Convolutional Neural Networks.," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), (pp. 2414-2423).*, 2016.

[38] T. Dreossi, D. J. Fremont, S. Ghosh, E. Kim, H. Ravanbakhsh and M. Vazquez-Chanlatte, "VerifAI: A Toolkit for the Formal Design and Analysis of Artificial Intelligence-Based Systems," in *31st International Conference on Computer Aided Verification (CAV)*, 2019.

[39] M. Mousavi, A. Khanal and R. Estrada, "AI Playground: Unreal Engine-based Data Ablation Tool for Deep Learning," 2020. [Online]. Available: https://arxiv.org/pdf/2007.06153v1.pdf.

[40] Q. Zhang, W. Wang and S.-C. Zhu, "Examining cnn representations with respect to dataset bias," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.

[41] E. del Barrio, F. Gamboa, P. Gordaliza and J.-M. Loubes, "Obtaining Fairness using Optimal Transport Theory," in *International Conference on Machine Learning (ICML)*, 2019.

[42] M. T. Ribeiro, S. Singh and C. Guestrin, "" Why should I trust you ?" Explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 2016.

[43] C.-H. Cheng, C.-H. Huang, H. Ruess and H. Yasuoka, "Towards dependability metrics for neural networks," in *The 16th ACM/IEEE International Conference on Formal Methods and Models for System Design (MEMOCODE)*, 2018.

[44] N. Bansal, C. Agarwal and A. Nguyen, "SAM : the sensitivity of attribution methods to hyperparameters," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020.

[45] DGA, "Guide méthodologique pour la spécification et la qualification des systèmes intégrant des mo-dules d'intelligence artificielle," 2020.

Can we reconcile safety objectives with machine learning performances?

Lucian Alecu, Continental Lucian.Alecu@continental-corporation.com
Hugues Bonnin, Continental hugues.bonnin@continental-corporation.com
Thomas Fel, SNCF thomas.fel@sncf.fr
Laurent Gardes, SNCF l.gardes@sncf.fr
Sébastien Gerchinovitz, IRT Saint Exupéry and IMT sebastien.gerchinovitz@irt-saintexupery.com
Ludovic Ponsolle, Apsys Airbus ludovic.ponsolle@apsys-airbus.com
Franck Mamalet, IRT Saint Exupéry franck.mamalet@irt-saintexupery.com
Eric Jenn, IRT Saint Exupéry and Thalès Avionics eric.jenn@irt-saintexupery.com
Vincent Mussot, IRT Saint Exupéry vincent.mussot@irt-saintexupery.com
Cyril Cappi, SNCF cyril.cappi@sncf.fr
Kevin Delmas, ONERA Kevin.Delmas@onera.fr
Baptiste Lefevre, Thalès baptiste.lefevre@fr.thalesgroup.com

Abstract— The strong demand for more automated transport systems with enhanced safety, in conjunction with the explosion of technologies and products implementing machine learning (ML) techniques, has led to a fundamental questioning of the trust placed in machine learning. In particular, do state-of-the-art ML models allow us to reach such safety objectives? We explore this question through two practical examples from the railway and automotive industries, showing that ML performances are currently far from those required by safety objectives. We then describe and question several techniques aimed at reducing the error rate of ML components: model diversification, monitoring, classification with a reject option, conformal prediction, and temporal redundancy. Taking inspiration from a historical example, we finally discuss when and how new ML-based technologies could be introduced.

Keywords— machine learning, safety, certification, probabilistic assessment, trustworthiness.

I. INTRODUCTION

Recent efforts in developing next generation transportation systems bet on significant contributions from machine learning-based predictive models to implement highly automated functions. Yet, despite the impressive progress seen in recent years, many questions remain open in regard to the capacity of such statistical models to comply with existing safety standards. Here we focus on one such central question: is the error rate of current state-of-the-art ML models low enough to allow the implementation of safety-critical functions?

In this paper, we explore this question through two examples from the railway and automotive industries and make the following contributions:
- In Section II, we note that the performances of ML-based detection systems used in each of the two use-cases is orders of magnitude lower than that required to reach safety objectives (or at least for now, or to reach it directly, i.e., without non-ML software components).
- In Section III, we describe potential directions for safe integration of ML models: can we use several redundant ML models in parallel? Can the ML component be monitored by rejecting specific inputs or changing the (confidence of the) outputs when needed? Can we improve ML performances with temporally redundant inputs? We report experimental results from the ML literature indicating that these solutions can be useful, but are currently not at all sufficient to reach safety objectives. Since some of these techniques reduce the availability of the ML component, we also discuss the availability-reliability trade-off that naturally appears.
- Finally, in Section IV, we take a historical perspective on how technologies were introduced in the past and try to relate it to (and differentiate it from) how ML-based technologies can be introduced.

From a pedagogical viewpoint, this paper targets a mixed audience of safety engineers and ML researchers. We hope it can help highlight some key concepts and refocus research efforts on relevant topics from the safety perspective.
In this document we focus our analysis exclusively on the performance of ML models, since they have recently shown considerable success in many complex tasks, and they currently represent the dominant AI paradigm used in many industrial applications. Other AI-flavored techniques are not discussed here, even if they may partly address some of the issues raised in this paper.

Related works. The question of ML reliability and trustworthiness for integration to safety-critical systems has received a lot of attention recently. Many projects, institutes or working groups (e.g., ANITI, DEEL, SafeAI, the EUROCAE WG-114 and SAE G-34, ISO TC22/SC32/WG14, ISO-IEC JTC1/SC42/WG3, RISE SMILE), as well as workshops or conferences (e.g., AAAI-SafeAI, MLCS, WAISE, ERTS, Future Intelligence) were created to address these challenges. Though several reports and papers have been published very recently (e.g., white paper from DEEL [1], CoDANN II [2], [3]), many questions remain open.
Authors from the DEEL project [3] provide a thorough review of existing methods aimed towards the certification of ML-based software in critical systems.
Work by authors from RISE SMILE project series [4] provides another review of verification and validation methods that aim to embed ML technologies into automotive critical systems. The authors highlight the gap between current standards and the ML-based software engineering and conclude that "ISO 26262 largely contravenes the nature of DNNs". They enumerate and discuss several challenges and directions towards new methods and norms for certifying critical systems based on ML.

A similar assessment is provided by [5], which claims that ISO 26262 cannot appropriately manage ML-based software and propose new measures to adapt the current norms.
From the side of ML literature, many papers have addressed properties such as generalization, robustness, or explainability [6], [7], [8], [9], [10], yet we argue that none of these results alone can layout the necessary safety foundations for ML-based critical systems, at least for now.

II. From ML performance to Safety objectives

A. ML performance and safety objectives

In ML parlance a model refers to a software implementation of some stochastic function which provides predictions on unseen samples from a given input domain. In the context of system engineering, a function implements a decision to be taken, given the available information and constraints. Its role is to implement the necessary logic in order to alter or to maintain the current state of the system, according to a given specification.
Safety analysis seeks to establish causal links between the erroneous components, the decisions taken by the system and the foreseeable failures.
Safety objectives can be formulated in terms of acceptable failure rates for each critical function implemented by a system. In industrial standards the acceptable failure rates are expressed as number of average failures per hour of operation. In addition, assurance levels are considered: e.g., DAL (Design Assurance Level) for aviation industry, as introduced by ARP4754 [11] [12] [13] [14], SIL (Safety Integrity Level) for industry in IEC 61508 standard [15], adapted for Railway systems in EN 50126 [16] [17] [18], and tailored in ASIL (Automotive Safety Integrity Level) in ISO 26262 [19]. Least stringent levels of performance require less than 10^{-3} failures per hour for such functions, while the strictest one generally require less than 10^{-9} failures per hour on average[1].
In order to assess whether an ML-based function complies with the safety objectives it is thus essential to estimate its failure rate. While everyone can acknowledge that there is a close dependency between the error rate usually reported in the ML literature and the failure rate of the system relying on ML predictions in operation, the exact relationship is far from obvious to describe analytically. Moreover, this relation is use-case dependent.
In this section we consider ML-based components for which every decision relies on a single prediction provided by the ML model. To conduct a rigorous safety assessment of a ML-based component it is key to estimate its error rate per hour of operation of the system, based on some set of assumptions and empirical observations.
We can draw here a parallel between the methods used to compute error rates for hardware components and for ML-based components. At first glance the comparison seems appropriate as both types of components are characterized by stochastic processes.
For most ML models these estimates can be computed from observational data, by sampling the operational domain. Depending on the sampling schemes and other assumptions used to compute it, statistical guarantees can be obtained with respect to the gap between the empirical error rate (i.e. computed from the sampled observations) and the true error rate (the one that would have resulted by performing the computation on all possible observations of the entire operational domain).
On the other hand, error rates of hardware components are reported after performing experiments in various regimes (normal operation, operation under stress, accelerated aging, etc.) and calibrating some expected distributions for the error rates as to closely match this empirical evidence. Failure rates of the system embedding them are then estimated according to various failure analysis tools and methods (FTA, FMEA, Human Factor Analysis, Functional Hazard Analysis, etc.). Most of these methods rely on assumptions and practices that must be used with a lot of attention for ML components (e.g., sub-components may not be independent as shown in Section III.A, the error distribution in operation is unknown, etc). Thus, in order to make accurate estimations of error rates for ML-based functions, one must rely almost exclusively on empirical observations (for now) of the actual behavior of these functions in operation-like scenarios. In order for these to be statistically significant, they require a large number of samples to be collected in real-life scenarios. For example, this would require at least 100 000 hours of operational testing (obviously simulation can help to accelerate the time) to decide whether the expected failure rate of a ML-based system function is lower than 10^{-5} errors per hour, and this under the stringent assumption that all the predictions provided by the ML model are i.i.d. over training and operation. The further we are from these assumptions, the more samples we would need to gather in order to reach such conclusions. Unless prior knowledge regarding mathematical properties of the operational domain and / or of the ML model, we are required to perform extensive testing in order to be able to guarantee statistically sound estimations of the error rate. In spite of these challenges, it can be useful to estimate the error rate under idealized assumptions. If the safety objectives are met in these conditions, further refinements (i.e., under realistic assumptions) should be conducted, but if they are far from being satisfied, the negative conclusion is a good indication that safety objectives are difficult to prove in realistic conditions.

B. Two examples from the railway and automotive domains

To illustrate the large gap between the performance of ML system and the targeted safety objectives, let us consider two examples: a computer vision system used in the railway domain, and a weather prediction system used in the automotive domain.

1) Railway and computer vision

Functional description. In our first example, we consider a railway automatic signal reading system. This system aims at recognizing the state of a light signal applicable to a train, a task that is currently performed by train drivers.
Figure 1 gives an overview of the actions performed by a train driver. The system shall determine the indication of the signal automatically. The system must be able to operate in any environmental conditions in the open world of a typical railway infrastructure. This includes scenarios where the signal can be partially occluded by vegetation or due to some weather condition (fog, snow, etc.) or damaged. The recognition system relies on a ML-based computer vision model. The system takes in input an image (coming from a camera in front of the train) in which the approximate location of the signal is

[1] ML technologies are not completely covered by any safety standard/norm today, even recent ones like SOTIF [53]. In this paper, we consider that, due to its probabilistic nature, it makes sense to allocate to the ML components some probabilistic objectives, based on the analogy with HW components, or the system level components.

known. This allows us to use a bounding box and thus to limit the detection effort of the signal in the complete image. In the end, the ML model has to cope with a small image containing only one signal that shall be classified.
This use-case features two interesting properties with respect to the objective of our study: it implements a simple task (recognizing a light signal) and it operates in an open and weakly structured environment.

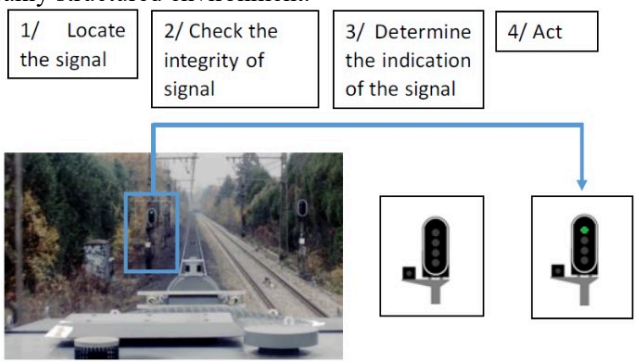


Figure 1 : Action performed by a train driver.

Safety objective. The analysis of the current human-operated system leads to an average target of maximum 10^{-5} failures per hour, which is the observed number of failed recognitions of a red signal per hour of driving on average. This in turn corresponds to 10^{-5} failures for red signal recognition if we consider the number of red signals that a train driver usually encounters during a journey, which is one red signal per hour on average. The red signals are the most critical ones, and this consideration determines the safety objectives that we consider here for the ML system that shall recognize the signals, in particular red signals.
ML performances compared with safety objective. As of today, in the performance results carried out on test sets deemed representative of real-world situations the best classification models achieve on average 10^{-2} errors per signal.
Compared with the safety objective, the order of magnitude of the observed performance is clearly insufficient.

2) Automotive and weather prediction

Functional description. Let us consider an assisted driving service using weather data to enhance the safety of the driving experience. Concretely, this service predicts the state of the road surface (icy, wet or dry) from various weather-related indicators (e.g., air temperature, humidity and water level on the road surface) available at any given location. This ML-based predictive system is used to implement an assisted speed management system as a highly automated driving function (HAD). The system takes control from the driver and handles the vehicle accordingly in scenarios deemed safe, while handling the control to the driver in the remaining situations.

Safety objective. The proposed ML-based service is expected to introduce some risks, especially when the predictions of the road conditions are erroneous (e.g., the road surface is deemed dry instead of wet or icy and the vehicle moves at high speeds). In such scenarios, due to its autonomous nature, the HAD function reduces the controllability of the vehicle, which potentially increases the risks of collisions. For this reason, the expected failure rate that would qualify as minimally safe by current automotive standards is 10^{-5} errors per hour[2].

ML performances compared with safety objective. The described service relies on an ML-based prediction model which computes periodically the road conditions in the near future based on recently observed weather-related data, for each square tile of size S of the Earth's surface. Each vehicle having enabled such a service would make a request for a new prediction periodically, depending on its geographical position. A very simplistic model in which the vehicle is considered to move at a constant speed, a periodic tile change, and an error rate per prediction (ML error rate) of 0.01, leads to a failure rate of 0.5 failure per hour. Again, to reach the required safety objectives, it would be necessary for this error rate to be orders of magnitude lower.

C. The Gap

We concluded in the two use-cases above that the ML performances are several orders of magnitude below those that would be required to reach the safety objectives. Such a gap should not be omitted by the new guidelines aiming for the integration of ML into safety-critical systems.
In addition, neither rigorous software development efforts nor properly documented data collection rules are sufficient to reduce the gap between the actual performance of ML components and the safety objectives. Even a properly coded ML model trained on properly collected data could lead to unacceptable error rates—at least for now.
Does this mean there are no hopes for integration of ML components into safety-critical systems? We believe not but urge to pursue research efforts.

III. Reconciling Safety Objectives with Machine Learning Performances?

Generally speaking, the prediction performances of ML models can be enhanced either by improving the data management and processing or the model's engineering.
Enhancing performances with data can be done by collecting more data, by improving their quality, by including more diverse sources of data (e.g., sensors), by exploiting certain constraints or properties of the operational domain, etc. Obviously, more data usually means dealing with more uncertainty, therefore obtaining overall better performances is not guaranteed.
From the model's engineering viewpoint, we can apply various techniques to improve, e.g., robustness, generalization or explainability of the models.
Next, we focus on possible solutions to decrease the error rate of ML components. We first question the possibility of using several redundant ML systems in parallel (Section A). Then, we describe methods that either detect inputs potentially leading to erroneous predictions (Sections B and C.1), or that are allowed to output less precise predictions for hard-to-classify inputs (Section C.2). We finally question the possibility of using temporally redundant inputs (Section D).
Part of these methods reduce the ML model's availability by, e.g., ignoring its predictions in some situations. We thus also discuss the availability-reliability trade-off that arises there.

A. Model diversification

Using several redundant components in parallel is a classical way to reduce the failure rate of a system when the

[2] This requirement mainly arises from two assumptions: 1) as explained in II.A a probabilistic objective is allocated to the ML component 2) the predicted state of the road is not the only information used by the envisaged system, this particular contribution being considered as ASIL C or B.

components' failures are independent (and when the costs incurred by the additional components are justified from the safety perspective). Though it is very natural to implement redundant ML systems, it is now known that the independence assumption is hard to ensure in practice. As an illustration, Figure 2 below shows on the ImageNet dataset how the joint error rate of n ML models in parallel would decrease as a function of n if the models' errors were independent (exponential decrease, in blue), and how this joint rate actually decreases in practice (slow decrease, in red). For any value of n, we considered the first n models in the list {EfficientNet,DenseNet121,ResNet50V2, MobileNet,VGG16models}, which are state-of-the-art deep learning models for this dataset. We say we have a joint error if all n models make a prediction error simultaneously.

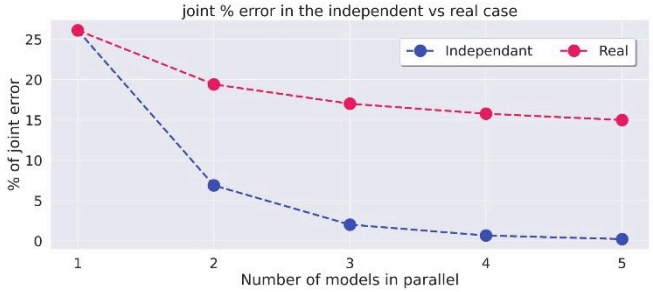


Figure 2: joint error rate versus number of redundant ML models

Of course, we cannot conclude from Figure 2 that it is impossible to build independent (and accurate) deep learning models. What if we tried to get more diverse models, such as models trained with different hyperparameters, loss functions or optimization procedures, models with different architectures, or models trained on different datasets? Though somewhat negative answers were formulated by [20] (about standard ImageNet deep learning models) and by [21] (about bootstrapping deep learning models), the very recent paper [22] provides a more positive answer. The authors show that significantly different training methodologies can in fact lead to models with partially uncorrelated errors or, equivalently, models that err on partially disjoint test points. More precisely, for a pair of models, let us call "error inconsistency" the proportion of test points for which only one model errs (while the other is correct). On ImageNet, the authors show that two different optimization objectives (supervised versus contrastive learning) can yield an error inconsistency of around 13%, while it can go up to approximately 20% for models that are trained on two different datasets.

Though the results of [22] indicate that it is possible to build diverse models, they do not reach the diversity level that independent models would do: for two models with around $p = 25\%$ test error rate (as was the case in that paper), if the two models were independent, the error inconsistency would be of $2p(1-p) = 0.375$, which is larger than the proportions of 13% or 20% mentioned above. Nonetheless, as shown by [22], producing diverse ensembles of deep learning models can help increase the ensemble test accuracy. For instance, two very diverse models trained with supervised learning on the one hand and contrastive learning on the other hand, both with about 75-76% test accuracy on ImageNet, were shown to achieve about 83% after ensembling. The main reason seemed that these diverse models specialize to two (partially) different subdomains of the data. Note that the gain in accuracy is however not as high as the one we could hope to get with independent models and remains orders of magnitude worse than that prescribed by safety objectives.

Two natural yet important-for-safety remarks should be made. First, the fact that two ML models do not have independent test errors is not at all surprising, since the two models are evaluated on the same test points, with the same inputs. There can be test points that are easy to classify with both models, while other points can be hard to classify for both models (e.g., occluded images for which the ground truth is not at all readable), even if these two models were trained on different datasets. This indicates that their failure modes are typically not independent. Second, if we accept to work only with `partially independent' models, estimating their error correlation accurately seems at least as hard as estimating the error rate of the ensemble itself (that is, the model after consolidation). It is thus not obvious a priori how one could leverage partial independence to prove safety at system level by combining low-reliability ML components. While the independence assumption is common in many safety analysis methods when combining various components, and helps prove a small system failure rate by, e.g., multiplying individual error rates, the formula to combine individual error rates is not known a priori when ML is involved.

B. Monitoring

Monitoring techniques are common means for detecting unexpected inputs or conditions that could lead to erroneous output of a system. Once those events are detected, prevention, recovery, or passivation actions can be taken to prevent an unsafe failure of the system. Those events may be due to the activation of intentional (cyber attack with malicious purpose) or unintentional errors, as shown in the following table.

Category	**External**	**Internal**
Intentional	Spoofing, adversarial attacks	*Not relevant*
Unintentional	Out Of Distribution inputs	Lack of robustness, Inconsistent behavior

Different monitoring approaches can be used:

- **ODD (Operational Design Domain) Monitoring** [23] verifies that the ML-based system is operated in its usage domain (e.g., range of brightness, temperature, speed…)
- **OOD (Out Of Distribution) Monitoring** [24] [25] ensures that the ML Model operates in the distribution defined during the training process. Monitoring techniques include distance-based approaches, One-class classification, probabilistic approaches, reconstruction approaches, etc.
- **Attacks monitoring** [26] allows to detect adversarial attacks. It can be based on Input source diversification, properties imposed at design phase, ML adversarial detector, prediction inconsistency, etc.
- **Robustness monitoring** ensures that the ML Model is used in a stable area. Robustness can be verified using constraint programming, abstract interpretation, geometrical approaches, statistical approaches, etc.
- **Consistency monitoring** analyzes the consistency of outputs with the inputs. In particular, inconsistent sequences of outputs can be detected using Detection of unstable states, Functional rules, etc.

In this paper, for reason of space, we will focus on OOD and Robustness monitoring, presenting an overview of State-of-the-Art techniques and their application for monitoring.

1) OOD Monitoring

The monitoring of Out-of-distribution (OOD) is a complex topic, often related to the detection of anomalies and to the underlying question of input normality. OOD inputs correspond to samples that seem to be drawn from a distribution different from the one used during training. Indeed, the use of a training data distribution representative of the real-world data is crucial for the quality of the learning process and the adequacy with the operational domain. However, the capabilities of the trained model to fulfill its task and the probabilistic guarantees on the model performances can only be ensured for the same type of data that it has seen during training. Therefore, OOD monitoring focuses on the detection of inputs that do not correspond to the normality represented by this training distribution. This led to families of OOD detection techniques relying on an estimation of the similarity between a sample and the ones seen during training, such as distance-based approaches [27] or one-class classification [28]. This kind of techniques can be used for obvious tasks like detecting major anomalies in the data, for instance when the expected task cannot be performed on the input (e.g., a picture of a dog given to classifier trained on digits). However, it can be harder to specify for elements belonging to the tail of the distribution, where we could also expect to rely on the generalization capability of the model. For this reason, these techniques always use a form of threshold on the similarity to discriminate OOD from ID inputs, and this choice of threshold will have a direct impact on the availability of the function.

Furthermore, we also need to consider the question of the performance of OOD detection methods, which is addressed in a few papers [29, 30]. These articles compare the various techniques of the literature on dedicated datasets, typically with various images corruptions and noises, or representing industrial defects. It is worth noting that one major result of these comparisons is the inconsistency of most techniques, which may perform exceptionally well on specific types of simple corruptions (up to 100% detection) but may perform below average on others. Moreover, on complex tasks, or in high dimension, the average accuracy of the best methods remains particularly low, typically below 80% for realistic settings.

In the railway use case, we considered primarily the detection of OOD inputs as a means to detect unreadable signals, which could be caused by several things:

- An occlusion by an environmental element (e.g., a pole, a bird, or a tree) which makes the signal partially or entirely unreadable. We also consider other environmental elements such as fog or brightness issues, even if these could be easily associated with other types of monitoring such as the ones based on robustness.
- A failure in the cropping algorithm (e.g., crop of a wrong area of the picture, an issue in the crop size, a crop of the wrong signal, etc.), which may result in the absence of the expected signal on the input
- A defective sensor, (e.g., a failure of the camera lens), could be detected with a robustness monitoring, but the resulting ML images will be unreadable, and the OOD detection should trigger as well in this case.

In fact, most of these root causes can be monitored separately from the ML component, but the OOD detection appears to be valuable, as it can cover all these cases independently from the cause. However, the rate of potential OOD inputs is an important factor, as it is considered to be very low in our case (less than 10^{-4}). Moreover, it is important to remember that an input classified as OOD does not necessarily mean that the model will not be able to perform its task correctly, only that the model performances cannot be guaranteed. This is especially true for partial occlusions, for which there is no way to define how much of the signal needs to be occluded before considering it unreadable.

If we take into account the poor performances of OOD detection methods, we can safely consider that a method providing a true positive rate of 80% or 90% with a detection threshold chosen to have 1% false alarms is an excellent method in this realistic setting. In our use case, if we consider an expected OOD rate of 10^{-4}, the benefits of the OOD detection appear to be very limited: the few OOD inputs detected will not change significantly the accuracy of the model, but the availability will drop from 100% to 99% due to the false alarms. In this situation, the use of OOD monitoring mechanism will be damaging the availability without improving the reliability, but it should not be the same for systems with higher expected OOD rates. A general formula will be detailed in the following section with a numeric analysis on Robustness monitoring for which the conclusions are transferable to other types of monitoring.

2) Robustness Monitoring

According to the CODANN [2], two types of robustness can be considered; either the capability of the learning algorithm to produce "similar" models for "close" training datasets; or the model stability e.g., if inputs, x and x' are close then predictions f(x) and f(x') are also close. In this paper, we address the runtime monitoring of a trained ML component hence we focus the remainder of this section on the state-of-the-art approaches enabling model stability assessment.

The model stability is classically specified w.r.t. a measure $|.|$ on inputs and outputs to formally capture the "closeness", thus a stable model ensures that for any delta-close inputs (x,x'), predictions (f(x),f(x')) are epsilon-close, that is:

$$|x - x'| < \delta \Rightarrow |f(x) - f(x')| < \epsilon$$

According to [31], a poor robustness radius may indicate a use of the component in an unstable decision area hence a potential erroneous inference. This instability can typically be exploited by errors adversarial attacks. For our use case this may indicate that slight modifications of the image may change its classification.

Numerous methods propose to perform at runtime a formal robustness checking centered on the received input. Many of these methods are founded on the abstract interpretation that provides an over-approximation of the ML-component output domain for a given input domain i.e., the studied robustness ball. The main challenge encountered by these methods is the computation resources and the computation time needed to compute the output domain. We can especially consider the works [32] proposing GPU-based implementation of the abstract interpretation to speed-up the analysis process (less than 1s second for MNIST). Other works like [33] promote new abstract domains to increase both the accuracy and efficiency of the computation (1-100s for MNIST). Such approaches suffer from over-approximation, thus may consider that an input is non-robust whereas it is.

Quite different approaches like [34] provide some statistical robustness guarantees of local robustness. These methods usually rely on the sampling of a set of inputs within the studied robustness ball. Such methods could provide a quite simple way to assess the local robustness. But as the authors of [35] has identified, the main problem is to justify the representativeness of the computed statistical bounds since the sampling must be performed in conformity with the operational input distribution.

To apply these techniques, one may specify the requested local model stability that will be monitored at runtime. At the best of our knowledge, very little works propose methods to identify the requested robustness radius. One may consider that the robustness radius should be derived from the specifications of the ML component. Nevertheless, in many cases the selection of a robustness radius is difficult to argue and even more to relate to the specification. Hopefully, the authors of [31] propose a method to identify the expected robustness radius out of the training and test data sets. The capability of the requested robustness radius to identify instability areas at inference time has been assessed in their experiments.

Just like OOD monitoring, the performance of robustness-based runtime monitoring is paramount to ensure that a safety benefit can be achieved with an acceptable impact on system's availability. To address this aspect, we propose a simple formulation of the safety/availability tradeoff problem and illustrate it thanks to the experiments made by [31].

3) Safety/Availability Tradeoff

Let *S* be a simple system containing: ml, a ML component (e.g., a classifier) and m a mechanism validating (or rejecting) ml decisions (*e.g.,* an external monitor). We consider that m can only be *triggered* (*i.e.,* the prediction of ml is not trusted) or not (*i.e.,* the prediction of *ml* is trusted).

So, for a given input the system generates three possible outcomes:

- a correct output
- an erroneous-safe output, which covers a) the situation where ml generates an incorrect but safe output while *m* has not triggered, and b) the situation where the monitor triggers.
- an erroneous-unsafe output.

Let us assume that the ML system is free of implementation errors and is executed on a error-free hardware. Even with a perfect implementation and hardware we consider here that:

a) *ml* can produce an incorrect output; b) the occurrence of an error (*e.g.,* an OOD input), denoted *F*, may prevent *ml* to properly process the input.

In addition, we consider that *m* is unable to detect any failure caused by something else than *F*. This assumption is conservative since a monitoring mechanism, typically output monitoring, *may* also detect other errors like implementation or hardware errors.

The issue is to find the appropriate *sensitivity level* for *m* allowing complying with both safety and availability requirements. To assess the appropriate sensitivity level, one must formalize the notion of *unsafe* and *unavailable* system.

Let us consider that *ml* can either produce a correct result, fail safely or unsafely (event U_{ml}) and *m* can either be triggered (denoted *T*) or not. Let

- $P(U_{ml}|\neg F) = \alpha_{ml}$ be the conditional probability that the component fails unsafely knowing the failure *F* does not occur.
- $P(F) = \lambda$ be the probability of occurrence of *F* on demand
- $P(T|\neg F) = \delta$ be the false positive rate
- $P(\neg T|F) = \gamma$ be false negative rate.

a) General case

The scenarios leading to an *unsafe failure* (Unsa) are (a.1) *F* occurs and (a.2) *m* fails to detect it and (a.3) *ml* fails unsafely or (b.1) *F* does not occur and (b.2) *ml* fails unsafely and (b.3) *m* does not spuriously detect *F*, *i.e.*

$$\begin{aligned} P(Unsa) &= P(F, U_{ml}, \neg T) + P(\neg F, U_{ml}, \neg T) \\ &= P(U_{ml}, \neg T|F)\lambda + P(U_{ml}, \neg T|\neg F)(1-\lambda) \end{aligned}$$

Similarly, *S* is *unavailable* (Unav) when *m* is triggered. Among these scenarios some are expected since *ml* must not be used when *F* occurs. So, we propose to consider the scenarios where *S* is unavailable even if *F* did not occur, that is:

$$P(Unav) = P(\neg F, T) = \delta(1-\lambda)$$

b) Simplifications

Correlation between U_{ml} and F. If *ml* is likely to produce an unsafe output when *F* occurs that is $P(U_{ml}, \neg T|F) \simeq P(\neg T|F)$, then:

$$P(Unsa) = \gamma\lambda + P(U_{ml}, \neg T|\neg F)(1-\lambda)$$

Correlation between U_{ml}, $\neg T$ and $\neg F$. Additionally if we consider that the ability of *ml* to produce an unsafe output when *F* does not occur is similar when *F* does not occur and the *m* confirms it, that is $P(U_{ml}|\neg F) \simeq P(U_{ml}|\neg T, \neg F)$ then:

$$P(Unsa) = \gamma\lambda + \alpha_{ml}(1-\delta)(1-\lambda)$$

Robustness monitoring. Let us consider that *F* is an *input in an unstable area for ml*. We will consider that the *ml* component is very likely to produce an erroneous output if *F* occurs. Additionally, we assume that the robustness monitor does not significantly reject robust data.

Hence one may use the following formula for Unsa and Unav:

$$P(Unsa) = \gamma\lambda + \alpha_{ml}(1-\delta)(1-\lambda)$$
$$P(Unav) = \delta(1-\lambda)$$

Concerning abstract interpretation-based monitoring; some numerical values can be extrapolated from the Figure 1a of [31]. Let us consider a robustness radius of 10^{-2}, if we assume that their experiments (made over 100 images) can be generalised then we have:

$$P(F|\neg U_{ml}) = 3.6 \cdot 10^{-2} \qquad P(F|U_{ml}) = 6.3 \cdot 10^{-1}$$

The initial accuracy of the FNN-MNIST is 95.8%, let us consider that any misclassification is unsafe we have:

$$P(U_{ml}) = 4.2 \cdot 10^{-2}$$

One may estimate λ as follows:

$$\begin{aligned} P(F) &= P(F|U_{ml})P(U_{ml}) + P(F|\neg U_{ml})\big(1 - P(U_{ml})\big) \\ &= 6.1 \cdot 10^{-2} \end{aligned}$$

The performance of the ML model without F can be computed as follows:

$$\alpha_{ml} = P(U_{ml}|\neg F) = P(\neg F|U_{ml})\frac{P(U_{ml})}{P(\neg F)} = 1.65 \cdot 10^{-2}$$

Let us consider that we are using the approximate robustness radius computation whose performance are depicted on the Figure 1b of [31]. This monitor will compute an under-

approximation of the exact robustness ball (i.e., it is a pessimistic monitor) so there are only false positives w.r.t. F that is images that are rejected even if their true robustness is above the threshold. So, we have

$$\alpha_{ml} = 1.65 \cdot 10^{-2} \qquad \delta = 1.5 \cdot 10^{-2}$$
$$\lambda = 6.1 \cdot 10^{-2} \qquad \gamma = 0$$

The resulting unavailability and safety measures are:

$$P(Unsa) = 1.53 \cdot 10^{-2} \qquad P(Unav) = 1.4 \cdot 10^{-2}$$

Thus, according to the experiments of [31], adding the online robustness monitoring enhances slightly the integrity ($P(U_{ml})$ / $P(Unsa)$ ratio is approximately 2.7) with a one percent *availability* loss. In this example, the measures quantify the probability of misclassification (and lack of classification) per image. Let us translate these results for the use-cases of the section II.B. For the signal recognition system, the measures are expressed per signal. If we assume that the errors are not correlated to the signal's color, we obtain an error rate of $1.53 \cdot 10^{-2}$ per red signal that is higher than the expected 10^{-2} rate. For the automotive use case, the resulting error rate with a very simple vehicle model is $7.6 \cdot 10^{-1}$ per hour that is not in the same order of magnitude than the expected error rate. Note that these numerical values have been obtained through experiments considering idealized assumptions [31] and there is no conclusive evidence that such detection performances could be met during the operation.

Even if the previous assessment of a robustness monitoring technique is overly simplistic, it provides and illustrates a simple method to assess the contribution of monitoring techniques to the safety-availability trade-off. This illustration shows that monitoring slightly improves the safety of the system but also affects the system's availability in a non-negligible way.

C. Reject option and conformal prediction

We now describe two other mechanisms that allow to detect or temper ML errors, by either filtering out some inputs (classification with a reject option) or by outputting less informative predictions (conformal prediction). From a system architecture viewpoint, the reject option is similar to OOD monitoring (it filters out inputs that can lead to erroneous predictions), though the reject option is meant to be used for typical (in-distribution) yet hard-to-classify inputs.

1) Classification with a reject option

This setting, also known as *selective classification*, is an extension of the multi-class classification setting and has been studied for many decades; e.g., [36], [37], [38].

Given a new input X (e.g., an image), the goal is to predict the associated label Y out of several possible labels. An algorithm with *reject option* can decide to predict or not. More formally, such an algorithm is given by a *selective function* g and a *classification model* f. When $g(X) = 1$ the algorithm predicts the unknown label Y with $f(X)$. When $g(X) = 0$, the algorithm refrains from predicting. There are two competing objectives, which correspond to the reliability-availability trade-off:

- minimize the *selective risk*, i.e., the average number of errors when the ML algorithm predicts:
$$P(Y \neq f(X) | g(X) = 1)$$
- maximize the *coverage*, i.e., the proportion of inputs for which the ML algorithm outputs a prediction:
$$P(g(X) = 1)$$

For instance, in the railway use-case described earlier, the algorithm could predict or not predict the state Y of the light signal on the input image(s) X. A small selective risk corresponds to making few errors among the predicted light signals. A large coverage corresponds to predicting most light signals, leading to a large availability of the ML system.

Intuitively, there is a trade-off between selective risk and coverage: we can reduce the selective risk by rejecting hard-to-classify inputs x, but this also reduces coverage. This trade-off has been studied theoretically for binary labels and sometimes simple models (e.g., [37], [39], [40], [41]), but also empirically for multiple labels and deep learning models. For instance, on classical benchmarks such as CIFAR-10 or Cats vs. Dogs, empirical results from [42], [43] typically show an improvement of up to a factor of 2 in the risk when coverage is reduced to around 90 – 95%, and up to a factor of 20 for around 70% coverage. For other datasets such as CIFAR-100, SVHN, and ImageNet, the empirical results from [42] are of the same order of magnitude, but the trade-off is less favorable: a smaller coverage is needed to achieve the same risk reduction. In any case, even on CIFAR-10 for which the SelectiveNet algorithm [43] is reported to reduce the risk from around 6.7% at full coverage to around 0.3% at around 70% coverage, the value of 0.3% is still orders of magnitude larger than what would be required from a safety perspective if the ML algorithm errors could not be compensated in a drastic way. Therefore, the reject option is likely not a sufficient solution to fill the gap between safety objectives and ML predictive performances, though it should be considered as an interesting component towards ML error reduction.

We should also note that in many safety-related applications some errors may be considered more costly than others (i.e. carry a higher risk). For example, in our railway use-case, if the problem is cast as a binary classification (detect whether the image contains a red signal or not), then the type II error (missing a red signal) has a far higher risk than the type I error (falsely reporting a non-existent red signal). The theoretical formalism presented above can be extended to take into account this distinction between the types of errors by associating a different cost (also known as risk or loss) with each one. The new risk minimization criterion then reads:

$$\text{minimize} \quad P(f(X) \neq red, Y = red | g(X) = 1) + \lambda P(f(X) = red, Y \neq red | g(X) = 1)$$

where λ is the relative cost of making a type I error as a fraction of the cost of making a type II error. The coverage criterion we seek to maximize remains unchanged. Statistical guarantees for this asymmetric cost or risk formulation have been explored in the literature and have been successfully applied to the medical domain [39].

2) Conformal prediction

Conformal prediction is another way to reduce the risk. It consists in post-processing an ML algorithm and predicting a set $C(x)$ of possible labels for each new input x, instead of a single prediction $f(x)$. Now, predictions are made at all times, but the fact that $C(x)$ can contain more than one element is a way to reduce the probability of making an error: $P(Y \notin C(X))$.

In our railway use-case, this means we allow the ML system to make predictions with less information (e.g., this light signal can be this or that), but with the benefit of making fewer errors. This can prove useful as long as labels in the predicted set $C(x)$ do not often correspond to very different decisions.

Several algorithms have been proposed; see, e.g., [44], [45]. Typically, $C(x)$ is defined as the set of labels with highest probability scores at the output of a deep learning model, using

a data-driven threshold, and possible regularization when tail probabilities are poorly calibrated [46].
The aforementioned methods enjoy theoretical guarantees that are typically of the following form: given a risk level $\alpha \in (0,1)$, we can tune the `size' of $C(X)$ so that

$$P\big(Y \notin C(X)\big) \leq \alpha$$

Importantly, this guarantee relies on statistical assumptions (i.i.d. or exchangeable data), the verification of which can be a very difficult task. Furthermore, the probability above is an average error which does not imply a guarantee for each image, but a guarantee for most images (as those seen in the calibration dataset).
The fact that virtually any ML model can be conformalized (as a post-processing step) to yield a probabilistic guarantee as above makes conformal prediction an appealing technique for safety purposes. This field is currently receiving a lot of attention from the ML community, and we hope efforts will be made to incorporate safety-specific considerations.

D. Temporal redundancy

If we focus on the case of systems that may process sequences of inputs (e.g., consecutive video frames), it is natural to consider exploiting temporal redundancy to consolidate decisions or to exclude sporadic errors. This principle is applicable to ML-based systems, where consecutive inputs can be fed to a single ML model. These inputs are usually correlated; therefore independence cannot always be claimed. Nevertheless, a gain can be expected from such approach when consecutive inputs vary (for instance when the train is running).

In order to understand that gain, let us denote by X_t the correct classification at time t. Then the probability of two consecutive failures can be decomposed as follows:

$$P(\overline{X_t} \cap \overline{X_{t+dt}}) = P(\overline{X_t}) \times P(\overline{X_{t+dt}}|\overline{X_t})$$

If $P(\overline{X_{t+dt}}|\overline{X_t}) < 1$, that is to say if the inputs are not fully correlated, then the consideration of two consecutive inputs instead of one single input can increase the reliability.

Considering the ML component, two categories of events can affect single input performance:

- Extrinsic events concerning the environment of the system as perceived by its sensor (here, a camera). A typical example is the masking of the signal by a pole, or a bird, etc. that lead to a reduction of the recognition performance and possibly to an erroneous decision. So, by considering a series of predictions with an appropriate interval, the effects of these events will be filtered out and the capability to take a good decision will be improved.
- Intrinsic events concerning the capability of the system to take a good decision for some input. In that case, the event lasts as long as the input stays in the domain where the ML performance is bad. Temporal redundancy does not improve the capability to take a good decision.

The simplest approach to take advantage of the potential gain on extrinsic events is to use a voting scheme mechanism: the decision resulting of a sequence of predictions is taken as the majority vote of the single predictions.

Alternately, recurrent neural networks (LSTM or GRU [47] [48]) can also be used and show encouraging results for taking decision with time series. The principle is to feed over time a Neural Network with a feedback connexion. The input can either be the raw data (for instance the video sequence), or features extracted from each step in the sequence. The gain in sequence classification accuracy is of several percentage points. For instance, [49] for an action video classification use case obtain a 3% accuracy gain with LSTM compared to a voting scheme.

For safety purpose, we suggest an algorithm based on Finite State Machines (FSMs): each time a new input is received, the output of the ML classifier is used to update the state of a FSM, according to the logic depicted on *Figure 3*. This logic requires N ($N = 4$ for illustration purpose only on the figure) consecutive consistent classifications to make a decision. Otherwise, the output remains undefined.

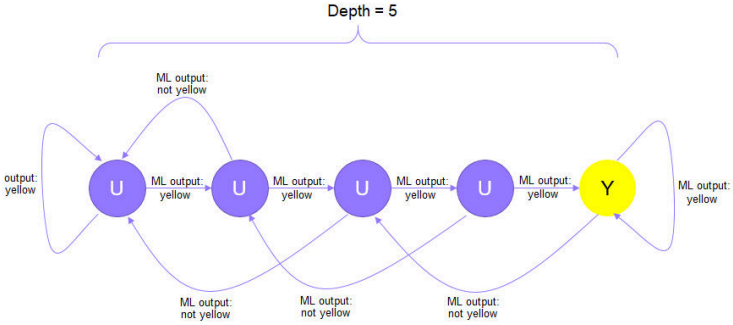


Figure 3: FSM for the classification of a yellow traffic light.

Temporal redundancy suffers from a correlation bias: it is not unusual to face situations where a ML model consistently misclassifies several consecutive inputs that are part of the same image sequence. Indeed, the variation of the input image depends essentially on the variation of the train pose. Unfortunately, if the speed of the train is low, or if the signal is close to the track, the variation of the pose will be low too. Additionally, the mechanism that extracts the relevant part of the image may also filter out variations of the image since it crops and scale it in order to keep the signal in a given bounding box.

In order to estimate the correlation bias and its impact, a simple testing approach is suggested. It consists in comparing the rate of occurrence of the following two events: (1) "N consecutive wrong outputs in the same sequence" and (2) "one wrong output". The former measures the failure rate with temporal redundancy, whereas the latter measures the failure rate without temporal redundancy. A reduced failure rate proves the added value in terms of reliability of this temporal redundancy mechanism.
Additionally, hybridization of various mitigation techniques could also be used. It consists in mixing the algorithm with other methods listed in this paper, in order to compensate for any identified weaknesses.

E. Concluding remarks

As we presented in this section, various methods can be considered (and even combined) to attempt to fill the gap identified between ML performances and safety requirements. Even if most of these methods rely to a certain degree on the independence hypothesis, which is often impossible to guarantee, these solutions still deserve a serious attention as their improvement and their potential combination could become sufficient in the future.
At ML level typically, structural redundancy remains interesting and could help exploiting the partial independence

of ML errors. Moreover, methods for monitoring ML components for instance, such as robustness or OOD monitoring, may be difficult to specify, but they also represent a promising direction to explore. However, for now, the variability of their performances and their poor reliability often negatively impacts the tradeoff between safety and availability. Other ML-based approaches may also be used, such as selective classification, or conformal prediction, which address the reliability-availability trade-off from a statistical viewpoint are able to provide specific statistical guarantees on ML predictions. Finally, when a system processes sequences of inputs, temporal redundancy can also be leveraged either through specific algorithms or ML architectures, to improve the overall results. However, here again, the fact that temporal independence cannot be guaranteed represents a serious impediment.

IV. DISCUSSION: A HISTORICAL PERSPECTIVE

A. Typical stages in the introduction of a new technology

At first sight, the performance gap stated above seems to indicate that ML-based systems are not able to be the core part of a safety-critical system at least for complex tasks [50] [51]. Nevertheless, looking carefully at the history of typical safety-critical systems, we can observe how new technologies can be introduced smoothly in safety-critical systems.

We can distinguish these main stages:

- First, the *bonus*: the new technology is only used to mitigate a risk that is not yet addressed at all, or to help address it beside some existing solution. The risks introduced by the technology itself are considered either negligible or mitigated by other means.
- Second, the *integration*: when the technology is used in the market, there are a lot of opportunities of feedbacks. In fact, there are two sides for this feedback: the overall risk reduction impact, and the adoption by the users. The feedback on the first aspect is given by the facts that some specific new risk could appear only during the operation phase, given that these new risks are (hoped to be) at a low level. This phase is the opportunity to accumulate data, in order to *quantify* the global balance between risk reduction and risk introduction. Meanwhile, the users have time to use the technology and to adopt it, to ask for modification or to reject it. It could be that he misuses it, too, which could generate a new type of risks, which will be added to the overall risk impact seen before.
- Third, the *acceptability change*: when a technology serving safety purposes is more and more used, it is more and more demanded, and can at some point become mandatory. It means that the mitigation of the risk that was an option at the beginning, becomes a requirement, which is asked by some *norms*. Afterwards, it could be that the acceptability level is raised (i.e., the failure rate has to be lowered).

In order to figure out this three-phases safety related systems evolution, let's take the example of the ABS (Anti-blocking Brake System) in the automotive field. In the last seventies, this system was introduced as an option in the premium cars. The market proposition was to reduce the risk of slipping because of wheels blocking due to a too strong braking regarding the road condition (e.g., driving on snow or very wet road). At this moment in the car's history, the road users accept the fact that, in certain conditions, a car could slip when the driver brakes too hard. Then having an equipment that has the ability to avoid this slipping situation was really a safety improvement, a *bonus* to the safety. At the same time, if this new equipment failed by not avoiding the slipping, then it was not less safe than the accepted current situation. Obviously, it was introduced from the beginning by taking care at least as safely as before to the other risks like keeping the ability to brake or to move (i.e., not blocking or releasing the brake unintendedly).

After this phase, it was the phase of *integration*, in the 80's and 90'. The product was more and more used, allowing to know how it really helped the drivers to overcome slipping situations, and what it brought as new effects, that were not expected in the first versions of the new product. For example, the first ABS equipments were not usable on roads with cobblestones, because of the periodic loss of wheel-road contact, which could be unfortunately at the same rhythm that the ABS order to release the brake: this behavior was not expected and was discovered with the usage on the road. More largely, this usage phase allowed to measure the impact of the ABS on the overall road safety, and to quantify all the expected and unexpected benefits and losses, and their balance.

This integration phase resulted in the *acceptability change* phase. This system has clearly been adopted by the users, and the balance of risks is positive. For this reason, for example in Europe in 2004, the ABS equipment was made mandatory by the regulation. The promulgation of this regulation shows that at this time the society considered that the ABS system was adopted by the users and had a positive risk balance. But it shows one more thing: it was not anymore accepted by the society that the accident due to slipping on the road because of a too strong braking application; in other words, the acceptability bar on the danger of slipping has been raised. It means that the risks that were accepted as *normal risk* in the early 80's, were not anymore accepted, and need to be mitigated, in the 2000's.

B. Are such stages applicable to ML?

In the case of ML-based systems, it is obviously too early to predict that this technology will follow this kind of cycle. In particular, the current rapid evolution of these technologies necessarily modifies the way they are integrated and adopted, since no sooner is the technology on the market than it is rendered obsolete by the next version. Furthermore, the nature of the products that can embody ML is much broader than in the case of ABS where the product is confused with the technology.

Having said that, the interesting point is that one can envisage ML-based products that improve overall safety in a domain, even if it is not a safety critical product. The underlying principle is that the new technology will not do worse than the existing one, and therefore if the existing one is acceptable then the new technology will be acceptable. This principle has obvious limitations related to the fact that important changes can also generate fears, and therefore the perceived risk could be different from the real risk. But let's leave that aside and take a few examples of what ML can bring today in a safety critical context.

The anticipation aids that ML can provide are an interesting category. They are based on two principles: it is about helping a human operator (pilot, driver, etc.), and therefore the operator will fully play his role as the main risk mitigation means; it is about helping anticipation, and therefore a potential failure of the system is far in the causal chain from the realization of the risk, which makes it completely legitimate to rely on the operator as a risk mitigation (i.e., he will have time and ability to react). In this case, as in all the others, we remind that we assume that the technology introduced to help mitigate a risk does not increase any other risk simultaneously. Incidentally, this category corresponds to

the category 1A that the EASA has identified in their recently released roadmap [2] as the first to be addressed.

As noted above, how precisely these technologies will be integrated into products, and even more how they will influence their acceptability, is unknown at this time. Typically, it seems unreasonable to think that their integration into products will fundamentally improve their reliability to such an extent that the orders of magnitude are changed, and that the compatibility of this criterion with the requirements of safety critical systems is made possible. But this conjecture is not to be totally rejected either, because an operational life can allow the development of a way to select training data whose representativeness and quantity would allow to reach this necessary reliability.

These considerations lead us to formulate a line of research. It appears that the cycle described above is not formalized at all today. It seems interesting to study a formalization so that these system evolutions can be anticipated, or even programmed, and that this would allow the identification of clear steps for a system to become "safety-critical", or even, why not, to obtain a continuum between non-critical systems and critical systems.

V. Conclusion

In this paper we address the complex problem of integrating ML predictive models into safety-critical systems. Through the lens of two practical use-cases we highlight the discrepancy between the performances of current ML models and the acceptable failure rates required by the industrial safety standards.

We observe that initiatives that are trying to propose adjustments of current practices to produce safety-critical software do not address the failure rate question. Therefore, we present several techniques from both domains (ML and safety) and analyze their potential application or extension to address the challenges raised by this assessment.

Throughout our analysis we note that many of these problems can be viewed as a reliability-availability trade-off and each of these techniques can address it from a different perspective.

We further investigate analogies with current practices and norms, as well as historical perspectives on the introduction of pioneering technical innovations.

While we take inspiration from many such precedents, we conclude that none of the enumerated techniques or norms offer satisfactory solutions, at least for now. The introduction of ML components in safety-critical systems remains an open question very much.

Nevertheless, we argue that ML can still contribute to the safety enhancement of current critical systems when implemented as "smart assistant solutions", which address otherwise unmitigated risks while ensuring they do not introduce additional ones (e.g., without any negative impact on the controllability of the system or the human capacity required for safe operation).

In addition to these technical aspects, our aim is to bring together the two communities (ML and safety), in order to build a common and solid foundation for the engineering of future intelligent safety systems. We hope that the discussions and the research directions presented here will motivate other contributors in this challenging endeavor.

VI. Appendix: definitions of key concepts

To help the reader unfamiliar with safety terminology, but also to avoid any ambiguities, we provide definitions of key safety concepts below. These definitions tend to be as generic as possible in order to be field-independent and focus on principles more than on normative (implementation) details. In fact, this paper is mainly about transportation systems, trying to be independent of its type (aeronautical, railway, automotive). We also refer the reader to [52] for further definitions.

1. **Failure**: Inability of a system or component to perform required function according to its specification and may have severe consequence on its usage.
2. **Risk**: in this document we only deal with safety risk. A safety risk is the potentiality of a system to provoke some injuries or even death to a person, due to its **failures** or insufficiencies. It is described essentially by its **severity** and **frequency** and can be associated to their mathematical product.
3. **Severity**: the impact level of a **risk**, in terms of number of deaths, number or type of injuries, number or type of other effect leading indirectly to injuries or death. The severity is a discrete (resp. scalar) value, on a finite (resp. bounded) set of values.
4. **Frequency**: the number of occurrences of the **failures** associated to a **risk** in a given time unit. This measured frequency is the **reliability**.
5. **Acceptability**: the fact that a society is keen to authorize the use of a product, because the residual **risk**s are considered sufficiently low (i.e., under the acceptability level).
6. **Risk mitigation means**: the means to decrease a **risk**, by acting on its **severity** or its **frequency**. It can be technical, organizational, or procedural.
7. **Norm/Standard**: a norm or a standard is a reference document (or set of documents) where the **acceptability** level is defined, and the recognized **risk mitigation means** are described. This document is written by a community of people that agree on what they accept as risk and what they do not accept, given the usage of a product. For example, the International community defines through the ISO 26262 what they accept as risk regarding the failures of the electric and electronic equipments for road vehicles (See [11] [12] [13] [14] [15] [16] [17] [18] [19] [53]).
8. **Error**: the occurrence of the state of a part of the system which is not compliant to the specified or intended state. An error can be the unique or the partial cause of a failure (or causes no failure at all).
9. **Exposure**: one aspect of the **frequency** of a failure, to help quantify operational situations for which it can occur. It allows to treat differently the rare and the frequent situations, from a safety point of view. This parameter is more used in the automotive field than in the others; the smaller the exposure, the smaller the frequency.
10. **Controllability**: the ability for a user of the product to avoid the dangerous situation provoked by a **failure**, or at least to decrease its effects. This parameter is more used in the automotive field than in the others; the higher the controllability, the smaller the frequency.

VII. REFERENCES

[1] H. Delseny, C. Gabreau, A. Gauffriau, B. Beaudouin, L. Ponsolle, L. Alecu, H. Bonnin, B. Beltran, D. Duchel, J.-B. Ginestet, A. Hervieu, G. Martinez, S. Pasquet, K. Delmas and Pag, "White Paper Machine Learning in Certified Systems," 2021.

[2] J. M. Cluzeau, X. Henriquel, G. Rebender, G. Soudain, L. van Dijk, A. Gronskiy, D. Haber, C. Perret-Gentil et R. Polak, *Concepts of Design Assurance for Neural Networks,* 2020.

[3] F. T. Laviolette, L. Gabriel, A. Le, N. Amin, S. N. M. Paulina, P. Yann, K. Foutse, A. Giulio and M. Ettore, *How to Certify Machine Learning Based Safety-critical Systems? A Systematic Literature Review,* 2021.

[4] M. Borg, C. Englund, K. Wnuk, B. Duran, C. Levandowski, S. a. T. Y. Gao, H. Kaijser, H. Lönn et J. Törnqvist, «Safely Entering the Deep: A Review of Verification and Validation for Machine Learning and a Challenge Elicitation in the Automotive Industry,» *Journal of Automotive Software Engineering,* n° %11, pp. 1-19, 2019.

[5] R. Salay et K. Czarnecki, *Using Machine Learning Safely in Automotive Software: An Assessment and Adaption of Software Process Requirements in ISO 26262,* 2018.

[6] A. B. Arrieta, N. Díaz-Rodríguez, J. Del Ser, A. Bennetot, S. Tabik, A. Barbado, S. García, S. Gil-López, D. Molina, R. Benjamins et et al., «Explainable artificial intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI,» *Information Fusion,* vol. 58, pp. 82-115, 2020.

[7] C. Molnar, Interpretable Machine Learning, 2019.

[8] A. Chakraborty, M. Alam, V. Dey, A. Chattopadhyay et D. Mukhopadhyay, «Adversarial Attacks and Defences: A Survey,» *CAAI Transactions on Intelligence Technology,* vol. 6, 2021.

[9] A. Mehta et S. Kumar, «A Survey on Resilient Machine Learning,» 2017. [En ligne]. Available: https://arxiv.org/abs/1707.03184.

[10] D. Carvalho, E. V., M. Pereira et J. Cardoso, «Machine learning interpretability: A survey on methods and metrics,» *Electronics,* vol. 8, n° %18, 2019.

[11] SAE/EUROCAE, *ARP4754A/ED-79A, Certification considerations for highly-integrated or complex aircraft systems,* 2010.

[12] SAE/EUROCAE, *ARP4761/ED-135, Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment,* 1996.

[13] RTCA/EUROCAE, *ED-80/DO-254, Design Assurance Guidance for Airborne Electronic Hardware,* 2000.

[14] RTCA/EUROCAE, *DO-178C/ED-12C, Software considerations in airborne systems and equipment certification,* 2012.

[15] IEC, *61508, Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems,* 2010.

[16] CENELEC, *EN-50127, Railway Applications - The Specification and Demonstration of Reliability, Availability, Maintainability and Safety (RAMS),* 2017.

[17] CENELEC, *EN-50128, Railway applications - Communication, signalling and processing systems - Software for railway control and protection systems,* 2020.

[18] CENELEC, *EN-50129, Railway applications - Communication, signalling and processing systems – Safety related electronic systems for signalling,* 2020.

[19] ISO, *26262, Road vehicles -- Functional safety,* 2018.

[20] H. Mania, J. Miller, L. Schmidt, M. Hardt and B. Recht, "Model Similarity Mitigates Test Set Overuse," in *Advances in Neural Information Processing Systems 32 (NeurIPS 2019)* , 2019.

[21] J. Nixon, B. Lakshminarayanan and D. Tran, "Why Are Bootstrapped Deep Ensembles Not Better?," in *ICBINB Workshop at NeurIPS 2020*, 2020.

[22] R. Gontijo-Lopes, Y. Dauphin and E. D. Cubuk, "No One Representation to Rule Them All: Overlapping Features of Training Methods," 2021.

[23] SAE, *J3016, Taxonomy and Definitions for Terms Related to On-Road Motor Vehicle Automated Driving Systems,* 2018.

[24] G. Pang, C. Shen, L. Cao et A. V. D. Hengel, «Deep learning for anomaly detection: A review,» *ACM Computing Surveys (CSUR),* vol. 54, n° %12, pp. 1-38, 2021.

[25] R. Lukas, R. K. Jacob, A. V. Robert, M. Grégoire, S. Wojciech, K. Marius, G. D. Thomas et M. Klaus-Robert, *A unifying review of deep and shallow anomaly detection,* 2021.

[26] C. Liu, T. Arnon, C. Lazarus, C. Strong, C. Barrett et M. J. Kochenderfer, *Algorithms for verifying deep neural networks,* 2019.

[27] M. Breunig, H.-P. Kriegel, R. Ng et J. Sander, Lof: identifying density-based local outliers, ACM SIGMOD international conference on Management of Data, 2000.

[28] J. Hansi, W. Haoyu, H. Wenhao, K. Deovrat et C. Arin, Fast incremental svdd learning algorithm with the gaussian kernel, AAI Conference on Artificial Intelligence, 2019.

[29] L. Ruff, J. R. Kauffmann, R. A. Vandermeulen et G. Montavon, A unifying review of deep and shallow anomaly detection, IEEE, 2021.

[30] S. Alireza, S. Mark et J. L. James, A Less Biased Evaluation of Out-of-distribution Sample Detectors, arXiv:1809.04729, 2019.

[31] J. Liu, L. Chen and A. Miné, "Input validation for neuralnetworks via runtime local robustness verification," in *CoRR*, 2020.

[32] C. Müller, F. Serre, G. Singh, M. Püschel et M. Vechev, «Scaling polyhedral neural network verification on gpus,» chez *Proceedings of Machine Learning and Systems*, 2021.

[33] M. Niklas, C. Müller, G. Makarchuk, F. Serre, G. Singh, M. Püschel et M. Vechev, «Prima: Precise and general neural network certification via multi-neuronconvex relaxations,» chez *arXiv preprint arXiv:2103.03638*, 2021.

[34] J. Cohen, E. Rosenfeld et Z. Kolter, «Certified adversarial robustness viarandomized smoothing,» chez *International Conference on Machine Learning*, 2019.

[35] A. Fromherz, K. Leino, M. Fredrikson, B. Parno et C. Pasareanu, «Fast geometric projections for local

robustness certification,» chez *International Conferenceon Learning Representations*, 2020.

[36] C. K. Chow, "An optimum character recognition system using decision functions," *IRE Transactions on Electronic Computers,* pp. 247-254, 1957.

[37] C. K. Chow, "On optimum recognition error and reject tradeoff," *IEEE Transactions on Information Theory,* pp. 41-46, 1970.

[38] M. E. Hellman, "The Nearest Neighbor Classification Rule with a Reject Option," *IEEE Transactions on Systems Science and Cybernetics,* vol. 6, no. 3, pp. 179-185, 1970.

[39] R. Herbei and M. H. Wegkamp, "Classification with Reject Option," *The Canadian Journal of Statistics / La Revue Canadienne de Statistique,* vol. 34, no. 4, pp. 709-721, 2006.

[40] R. El-Yaniv and Y. Wiener, "On the Foundations of Noise-free Selective Classification," *Journal of Machine Learning Research,* vol. 11, no. 53, pp. 1605-1641, 2010.

[41] J. Lei, "Classification with confidence," *Biometrika,* vol. 101, no. 4, pp. 755-769, 2014.

[42] Y. Geifman and R. El-Yaniv, "Selective Classification for Deep Neural Networks," in *Proceedings of NeurIPS 2017*, 2017.

[43] Y. Geifman and R. El-Yaniv, "SelectiveNet: A Deep Neural Network with an Integrated Reject Option," in *Proceedings of ICML 2019*, 2019.

[44] Y. Romano, M. Sesia and E. Candes, "Classification with Valid and Adaptive Coverage," in *Proceedings of NeurIPS 2020*, 2020.

[45] M. Cauchois, S. Gupta and J. C. Duchi, "Knowing what You Know: valid and validated confidence sets in multiclass and multilabel prediction," *Journal of Machine Learning Research,* vol. 22, no. 81, pp. 1-42, 2021.

[46] A. Angelopoulos, S. Bates, J. Malik and M. I. Jordan, "Uncertainty Sets for Image Classifiers using Conformal Prediction," in *Proceedings of ICLR 2021*, 2021.

[47] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation,* vol. 9, no. 8, p. 1735 1780, 1997.

[48] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk et Y. Bengio, «Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation,» 2014.

[49] M. Baccouche, F. Mamalet, C. Wolf, C. Garcia et A. Baskurt, «Sequential Deep Learning for Human Action Recognition,» chez *2nd International Workshop on Human Behavior Understanding (HBU)*, Amsterdam, Netherlands, 2011.

[50] G. Katz, G. W. Barrett, D. L. Dill, K. Julian et M. J. Kochenderfer, «Reluplex: An efficient SMT solver for verifying deep neural networks,» chez *CoRR*, 2017.

[51] M. Damour, F. De Grancey, C. Gabreau, A. Gauffriau, J.-B. Ginestet, A. Hervieu, T. Huraux, C. Pagetti, L. Ponsolle et A. Claviere, «Towards Certification of a Reduced Footprint ACAS-Xu System: A Hybrid ML-Based Solution,» chez *International Conference on Computer Safety, Reliability, and Security (SAFECOMP)*, 2021.

[52] A. Avizienis, J. Laprie and B. Randell, "Fundamental Concepts of Dependability," 2000. [Online]. Available: https://www.researchgate.net/publication/2408079_Fundamental_Concepts_of_Dependability.

[53] ISO, *PAS 21448, Road vehicles - Safety of the intended functionality (SOTIF),* 2019.

Session Th.1.B

Security

Thursday 2nd June

10:00

–

Room Lauragais

Hijacking an autonomous delivery drone equipped with the ACAS-Xu system

Adrien Gauffriau*†, David Bertoin†§, Jayant Sen Gupta‡†
*Airbus Operations, † IRT Saint-Exupery, ‡Airbus AI Research, §Institut de Mathématiques de Toulouse

***Abstract*—In this paper, we want to show that automated anti-collision systems in aeronautical industry such as ACAS-Xu are vulnerable to hijacking threats in a urban environment which is less controlled than conventional airspace. Using reinforcement learning methods, we demonstrate the possibility to hijack the mission of a delivery drone equipped with the ACAS-Xu system in a simulated environment. Our objectives are first, to illustrate the security (interception) vulnerabilities of autonomous system and secondly, to enrich reinforcement learning benchmarks with a new one that comes from an industrial aeronautical application.**

***Keywords*—Autonomous and connected systems, Resilience, Artificial Intelligence, Reinforcement Learning, Security, ACAS-Xu**

I. Introduction

Autonomy is one of the hot topics where the potential of artificial intelligence, both for perception and decision making tasks, opens new possibilities. Nevertheless, autonomous systems also raise public acceptance and certification challenges. If autonomous cars are one of the best known examples, we see the emergence of autonomous systems in aeronautics, including delivery drones, autonomous air cabs, or airplanes [16]. This paper will focus on these particular systems.

Whereas most important concerns of the aeronautic certification are dependability and safety, the ability of a system to resist malevolent attack is also a major issue. Our work mainly focus on this topic with the development of an attack (interception) of an avionics system. Aviation security is mostly focused on ensuring that airports are secured by controlling people and goods passing through. Once the aircraft is flying, security is obtained by monitoring specific parts of the airspace all aircraft should respect, like airways. Airways are designed to ensure separation between pairs of aircraft, and any aircraft not respecting these rules is identified, tracked, and eventually neutralized. Nevertheless, what is valid to ensure security from malevolent attackers for standard aviation will no longer be applicable in the urban air mobility (UAM) context. Even if the concept of airway is extended in UAM, the distance to the ground, the distance between airways, and the potential number of threats will make it extremely difficult for humans to supervise all the traffic.

Systems like autonomous avoidance systems are thus needed to ensure, among others, the safety, and security of UAM. Nevertheless, new methods for attacks inspired by video game testing (for instance [2] and [18]) may change the paradigm that was traditionally used. In this work, we illustrate this statement by developing an attack on the ACAS-Xu avoidance system (see [10]) that will possibly be used by future autonomous vehicles.

We consider the following setting: an autonomous delivery drone (target), equipped with the ACAS-Xu system for collision avoidance, whose mission is to reach a predetermined delivery area, is attacked by another drone (attacker) that tries to hijack it and lead it to a different delivery area. When no risk of collision is detected, it follows the heading to the delivery area. When an aircraft (drone) enters the risk zone around the target, its heading is updated according to ACAS-Xu recommendation. The attacker policy (its strategy) is coded in a neural network which is trained to bring the target to the alternate delivery area using reinforcement learning (RL). In this paper, we want to show how vulnerable is a drone that uses a systematic way to avoid collisions. We limited our study to the horizontal recommendations of ACAS-Xu as it is preferred to deal with non-cooperative traffic (see [13]).

This paper is organized as follows. Section II presents the ACAS-Xu avoidance system. Section III presents the related works. Section IV introduces the principles of reinforcement learning (RL). Section V presents an experimental setup demonstrating the effectiveness of such attacks. Finally, section VII summarizes and concludes this paper on future perspectives.

II. Overview of the ACAS system

Among the family of ACAS X, the ACAS-Xu is dedicated to drones and urban air mobility. It provides a horizontal resolution for conflicts using in real-time a set of lookup tables (LUT) that were computed offline. Using geometric parameters, the ownship consults the LUT on collision's probability for five different advisories : COC (Clear of Conflict), WL (Weak Left), WR (Weak Right), L (Left), R (Right). The system does not rely on the fact that the intruder (the attacker in our setting) also applies the same avoidance system. Therefore, this system can also be used to avoid any static object (tower, crane) or birds as

The selected advisory is the one that minimizes the probability of conflicts. The geometry of a conflict is given in figure 1, the parameters definition stands as:

- ρ (ft): Distance from ownship to the intruder
- θ (rad): Angle to intruder relative to ownship heading
- ψ (rad): Heading angle of intruder relative to ownship heading direction
- v_{own} (ft/s): Speed of ownship
- v_{int} (ft/s): Speed of intruder

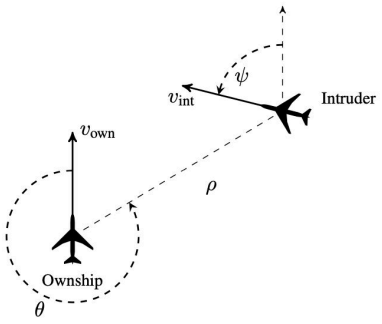


Fig. 1: ACAS-Xu geometry [20]

- τ (s): Time until loss of vertical separation
- R_{n-1}: Last recommendation provided by the ACAS-Xu

The 23 LUT provides the transitions costs between the previous advisory and the next advisory. When the ownship is not in the COC state, it has to initiate the turn given by the advisory. Otherwise it can continue its mission. More information on the ACAS-Xu system can be found in [25].

III. Related works

The use of RL methods to search for attack and interception scenarios is relatively classical. The originality comes from the use of these methods to find security weaknesses of an avionic systems. This will be illustrated in the following state of the art targeting the security in avionics, the safety of the ACAS-Xu system and the use of RL for unmanned aerial vehicle.

A. Security in avionics

The interest in the security of aeronautic systems is not new. In [39], the authors exploit vulnerabilities of the ACARS network to upload new flight plans in the Flight Management System (FMS) of aircraft. Nevertheless, the pilots still keep control of the aircraft, the attack mainly leads to increases in their workloads.

It is also possible to attack ground infrastructure like Instrument Landing System (ILS) [35]. The authors developed two different attacks of the ILS using radio signals. They demonstrate a systematic success rate with offset touchdowns of 18 meters to over 50 meters in lateral and longitudinal. Therefore, an attacked aircraft that performs a fully automatic landing miss the runway.

The literature is not limited to hacking the global system. [1] [34] focus on attacking aircraft networks. Nevertheless, most of these results remain theoretical or academic due to the complexity and cost of deploying such attacks. Moreover, none of these attacks enable complete control of the aircraft.

Even if there is an active research on safety in avionics, the results obtained are often mitigated by the presence of humans in the loop. Moreover, the popularization of drones and their accessibility has led to the emergence of new safety issues. In the future, the introduction of low complex autonomous systems may change this paradigm. Our contribution aims to make the avionic community aware of this future challenge.

B. ACAS-Xu

Detect and avoid is a task that guarantees the safety of flying vehicles, completed by the intervention of the pilot. For large aircraft, it is still the responsibility of the pilot that may be helped by systems that give advisory and recommendations for avoidance. The most famous one is the TCAS [31] that requires both planes to be equipped. The ACAS system [10] was developed for autonomous vehicles and mainly based on [22]. Several methods were explored to guarantee that this system is safe among Petri model [30] or formal methods [17]. The memory size (more than 4 GBytes) required by the look-up table of the ACAS-Xu system may be incompatible with the electronics of small drones. Thus, [19] developed training of neural networks that replace look-up tables with a small memory footprint. This raises the question of the safety of the neural network, which is still an open problem. [20] [7] propose formal methods to analyze neural networks and prove avoidance properties. Another approach [8] also based on formal methods, demonstrates that trained neural network behaves like the look-up table defined by the ACAS-Xu standard.

Our contribution aims to raise the security question of the ACAS-Xu system that differs from previous work that mainly focus on its safety. Up to our knowledge, this has never been done.

C. Reinforcement learning for Unmanned Aerial Vehicle

Recent successes in reinforcement learning have demonstrated the ability of reinforcement learning agents to outperform humans in many tasks [29], [36], [41], [42], [47]. Several recent works have sought to capitalize on the progress made in RL and Deep RL for the control and navigation of UAV. There are mainly two classical use cases of reinforcement learning for UAVs. The first one is flying control where the RL agents aim at providing stability and control and navigation. The second deals with mission planning where the agent is responsible for the high-level policy while control systems are implemented using classical methods such as Proportional-Integral-Derivative (PID) control systems [9].

While PID control has demonstrated excellent results in stable environments, it is less effective in unpredictable and harsh environments. Recently, several research projects have explored the possibility of using reinforcement learning to address its limitations. [49] compared the efficiency of a model based reinforcement learning controller with Integral Sliding Mode (ISM) control [52].

The authors of [15] trained neural-network policy for quadrotor controllers using an original policy optimization algorithm with Monte-Carlo estimates. The learned policy manages to stabilize the quadrotor in the air even under very harsh initialization, both in simulation and with a real quadrotor.

[21] train autonomous controllers flight control systems with state-of-the-art model free deep reinforcement learning algorithms (Deep Deterministic Policy Gradient [23], Trust

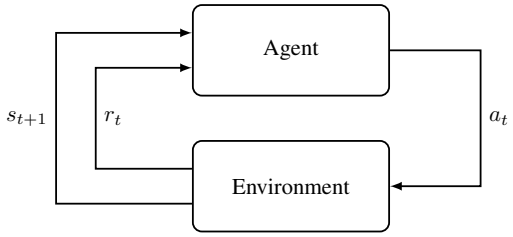


Fig. 2: Standard RL framework

Region Policy Optimization [37], Proximal Policy Optimization [38]) and compare their performance with PID controllers.

In [3], a sequential latent variable model is learned from flying sequences of an actual drone controlled with PID. This latent dynamic model is used as a generative model to learn a deep model-based reinforcement learning agent directly on real drones with a limited number of steps.

In [32], the authors combine a Q-learning [50] algorithm focusing on navigation policy with PID controllers. In [45], the navigation problem is decomposed into two simpler subtasks (collision-avoidance and approaching the target), each of them solved by a separate neural network in a distributed deep RL framework. An active field of research focuses on interception and defense against malicious drones. In a 1 vs 1 close combat situation, [48] demonstrates the effectiveness of an A3C [27] RL agent versus an opponent with Greedy Shooter policy [40]. In a multi-agent context, [51] uses a Multi-Agent Deep Deterministic Policy Gradient algorithm (MADDPG) [24] in an attack-defense confrontation markov game. [26] proposes a ground defense system trained with Q-learning to choose between high-level defense strategies (GPS spoofing, jamming, hacking, and laser shooting). While [12] and [5] use RL to train a drone attacker to intercept a target drone, [6] place the agent in the defender's position and train it with a Soft Actor-Critic algorithm [14] to avoid capture.

Our contribution also aims at intercepting a target UAV using an RL agent. However, it differs on two significant points. First, it highlights the security flaws of a deterministic policy dictated by the ACAS-Xu system for collision avoidance. Second, our attacker does not seek to capture the target directly but to guide it to a specific area where it can potentially be captured. This strategy does not require any attack equipment directly implemented on the attacking UAV and can easily be applied to any UAV.

IV. Reinforcement learning

Reinforcement Learning is a specific field of machine learning considering sequential decision-making problems. In RL, an *agent* interacts with its *environment* during a sequence of discrete time steps, $t = 0, 1, 2, 3, ...$ At each time step t, the agent is provided a representation of the environment *state* $s_t \in \mathcal{S}$, where $\mathcal{S}$ defines a state space. According to s_t, the agent takes an *action* $a_t \in \mathcal{A}$, where $\mathcal{A}$ is the set of all possible actions. Performing this action in the environment causes the environment to transition from s_t to s_{t+1}, and as a consequence of this transition, the agent receives a numerical *reward* $r_t \in \mathbb{R}$. Figure 2 illustrates the agent-environment interaction. The mapping of a state s to a probability of taking each possible action in $\mathcal{A}$ is called the agent's *policy* and denoted $\pi(a|s) = \mathbb{P}[A_t|S_t = s]$. Considering a discount factor $\gamma \in [0, 1]$, the return is defined as the discounted sum of rewards $R_t = \sum_{k=0}^{T} \gamma^{(k)} r_{t+k}$. Deep Reinforcement learning algorithms aim at finding the policy π_ϕ, represented by a neural network with parameters ϕ, that maximizes the expected return $J(\pi_\phi) = \underset{\tau \sim \pi_\phi}{\mathrm{E}}[R(\tau)]$ with $\tau = (s_0, a_0, ..., s_{T+1})$ the trajectory obtained by following the policy π_ϕ starting from state s_0. For continuous control problems (such as motor speed control), policy gradients methods aim at learning a parametrized policy π_ϕ through gradient ascent on $J(\pi_\phi)$. These methods rely on the policy gradient theorem [44]:

$$\nabla_\phi J(\phi) = \mathbb{E}_{s\sim\rho^\pi, a\sim\pi_\phi}\left[\nabla_\phi \log \pi_\phi(a \mid s) Q^\pi(s, a)\right],$$

where $Q^\pi(s, a) = \mathbb{E}_{s\sim\rho^\pi, a\sim\pi_\phi}[R_t|s, a]$ is the action-value function. $Q^\pi(s, a)$ represents the expected return of performing action a in state s and following π afterwards.

Policy gradients methods typically require an estimate of $Q^\pi(s, a)$. An approach used in *actor-critic* methods consists in using a parametrized estimator called *critic* to estimate $Q^\pi(s, a)$ (π_ϕ thus represents the *actor* part of the agent). By relying on this principle, [43] propose the deterministic policy gradient algorithm to compute $\nabla_\phi J(\phi)$:

$$\nabla_\phi J(\phi) = \mathbb{E}_{s\sim p_\pi}\left[\nabla_a Q^\pi(s, a)|_{a=\pi(s)} \nabla_\phi \pi_\phi(s)\right].$$

The DDPG algorithm [23] adapts the ideas underlying the success of Deep Q-Learning [28] [29] to estimate Q^π with a neural network with parameters θ. In DDPG the learned Q-function tends to overestimate $Q^\pi(s, a)$, thus leading to the policy exploiting the Q-function estimation errors. Inspired by the Double Q-learning [46], the Twin Delayed DDPG (TD3) [11] addresses this overestimation by taking the minimum estimation between a pair of critics and adding a noise to the actions used to form the Q-learning target. These tricks, combined with a less frequent policy update (one update every d critic updates) result in substantially improved performance over DDPG in a number of challenging tasks in the continuous control setting. Algorithm 1 describes TD3's complete training procedure.

V. Development of an attack on ACAS-Xu

A. Notations

The following notations are used in the next sections:

- D_1: Delivery area. The target's destination objective
- D_2: Alternate delivery area. The attacker destination objective: the interception zone
- I_t: Initial position of the target.
- I_a: Initial position of the attacker.
- v_t: Speed of the target that is constant.
- v_a: Speed of the attacker. v_a^{max} is the maximum possible speed of the attacker
- The target corresponds to the ownship of the ACAS-Xu

Algorithm 1: TD3

Initialize critic networks Q_{θ_1}, Q_{θ_2}, and actor network π_ϕ with random parameters θ_1, θ_2, ϕ
Initialize target networks $\theta'_1 \leftarrow \theta_1, \theta'_2 \leftarrow \theta_2, \phi' \leftarrow \phi$
Initialize replay buffer $\mathcal{B}$
for $t = 1$ to T **do**
  Select action with exploration noise $a \sim \pi_\phi(s) + \epsilon, \epsilon \sim \mathcal{N}(0, \sigma)$ and observe reward r and new state s'
  Store transition tuple (s, a, r, s') in $\mathcal{B}$
  Sample mini-batch of N transitions (s, a, r, s') from $\mathcal{B}$
  $\tilde{a} \leftarrow \pi_{\phi'}(s') + \epsilon$, $\epsilon \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}), -c, c)$
  $y \leftarrow r + \gamma \min_{t=1,2} Q_{\theta'_i}(s', \tilde{a})$
  Update critics
  $\theta_i \leftarrow \text{argmin}_{\theta_i} \frac{1}{N} \sum (y - Q_{\theta_i}(s, a))^2$
  if t mod d **then**
    Update ϕ by the deterministic policy gradient:
    $\nabla_\phi J(\phi) = \frac{1}{N} \sum \nabla_a Q_{\theta_1}(s, a)\big|_{a=\pi_\phi(s)} \nabla_\phi \pi_\phi(s)$
    Update target networks:
    $\theta'_i \leftarrow \tau\theta_i + (1 - \tau)\theta'_i$
    $\phi' \leftarrow \tau\phi + (1 - \tau)\phi'$
  end
end

- The attacker corresponds to the intruder of the ACAS-Xu. The attacker is not equipped with the ACAS-Xu.

B. Set up and objective

We consider an environment composed of two agents and two areas. The first agent is the delivery drone that has the mission to reach the delivery area D_1. The target (T) is equipped with the ACAS-Xu system to trigger autonomous avoidance actions by updating its heading. The second agent is the attacker (A) drone. The attacker aims at hijacking the target towards an alternate delivery area D_2, located in a different position than D_1 by exploiting the target's utilization of the avoidance ACAS-Xu system. The attacker's policy is trained for this purpose using Deep Reinforcement Learning. In our setting, for the sake of simplicity, both agents can only move in the same horizontal plan. Ascending and falling are not allowed. The figure 3 provides a graphical representation of the set-up.

The Xu version of the ACAS system is dedicated to drones, and only provides horizontal avoidance recommendations. Among the other version of the ACAS, it exists the Xa version dedicated to large aircraft that provides vertical avoidance like the TCAS. Thus, we restrict the interception to an horizontal plan since ACAS-Xu does not provide vertical avoidance recommendation. Even if, the target may have a vertical flight plan, it is very easy to configure, without reinforcement learning, the attacker to be always on the same horizontal plan of the target. In a future work, we may extend the study by considering a target equipped with the Xu and the Xa version and train an agent for a 3D interception.

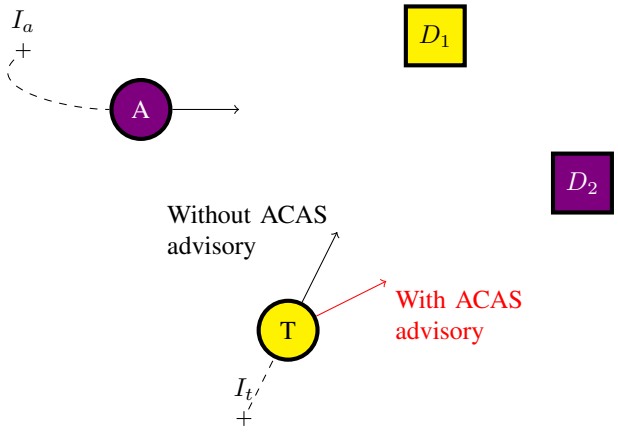


Fig. 3: Set up

C. Training environment

We implemented our training environment with the following settings:

a) State Space: The state of the environment is completely described by the state of the two agents (target and attacker) and the Cartesian positions of delivery areas. Each agent's state is composed of $P = (x, y)$ representing the agent Cartesian positions and a velocity vector $\vec{V} = (v_x, v_y)$ in the horizontal plan. The angle α of $\vec{V}$ is agent's heading.

b) Action Space: The attacker's actions at step n are represented by two updates $\lambda V_n \in [-200, +200]$ and $\lambda\alpha_n \in [-\frac{\pi}{2}, \frac{\pi}{2}]$ representing respectively an update for the velocity and heading.

c) Transitions: The target agent's velocity is constant during the whole episode. Its heading is updated according to the ACAS-Xu system advisory. If the advisory provided is different from COC, the following heading update $\delta\alpha_n$ will be used:

- WL : + 0.15 rad
- WR : - 0.15 rad
- L : + 0.3 rad
- R : - 0.3 rad

If the advisory is COC, the $\lambda\alpha$ update enables the target to reach the heading to the delivery area with a maximum variation of 0.3 rad. We limit this variation to be more representative of a real drone maneuverability and avoid instability due to big turns. For both agents, the update of the speed vector is given by

$$\|V_{n+1}\| = \|V_n\| + \delta V_n$$
$$\alpha_{n+1} = \alpha_n + \delta\alpha_n$$

and position update by $P_{n+1} = P_n + V_{n+1}$.

d) Reward model: The choice of the reward function is not trivial. The training capacity and future policy of the RL agents are deeply impacted by the reward model used during training. The reward model design, often called reward shaping, may lead to strange and unexpected behaviors.

$$R_n = \begin{cases} 0 & \text{if } n = 0 \\ R_{n-1} & \text{if } D_{n-1} \geq D_n \\ R_{n-1} + (D_n - D_{n-1}) & \text{if } D_{n-1} < D_n \end{cases}$$

with $D_n = \|P_n^T - P^E\|$. The reward is increased for each step that globally reduces the distance between the target agent and the interception destination. We used the state-of-the-art Gym [4] framework to implement our training environment.

VI. Experiments and results

This section presents the different scenarios of experiments conducted. In all experiments, the target has a fixed speed of 400 ft/s and is following ACAS-Xu avoidance recommendations in case of presence of another flying object in its vicinity. When there is no obstacle, the drone follows the direction leading to its delivery area.

A. Training setups

We train three different agents depending on the maximum speed we allow the attacker in order to study the influence of the speed ratio between the attacker and the target. Three configurations are tested:

- 300 ft/s
- 600 ft/s
- 1000 ft/s

We fix the size of the playground as a square of 100000 feet. We developed a gym environment for training purposes to simulate the target behavior, following the ACAS-Xu function, and train the attacker policy using our RL algorithm. For each training scenario, we randomly draw the positions of the delivery zone, the alternative delivery zone where the attacker has to bring the target, the initial position of the target, and the initial position of the attacker. At the end of this stage, we get three trained agents designated by $\mathcal{A}_{300}$, $\mathcal{A}_{600}$ and $\mathcal{A}_{1000}$.

In every setups, we trained an RL agent using Stable-baselines3 [33] implementations of TD3. In each experiment, the agent is constituted by an actor and two critics. We use a two-layer feedforward neural network of 400 and 300 hidden nodes respectively, with rectified linear units (ReLU) between each layer for both the actor and critic, and a final two neurons output layer with tanh for the actor.

TD3 is an *off-policy* algorithm, during training transitions $(s_t, a_t, s_{t+1}, r_{t+1})$ are stored in a *replay-buffer* [28] and drawn randomly in the form of mini-batches during the weight update phase. We conducted all of our experiments using a replay-buffer of size 50000 and a mini-batch size of 512. TD3 trains a deterministic policy in an off-policy way, which is not favorable for exploration during training. We encouraged exploration of the TD3 agent by adding an action noise drawn from the Ornstein-Uhlenbeck process, as suggested in [23]. For every scenario, we trained our agents on 6 million steps. Table I provides a complete description of the training parameters used during training.

Parameter	Value
Training steps	6,000,000
Learning rate	0.001
γ	0.99
Policy delay	2
τ	0.005
target policy noise	0.2
Ornstein-Uhlenbeck Noise	0.01
Replay buffer size	50,000
Mini-batch size	512

TABLE I: Hyper-parameters used in TD3 agents training

B. Evaluation setups

Once the different models are learned, we evaluate their performance by randomly sampling new testing scenarios where we fix the positions of the delivery zone, the alternative delivery zone where the attacker has to bring the target, and the initial position of the target. For each of these scenarios, we randomly draw a significant number of initial positions of the attacker in the playground. For each of these random initializations, we run the scenario using the three trained policies and to assert whether the attacker succeeds in hijacking the target. For each test scenarios, we plot a map of the successful initial attacker positions and unsuccessful initial positions and estimate a percentage of successful attempts (success rate). We run 1000 different scenarios with the three trained policies.

Depending on the ratio between the speed of the target and the maximum speed of the attacker, we may theoretically know if the interception is possible depending on the initial geometry. We define the disc $\mathcal{D}$ centers on the delivery zone with a radius of R that is computed taken into account the distance d_{TD} between the initial position of the target (I_t) and the delivery zone (D_1)and the ratio $\frac{v_a^{max}}{v_t}$ between the speeds of the attacker and the target.

$$R = \frac{v_a^{max}}{v_t} \, d_{I_t D_1} \tag{1}$$

When the attacker is in $\mathcal{D}$, the interception is theoretically achievable since the attacker has the possibility to reach the delivery zone before the target and can thus interact with it. For a given scenario where the delivery area, the alternative delivery area and the initial position of the target are fixed, $\mathcal{D}$ is also fixed for all episodes.

To evaluate the performance of the hijacking policy, we can empirically estimate the reconstructed probability density function of the success rate on all test scenario. We also define following metrics:

M1: Number of successes over the number of episodes for initial target position in playground.

M2: Number of successes over the number of episodes for initial target position in $\mathcal{D}$.

M3: Number of successes over the number of episodes for initial target position not in $\mathcal{D}$.

$\overline{n_{step}}$: Mean of the number of steps when the episode is a success

$\overline{r_{dist}}$: Mean of the distance covered by the target over the initial distance from the target to the alternate delivery area when the episode is a success.

M1, M2 and M3 are success rates depending on the initial position of the target. $\overline{n_{step}}$, and $\overline{r_{dist}}$ are complementary metrics that evaluate the difficulty for the attacker to hijack the target.

C. Results

1) ***Step 1*** *Exploration of scenarios:* Firstly, we generate 1000 random scenarios by fixing I_t and D_2 as described in the previous section. In order to ease analysis, D_1 is always in the center of the playground. This choice does not change initial geometry exploration, although it may mask some corner cases. For each scenario, we run for the three trained agent $\mathcal{A}_{300}$, $\mathcal{A}_{600}$ and, $\mathcal{A}_{1000}$ 1000 episodes with random position for I_t. As expected during training, the agent $\mathcal{A}_{300}$ was not able to perform interceptions. Thus, we only focus on $\mathcal{A}_{600}$ and $\mathcal{A}_{1000}$ in the rest of this section.

We present on figure 4 the distribution of the success rate for the 1000 scenarios for agent $\mathcal{A}_{600}$ and $\mathcal{A}_{1000}$ with histograms. We see that the maximum speed of the attacker has a huge impact on the capacity of the attacker to intercept the target. This is confirmed by an average success rate of 39.2% for $\mathcal{A}_{600}$ and 91.4% for $\mathcal{A}_{1000}$.

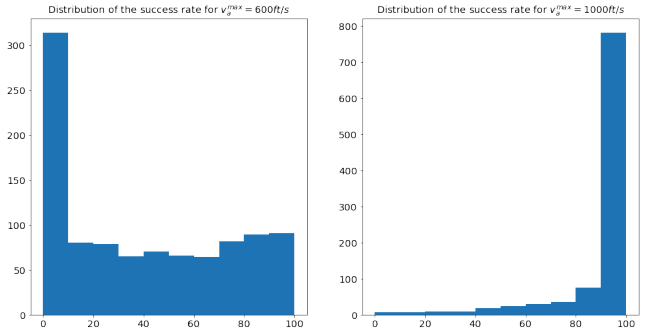


Fig. 4: Distribution of the scenario's success rate for $\mathcal{A}_{600}$ and $\mathcal{A}_{1000}$

Then, in figure 5, we present D_2 and I_t for the scenarios that have a good success rate ($> 90\%$) and a bad success rate ($< 10\%$) for trained agents $\mathcal{A}_{600}$ and $\mathcal{A}_{1000}$.

These figures highlight the impact of the initial geometry (relative position of the different elements) on the success rate. We recall that the D_1 area is in the center of the playground for all scenarios. When the ratio $\frac{v_a^{max}}{v_t}$ is high (top 2 of Figure 5), the distance $\overline{I_t D_1}$ has a huge impact on the success rate of a scenario. The closer the target is to the first delivery area D_1, the more the interception is difficult. Interpretation is not that straightforward when the ratio is low (bottom 2 of Figure 5). It seems that the learned policy is more efficient when the alternate delivery area D_2 is between or almost between the initial position of the target I_t and the initial delivery area D_1.

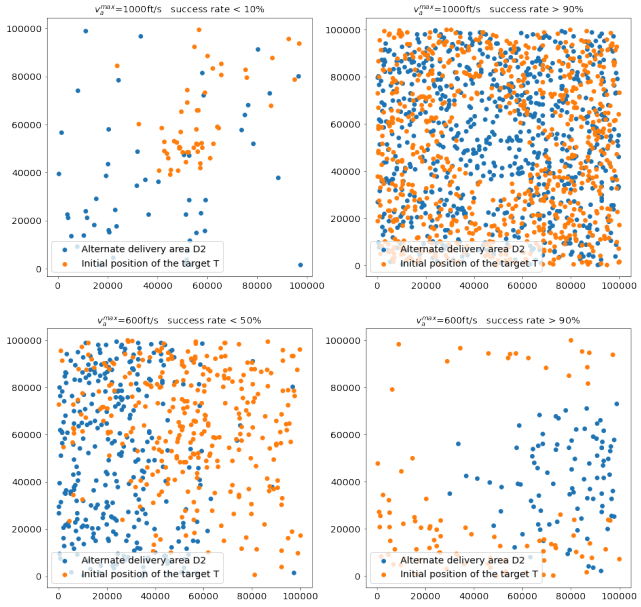


Fig. 5: Alternate delivery areas and initial positions of the target

2) ***Step 2*** *Exploration of selected scenarios :* this section focuses on some selected scenarios to provide a deeper look into how behave the different policies. In Figure 6, we plot the success rate of $\mathcal{A}_{1000}$ vs the success rate of $\mathcal{A}_{600}$. From this graph, we select 3 scenarios $\mathcal{S}_1$, $\mathcal{S}_2$ and $\mathcal{S}_3$.

- $\mathcal{S}_1$ is a scenario where both trained agent have a bad success rate;
- $\mathcal{S}_2$ is a scenario with a good success rate for $\mathcal{A}_{1000}$ and a relatively poor for $\mathcal{A}_{600}$;
- and finally $\mathcal{S}_3$ where both agents have a good success rate.

Each scenario correspond to a specific value of D_1, D_2 and I_t.

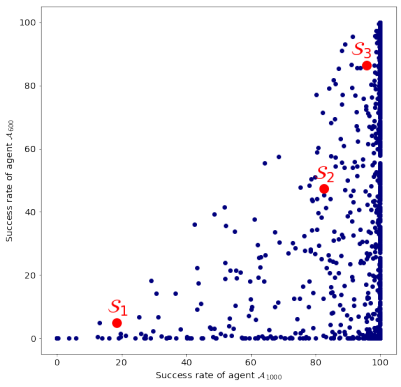


Fig. 6: Scenarios selected for exploration

For each couple $(\mathcal{S}_x, \mathcal{A}_y)_{x\in(1,2,3),y\in(1,2)}$, we launch 10000 episodes with random I_a positions. Then we compute in table

II metrics described in VI-B. The different graphs of Figure 7 present successes and failures for all initial positions of the attacker. We also plot D_1, D_2 and $\mathcal{R}$, the circle inside which the attacker should be able to hijack the target.

Scenario	Agent	m_1	m_2	m_3	$\overline{n_{step}}$	$\overline{r_{dist}}$
$\mathcal{S}_1$	$\mathcal{A}_{1000}$	.19	.56	.16	218	1.39
	$\mathcal{A}_{600}$	.06	.27	.05	346	2.21
$\mathcal{S}_2$	$\mathcal{A}_{1000}$	.84	.96	.73	197	1.47
	$\mathcal{A}_{600}$	.46	.79	.41	423	3.16
$\mathcal{S}_3$	$\mathcal{A}_{1000}$	.96	.96	NA	219	1.27
	$\mathcal{A}_{600}$	.86	.87	.77	370	2.15

TABLE II: Metrics for $(\mathcal{S}_x, \mathcal{A}_y)_{x\in(1,2,3),y\in(1,2)}$

For all scenarios, $\mathcal{A}_{1000}$ performs better than $\mathcal{A}_{600}$. This confirms the global success rate metric and highlights the necessity for the attacker to have a higher velocity than the target. The average ratio also shows that the task hijacking is more accessible when the maximum speed of the attacker is higher.

The success rate inside $\mathcal{R}$ (m_3) is consistently higher the than global one (m_1), (even if not always equals to 100% as we could expect).

From Figure 7, one characteristic which seems discriminant is the distance between the initial position of the target I_t and the delivery zone D_1: the smallest it is, the hardest it is to perform the hijacking. Even inside the $\mathcal{R}$, the success rate is poor, especially in $\mathcal{S}_1$. This can be explained by the fact that the circle has been defined considering the attacker goes directly to D_1. In practice, the initial heading the attacker is random so it has to change direction to head towards D_1. When there is not a lot of time, the change of heading is too long and the attacker cannot hijack. When time is longer, the effect is less visible, especially for $\mathcal{A}_{1000}$.

Failures inside $\mathcal{R}$ for $\mathcal{S}_3$ seem to form a pattern that would require additional investigations. It may show that our policy is not perfect or that our training scenarios did not explore enough certain configurations. This is the object of next step.

3) ***Step 3*** *Exploration of trajectories :* In this section, we present some representative observed trajectories to understand the red area of $(\mathcal{S}_3, v_{1000}^{max})$ and the red area that is inside $\mathcal{R}$ in $(\mathcal{S}_2, v_{1000}^{max})$.

Figure 8 presents the trajectories for two close initial positions of the attacker for scenario $(\mathcal{S}_3, v_{1000}^{max})$. This corresponds to the red area upper right of D_1 on figure 7. With the first position, the interception is not achieved. We can observe that the attacker has a trajectory that intercepts the target after D_1. On the contrary, for the second episode, the attacker intercepts the target before D_1. Failures of the $(\mathcal{S}_3, v_{1000}^{max})$ scenario have similar behaviors. The attacker tries to intercept the target after the D_1 area. This is mainly due to the reward function used during the training of the attacker agent. We used an heuristic that rewards the attacker when the target reduces the distance with D_2 in order to ease the learning of the policy. In order to push the attacker to hijack before the target reaches D_1, we would need to modify the reward to alleviate this undesired behavior. The proportion of episodes leading to such cases is low but definitely the reward function should be updated to avoid such phenomena.

Figure 9 shows the trajectories for two close initial positions of the attacker for the scenario $(\mathcal{S}_2, v_{1000}^{max})$. This corresponds to the red area inside $\mathcal{R}$. In the first episode, the initial velocity of the attacker is small and its acceleration is not strong enough to be able to interact with the target before it reaches D_1. When the initial velocity of the attacker is higher, in the second episode, the attacker is able to interact with the target almost from the beginning, and it is able to hijack it. These situations can occur when the attacker is opposite to the target compared to D_1. It could be solved by increasing the minimal initial speed of the attacker, fixing the initial heading of the attacker towards D_1 or increasing the attacker maximum acceleration.

VII. Conclusion

This work highlights the possibility of using reinforcement learning to train a malicious agent to hijack a delivery drone equipped with the ACAS-Xu avoidance system. Up to our knowledge, we are the first to demonstrate the possibility of fuzzing the algorithm of an autonomous avoidance system. With the current ambition in the aerospace industry for the development of autonomous systems (specifically air taxis and autonomous delivery drones), we are highlighting a security breach. The acceptance of these autonomous vehicles requires the resolution of these security issues.

As the ACAS-Xa system uses the same principle of tables, we believe that our method also applies.

Although we have made simplifying assumptions within the simulation environment, we believe that RL is efficient for attacking the ACAS-Xu system. An extension could be focused on adding complexity to the environment by adding static and dynamic obstacles and considering the vertical dimension. This should imply an extension of the action state of the attacker (new dimension) and an increase of the complexity of the reward function to handle obstacles.

In a future work, we consider training both agents with reinforcement learning in a zero-sum two-player games (as in [41], [42]) to produce collision avoidance system policies robust to malicious attacks.

We believe that, in order to improve the security, a reference use case should be provided with the ACAS-Xu. Access to Look Up Tables is unfortunately not possible for organizations that are not part of the RTCA/EUROCAE, the authority responsible for the normalization of the ACAS-Xu. Nevertheless, thanks to [20], neural networks approximating ACAS-Xu look-up tables are available. Therefore, we would like to replace the ACAS-Xu look-up table with these neural networks inside the gym environment and free it through our git repository (will be available for the conference). This would enable Reinforcement Learning and Security communities to work on this topic with the final objective of improving security in aeronautics.

Acknowledgement

This project received funding from the French "Investing for the Future – PIA3" program within the Artificial and Natural

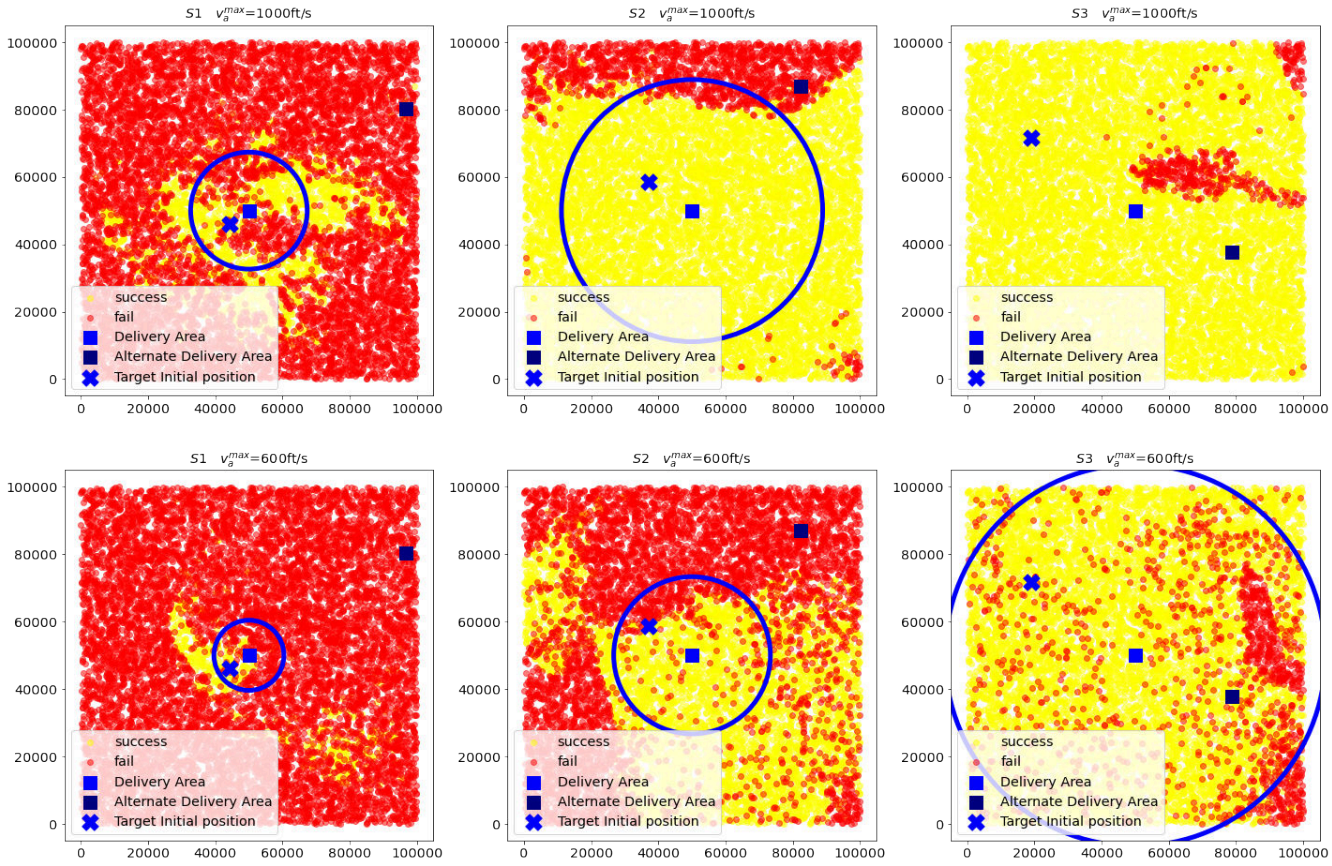


Fig. 7: Success/Failures for different initial position of the attacker among selected scenarios

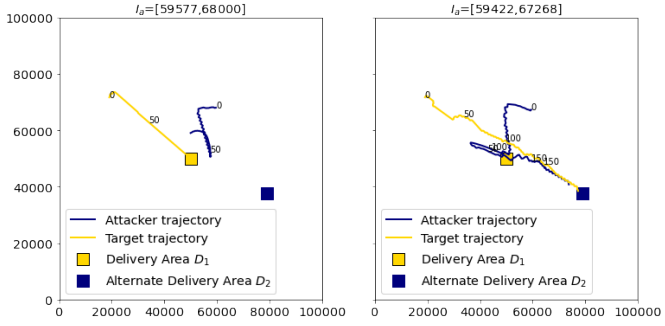


Fig. 8: Analyse of trajectories in S3 scenario for v_{1000}^{max}

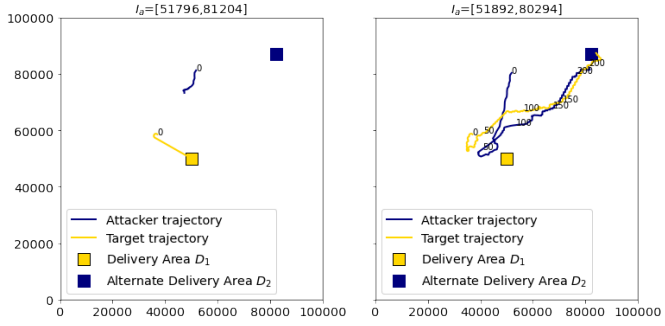


Fig. 9: Analyse of trajectories in S2 scenario for v_{1000}^{max}

Intelligence Toulouse Institute (ANITI). The authors gratefully acknowledge the support of the DEEL project[1].

REFERENCES

[1] R. N. Akram, K. Markantonakis, R. Holloway, S. Kariyawasam, S. Ayub, A. Seeam, and R. Atkinson. Challenges of security and trust in avionics wireless networks. In 2015 IEEE/AIAA 34th Digital Avionics Systems Conference (DASC), pages 4B1–1–4B1–12, 2015.

[2] B.-C. A. S. E. Ariyurek S. Automated video game testing using synthetic and human-like agents. arxiv, 1906.00317v1, 2019.

[3] P. Becker-Ehmck, M. Karl, J. Peters, and P. van der Smagt. Learning to fly via deep model-based reinforcement learning. arXiv preprint arXiv:2003.08876, 2020.

[4] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym. arXiv preprint arXiv:1606.01540, 2016.

[5] E. Çetin, C. Barrado, and E. Pastor. Counter a drone in a complex neighborhood area by deep reinforcement learning. Sensors, 20(8):2320, 2020.

[6] Y. Cheng and Y. Song. Autonomous decision-making generation of

[1] https://www.deel.ai/

uav based on soft actor-critic algorithm. In 2020 39th Chinese Control Conference (CCC), pages 7350–7355. IEEE, 2020.

[7] A. Clavière, E. Asselin, C. Garion, and C. Pagetti. Safety Verification of Neural Network Controlled Systems. In 7th International Workshop on Safety and Security of Intelligent Vehicles (SSIV 2021), 2021.

[8] M. Damour, F. D. Grancey, C. Gabreau, A. Gauffriau, J.-B. Ginestet, A. Hervieu, T. Huraux, C. Pagetti, L. Ponsolle, and A. Clavière. Towards certification of a reduced footprint acas-xu system: A hybrid ml-based solution. In International Conference on Computer Safety, Reliability, and Security, pages 34–48. Springer, 2021.

[9] R. C. Dorf and R. H. Bishop. Modern control systems. Pearson Prentice Hall, 2008.

[10] EUROCAE WG 75.1 /RTCA SC-147. Minimum Operational Performance Standards For Airborne Collision Avoidance System Xu (ACAS Xu), 2020.

[11] S. Fujimoto, H. Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. In International Conference on Machine Learning, pages 1587–1596. PMLR, 2018.

[12] M. Gnanasekera, A. V. Savkin, and J. Katupitiya. Range measurements based uav navigation for intercepting ground targets. In 2020 6th International Conference on Control, Automation and Robotics (ICCAR), pages 468–472. IEEE, 2020.

[13] Y. J. Guido Manfredi. An introduction to acas xu and the challenges ahead. In 35th Digital Avionics Systems Conference, Sep 2016, Sacramento, United States. IEEE/AIAA, 2016.

[14] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In International conference on machine learning, pages 1861–1870. PMLR, 2018.

[15] J. Hwangbo, I. Sa, R. Siegwart, and M. Hutter. Control of a quadrotor with reinforcement learning. IEEE Robotics and Automation Letters, 2(4):2096–2103, 2017.

[16] W. B. III. Airbus concludes attol project that featured world-first automated takeoffs and landings. Aviation Today, 2019.

[17] J.-B. Jeannin, K. Ghorbal, Y. Kouskoulas, R. Gardner, A. Schmidt, E. Zawadzki, and A. Platzer. Formal verification of acas x, an industrial airborne collision avoidance system. In 2015 International Conference on Embedded Software (EMSOFT), pages 127–136, 2015.

[18] K. T. L. G. Ì. Joakim Bergdahl, Camilo Gordillo. Augmenting automated game testing with deep reinforcement learning. arxiv, 2103.15819v1, 2021.

[19] K. D. Julian, J. Lopezy, J. S. Brushy, M. P. Owenz, and M. J. Kochenderfer. Deep neural network compression for aircraft collision avoidance systems. 35th Digital Avionics Systems Conference (DASC), 2016.

[20] G. Katz, C. W. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. CoRR, abs/1702.01135, 2017.

[21] W. Koch, R. Mancuso, R. West, and A. Bestavros. Reinforcement learning for uav attitude control. ACM Transactions on Cyber-Physical Systems, 3(2):1–21, 2019.

[22] M. J. Kochenderfer, J. E. Holland, and J. P. Chryssanthacopoulos. Next-generation airborne collision avoidance system. Technical report, Massachusetts Institute of Technology-Lincoln Laboratory Lexington United States, 2012.

[23] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. In 4th International Conference for Learning Representations, 2016.

[24] R. Lowe, Y. WU, A. Tamar, J. Harb, O. Pieter Abbeel, and I. Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. Advances in Neural Information Processing Systems, 30:6379–6390, 2017.

[25] G. Manfredi and Y. Jestin. An introduction to acas xu and the challenges ahead. In 35th Digital Avionics Systems Conference (DASC'16), pages 1–9, 2016.

[26] M. Min, L. Xiao, D. Xu, L. Huang, and M. Peng. Learning-based defense against malicious unmanned aerial vehicles. In 2018 IEEE 87th Vehicular Technology Conference (VTC Spring), pages 1–5. IEEE, 2018.

[27] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In International conference on machine learning, pages 1928–1937. PMLR, 2016.

[28] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602, 2013.

[29] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. nature, 518(7540):529–533, 2015.

[30] F. Netjasov, A. Vidosavljevic, V. Tosic, M. H. Everdij, and H. A. Blom. Development, validation and application of stochastically and dynamically coloured petri net model of acas operations for safety assessment purposes. Transportation Research part C: emerging technologies, 33:167–195, 2013.

[31] U. D. of transportation Federal Aviation Administration. Introduction to tcas ii version 7. https://www.faa.gov/documentlibrary/media/advisory_circular/tcas%20ii%20v7.1%20intro%20booklet.pdf, 2000.

[32] H. X. Pham, H. M. La, D. Feil-Seifer, and L. V. Nguyen. Autonomous uav navigation using reinforcement learning. arXiv preprint arXiv:1801.05086, 2018.

[33] A. Raffin, A. Hill, M. Ernestus, A. Gleave, A. Kanervisto, and N. Dormann. Stable baselines3. https://github.com/DLR-RM/stable-baselines3, 2019.

[34] R. santamarta. Arm ida and cross check: Reversing the 787's core network. https://act-on.ioactive.com/acton/attachment/34793/f-cd239504-44e6-42ab-85ce-91087de817d9/1/-/-/-/-/Arm-IDA%20and%20Cross%20Check%3A%20Reversing%20the%20787%27s%20Core%20Network.pdf, 2019.

[35] H. Sathaye, D. Schepers, A. Ranganathan, and G. Noubir. Wireless attacks on aircraft instrument landing systems. In 28th {USENIX} Security Symposium ({USENIX} Security 19), pages 357–372, 2019.

[36] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. Nature, 588(7839):604–609, 2020.

[37] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In International conference on machine learning, pages 1889–1897. PMLR, 2015.

[38] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347, 2017.

[39] P. A. Series. Aircraft hacking. 2013.

[40] R. L. Shaw. Fighter combat. Tactics and Maneuvering; Naval Institute Press: Annapolis, MD, USA, 1985.

[41] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. nature, 529(7587):484–489, 2016.

[42] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. Science, 362(6419):1140–1144, 2018.

[43] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller. Deterministic policy gradient algorithms. In International conference on machine learning, pages 387–395. PMLR, 2014.

[44] R. S. Sutton, D. A. McAllester, S. P. Singh, Y. Mansour, et al. Policy gradient methods for reinforcement learning with function approximation. In NIPs, volume 99, pages 1057–1063. Citeseer, 1999.

[45] G. Tong, N. Jiang, L. Biyue, Z. Xi, W. Ya, and D. Wenbo. Uav navigation in high dynamic environments: A deep reinforcement learning approach. Chinese Journal of Aeronautics, 34(2):479–489, 2021.

[46] H. Van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. In Proceedings of the AAAI conference on artificial intelligence, volume 30, 2016.

[47] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. Nature, 575(7782):350–354, 2019.

[48] B. Vlahov, E. Squires, L. Strickland, and C. Pippin. On developing a uav pursuit-evasion policy using reinforcement learning. In 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA), pages 859–864. IEEE, 2018.

[49] S. L. Waslander, G. M. Hoffmann, J. S. Jang, and C. J. Tomlin. Multi-agent quadrotor testbed control design: Integral sliding mode vs. reinforcement learning. In 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 3712–3717. IEEE, 2005.

[50] C. J. Watkins and P. Dayan. Q-learning. Machine learning, 8(3-4):279–292, 1992.

[51] S. Xuan and L. Ke. Uav swarm attack-defense confrontation based on multi-agent reinforcement learning. In Advances in Guidance, Navigation and Control, pages 5599–5608. Springer, 2022.

[52] K. D. Young, V. I. Utkin, and U. Ozguner. A control engineer's guide to sliding mode control. IEEE transactions on control systems technology, 7(3):328–342, 1999.

Practical Trust×Performance Metrics for Block Cipher Evaluation in Automotive Environments

Jacob Samuel, Keerthi K., Chester Rebeiro, Werner Schreiber, Geopeter Moothedan, and Ralph Mader

***Abstract*—With several interconnected Electronic Control Units (ECUs), modern automobiles are networks on wheels and an easy target for cyber-attacks. To achieve confidentiality, integrity and authenticity of network communication, security measures based on block ciphers can be employed. These block ciphers should have sufficiently low latencies to meet the strict requirements of the automotive environment. Most block ciphers developed today are extensively scrutinized and provide strong security guarantees. At a high-level, the security offered by these ciphers are at a par. It is thus a challenge for a designer to pick the right cipher for securing communication in the automotive network.**

This paper addresses this challenge by *(a)* providing a methodology to quantify the trust in a cipher and *(b)* studying the execution latencies on popular automotive platforms. The trust metrics are based on assessing the potency of the known attacks on a cipher, such as the threat of attacks in real-world environments. Second, it also provides a measure that quantifies world-wide scrutiny and adoption of a cipher based on heuristics such as the number of citations for the cipher. Together, the trust metrics and the execution latencies provide the designers with a systematic approach to choose block ciphers for automotive environments.

***Index Terms*—cyber-security, automotive, encryption, trust, latencies**

I. Introduction

In today's automotive environments, mechanical actions are governed by actuators which are controlled by a network of Electronic Control Units (ECUs). This network provides avenues for hackers to modify or inject malicious packets which could greatly affect the safety and security of the car. Several recent car hacking incidents, most notably of the Jeep by Charlie Miller and Chris Valasek [1], have emphasised the need to incorporate confidentiality, integrity, and authenticity of communication between ECUs.

A considerable challenge that hampers incorporating security features in an automobile is the latency overheads. Automotive systems are extremely time-critical [2]. Naïvely incorporated security features could potentially harm overall performance, increase communication latencies, and hamper the real time behavior requirements of the automobile [3]. A discussion on CAN bus security and its corresponding performance impacts can be found in [4]. Thus, automotive software developers have to pick cryptographic primitives that not just provide confidentiality, integrity and authenticity of communication, but at the same time ensure minimum overheads considering the capabilities and the performance of the microcontroller or microprocessor in use.

While popular microcontrollers like Infineon Aurix TC3xx [5] contain hardware accelerators for cryptographic primitives, medium and low performance microcontrollers (8/16 bit), have no or limited hardware accelerators for cryptography are still common in the automotive domain, for example in sensor/actuator communication. While in the latter case, software implementations of lightweight cryptography is essential, in the former, software implementations may employed to provide more flexibility in the protocols.

In the last few years, with global contests such as the NIST block cipher challenge[1] and the lightweight cryptography challenge[2], there are several lightweight block cipher algorithms that are potential candidates. The security of these new cipher algorithms are found to be mostly at a par. Most cryptanalytic attacks discovered are theoretical and cannot be mounted in practice. This is because cipher designers, after several decades of extensive research, now understand the science of designing strong ciphers. Thus, every new cipher is designed to resist most cryptanalytic attacks including linear, differential, related-key, and algebraic cryptanalysis. Further, the global nature of the contests, encourages researchers world-wide to thoroughly scrutinize new cipher designs to identify weaknesses.

Every cipher is designed with a different structure and operations. While most designs follow the extensively researched, Feistel [6] and Substitution Permutation [7] Networks, others experiment with new structures such as the Add-Rotate-Exor (ARX) [8]. The differences in the structure and operations results in implementations that have different execution time on different platforms. Thus, it is a challenge for an automotive software designer to pick amongst these: a cipher that it *trusts* would provide secure communication at minimum overheads on the target platform.

In this paper we **(a)** provide a methodology to quantify the trust of a block cipher algorithm. The trust metrics are based on the potency of the attacks published and the world-wide perception of the cipher. A more potent attack reduces trust while a more globally accepted cipher, increases its trust. **(b)** We use this methodology to evaluate and rank 9 block cipher algorithms from most trusted to least trusted. **(c)** We then evaluate the overheads of these ciphers on popular Automotive platforms Infineon Aurix 399 Renesas R-Car M3, and the TI

J. Samuel, Keerthi K. and C. Rebeiro are with the Department of Computer Science and Engineering, IIT Madras, e-mails: jacobsam29@gmail.com, cs17d013@cse.iitm.ac.in, chester@cse.iitm.ac.in.

W. Schreiber, G. Moothedan, and R. Mader are with Vitesco Technologies, e-mails: {werner.schreiber, geopeter.moothedan, ralph.mader}@vitesco.com

[1]https://csrc.nist.gov/projects/cryptographic-standards-and-guidelines/archived-crypto-projects/aes-development

[2]https://csrc.nist.gov/projects/lightweight-cryptography

MSP 430. While the Infineon Aurix processor has six "Tri-Core" cores. The Renesas R-Car M3 is comparatively more powerful with multiple ARM A5x cores, typically deployed. TI's MSP 430 processor on the other has a small 16-bit core and is deployed in sensor environments. The evaluation provided in this work thus identifies the most suitable cipher candidates for a wide range of automotive environments.

This paper is organized as follows. Sections II and III provides a brief background on block ciphers and the related work in evaluating security and performance overheads respectively. Section IV provides the rationale for the proposed trust metric and evaluates 9 block ciphers. Section V discusses the performance evaluation of the ciphers on the Infineon Aurix 399, Renesas R-Car M3, and the TI MSP 430. Section VI provides caveats on the methodologies developed. The conclusion (Section VII) provides insights on choosing ciphers.

II. Background

Block ciphers work on fixed-length blocks of plaintext and a secret key to generate ciphertexts that provide confidentiality. They can be used in primitives such as Cipher-based Message Authentication Codes (CMAC) to verify the integrity and authenticity of messages. A popular block cipher like the AES works with 128-bit plaintext blocks (block size) and keys which are either 128, 192, or 256, to obtain a ciphertext block. All block cipher algorithms designed today follow Shannon's theory of an iterative multiplicative cipher. Each block cipher has multiple rounds organized either as a Feistel structure or a Substitution Permutation Network (SPN), as seen in Figure 1. The first round takes the plaintext as input, the second round input is the first round's output, and so on. The final round output is the ciphertext. Each round has a fixed set of operations, which perform *confusion* and *diffusion* and a round key addition. While the round key adds randomness to the ciphertext, the confusion breaks the relationship between the round's output with its input and the diffusion ensures that a change in the input spreads to multiple bits in the output.

III. Related Work

There is an urgent need for light-weight block ciphers in applications related to the Internet of Things (IoT) and Cyber Physical Systems (CPS). While global contests like NIST's Light Weight Cryptography challenge[3] or DSCI's Light Weight Cipher Contest[4] encourage the development and evaluation of new ciphers, there is little work done to compare trade-offs of security$\times$ performance amongst them for application in specific environments. An effort to provide a comparative analysis is presented in [9] which evaluated a set of light-weight ciphers for security and overheads in Wireless Sensor Networks. The performance evaluation is done on the STM32F407[5] micro-controller, while the avalanche effect is considered for security evaluation. This study is limiting for

[3]https://csrc.nist.gov/Projects/lightweight-cryptography

[4]https://www.dsci.in/ncoe-light-weight-cipher-design-challenge-2020/

[5]https://www.st.com/en/microcontrollers-microprocessors/stm32f407-417.html

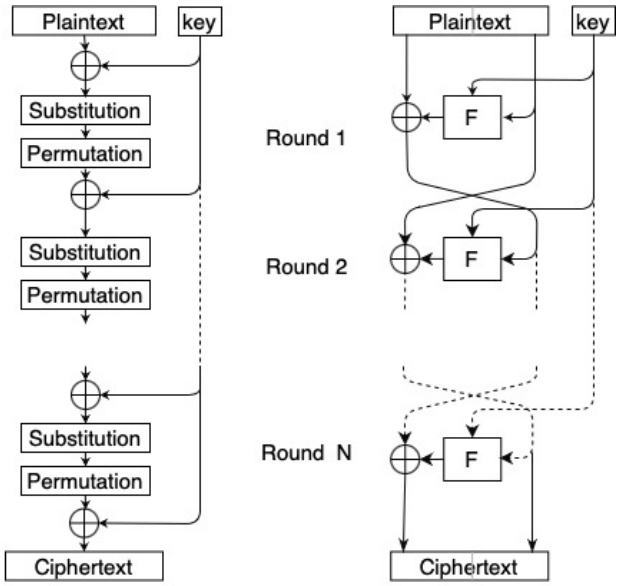


Fig. 1: Based on the structure, block ciphers are classified as either (*a*) SPN or (*b*) Feistel. While the AES adopts the SPN structure, many ciphers adopt the Feistel structure because of the rich body of research that has been done on Feistel networks.

two reasons. First, the performance evaluation on a single platform may not be sufficient as IoT and CPS environments can have a variety of different microprocessors. The throughput of a cipher execution on one platform may not match that on another. Second, the strength of a cipher is often dictated by its weakest link. The weakest link is not be easily identified. Thus evaluating a few cipher properties is insufficient to gauge the cipher's security.

In this paper, we evaluate ciphers for automotive environments considering both trust and performance. We measure trust of a cipher by carefully analyzing its structure, existing attacks, and scale of adoption/evaluation of the cipher. The latter is estimated by studying the citations of the cipher over the years. For performance evaluation, we cover the full range of automotive platforms from high end to low end. We evaluate the ciphers on the high-end Renesas R-Car M3 platform; the mid-range Infineon Aurix 39x; and the low end TI MSP430 micro-controller. In Section V we provides insights on the performance observed in these platforms based on the cipher's structure as well as the micro-architecture of these processors.

IV. Establishing trust metrics for block ciphers

During design, cipher algorithms are subjected to extensive security evaluation. Typically this is done by testing standard properties that are expected of a block cipher, such as, the indistinguishability of the ciphertext from random data, the cipher's ability to create ciphertexts that are not correlated to the corresponding plaintexts, and the ability to make two ciphertexts look very different even though the corresponding plaintexts differ in just a few bits. In addition to these tests,

every block cipher is critically analyzed for their ability to withstand a known set of cryptanalytic attacks. These attacks assume that the adversary possesses full knowledge of encryption algorithm except for the secret key and the aim of the attacker is to guess the right key. Due to strong assumptions about the attacker, it may seem apparent that all new ciphers with the same key size, provides the same levels of security.

One aspect that has changed with the wide-spread adoption of the Internet, is the public scrutiny of ciphers. Researchers extensively develop and publish cryptanalytic attacks on ciphers. Similarly, optimized implementations of ciphers on multiple platforms are published. Engineers extensively publish applications that use specific cipher algorithms. In this paper, we utilize the public scrutiny of the ciphers to establish trust metrics that are based on the potency of the published cryptanalytic attacks and global adoption of the cipher.

A. Gauging the potency of a cryptanalytic attack

A variety of cryptanalytic attacks have been developed over the years. Some, like the linear and differential attacks on DES could mark the end of a cipher. Other attacks, like the recent Biclique attacks on AES [10], are developed more from an academic perspective and would be almost impractical to launch in practice. In this section we introduce several metrics that are used to gauge the potency of an attack.

Adversarial Capabilities. Based on the attacker's capabilities, five attack models are popularly used. These models, ranked from most powerful to least powerful, are *(1)* Ciphertext only Attack (COA), *(2)* Known Plaintext Attack (KPM), *(3)* Chosen Plaintext Attack (CPA), *(4)* Chosen Ciphertext Attack (CCA), and *(5)* Related Key Attack (RKA). The threat of a particular attack model depends on the application scenario. The COA, which only requires ciphertexts is often considered the most powerful because the attacker knows very little about the encryption. However, in applications such as network communication, a device can spoof packets forcing a victim to behave in certain predictable manner, potentially leading to KPA, CPA, and CCA attacks. Known Plaintext Attacks (KPA) are more powerful compared to CPA and CCA attacks. This is because, the KPA attacker is a passive listener of the messages that are encrypted. The Chosen Plaintext (CPA) and Chosen Ciphertext (CCA) attacks are equally potent requiring an active attacker. In CPA attacks, the attacker coerces the encryption of specific messages, while in CCA attacks, the attacker chooses ciphertext that are then decrypted. Related Key Attacks are more theoretical, and the weakest form. It requires that the attacker study the cipher's behavior under different keys. Due to the huge key space, this is not easily achieved. Since we require a trust metric that is independent of the application, we consider the first four attack models (COA, KPM, CPA, and CCA) to be equally potent. Related Key Attacks (RKA), however, are less practical due strong assumptions made, therefore less potent than the other models.

Online and Offline Attack Complexity. An attack has two phases: online and offline. In the online phase, the attacker would require to collect information about the encryption as per the attack model. In a COA for instance, the attacker would need to collect sufficient number of ciphertexts. While, in a CPA, the attacker would need to choose sufficient number of plaintexts and collect the corresponding ciphertexts. After the online phase, the attacker initiates the offline phase involving huge amounts of number crunching functions on GPUs, specialized hardware, or clusters of computers, before the secret key can be retrieved.

The complexity of the online phase is called *data complexity*, while the complexity of the offline phase is called *time complexity* of the attack. These complexities influence the potency of an attack. Higher complexities imply a weaker attack. We consider the data complexity to be more critical compared to the time complexity. This is because typical communication protocols require periodic key changes. An attack should therefore occur before a key is changed. Furthermore, limits of today's compute technologies restrict practical time complexities to less than 2^{80}. This complexity implies that 2^{80} keys need to be searched in order to identify the correct key. In other words, one out of the 2^{80} possible keys is correct. To provide an intuition about the difficulty of performing such a search, we consider optimistically that one CPU costing USD 500, can perform 3 billion key searches per second. With this CPU, it would take 6 million years (on average) to find the correct key. If the key search is parallelized, it would cost over 3 Billion to have sufficient number of CPUs to find the correct key within one year. Thus any attack with a data complexity greater than 2^{80} can be deemed impractical.

Reduced Round Attacks. Very often, developing a full break on the cipher is extremely difficult. Cryptographers then start to develop attacks on simplified models. One popular simplification is to develop attacks on reduced number of rounds of the cipher. While the reduced round attacks are not complete breaks, they still can be used to gauge the potency of the attack. For example, the 7 round attacks on AES is more potent than the 5 and 6 round attacks, but less potent than the 8 round attacks.

The cipher structure plays an important role in gauging the power of a reduced-round attack. For example, since Feistel ciphers typically have about twice the number of rounds as an SPN cipher, attacks on Feistel ciphers are typically deeper than SPN ciphers. While comparing attacks on reduced rounds, this should be considered. For example, a 7 round attack on the SPN structure of AES is equivalent to a 12 or 13 round attack on the Feistel cipher, CLEFIA.

To gauge the potency of a reduced round attack, we introduce a metric called *Penetration Factor*, defined as

$$PenetrationFactor = \frac{SecurityMargin}{NumberofRounds} \ ,$$

where *SecurityMargin* is the number of rounds considered in the attack. A higher penetration factor indicates a stronger attack. If a cipher has an attack on 8 rounds (reduced rounds) and the total number of round in the cipher is 12, then the security margin is 8 and penetration factor is $8/12 = 0.66$. Similarly, a full attack has a penetration factor of 1. This is maximum value the penetration factor can have.

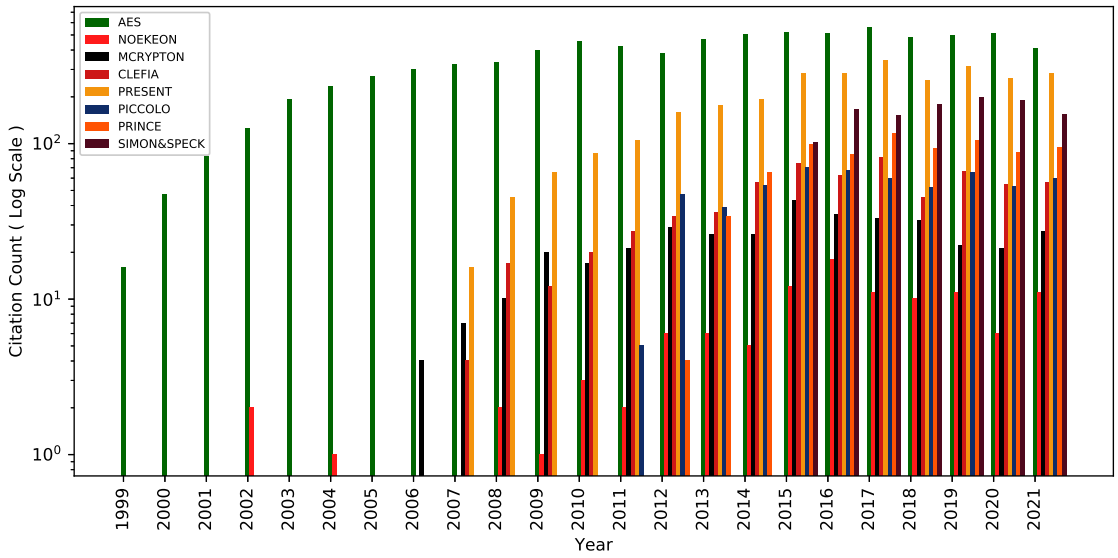


Fig. 2: Number of citations per year for different Block Ciphers. (Data obtained from Google Scholar (retrieved on 22nd December 2021). Citations taken from the following documents: AES [11], [7], NOEKEON [12], MCRYPTON [13], CLEFIA [14], PRESENT [15], PICCOLO [16], PRINCE [17], SIMON&SPECK [8]. As can be seen, after an initial growth phase, the number of citations per year for a cipher is almost a constant, indicating a linear growth.

B. Public Scrutiny of Ciphers

None of the ciphers in use today can guarantee security. However, most ciphers are published online and publicly scrutinized. Cryptographers world-wide would try their hand at breaking the ciphers. A classic example of this is in the standardization of the AES cipher, which was a global contest between 15 ciphers that were extensively studied and evaluated by cryptographers world wide, eventually resulting in Rijndael being was chosen as the standard. Even after the contest, which ended in October 2000, the AES has been extensively scrutinized by a large number of researchers for about two decades. Besides this, a large number of organizations have adopted AES for their encryption requirements. A cipher that is extensively scrutinized and adopted is more trustworthy than a cipher that is less scrutinized or adopted.

While it is difficult to find accurate numbers of how many cryptographers have tried to break a cipher or the number of organizations that have adopted the cipher for their applications, a reasonable indication can be obtained from the number of citations that the original ciphers have and the age of the cipher. The number of citations is an indicator of three aspects: **(1)** the adoption of the cipher in an application; **(2)** an attack or analysis on the cipher; and **(3)** an efficient implementation. While (2) corresponds to scrutinizing the cipher's security properties, (1) and (3) corresponds to users who trust that the cipher is secure. All three give an indication of the cipher's trustworthiness.

Figure 2 shows the number of citations per year for the ciphers considered. The citations were obtained from Google scholar (retrieved on 22nd December, 2021). From Figure 2, it is clear that each cipher's citations/year is distinctively different. For example, on average, the AES cipher has 350 citations per year, while a less known cipher like NOEKEON, which is about the same age as the AES has only 22 citations. Thus, it is evident that the AES has been either scrutinized, adopted, or implemented much more aggressively compared to NOEKEON, thus can be more trusted. Similarly, comparing another pair of ciphers, CLEFIA and PRESENT, that are approximately the same age, PRESENT with over 2860 citations is more scrutinized, adopted, or implemented compared to CLEFIA. Thus, PRESENT is more trusted than CLEFIA.

To compare the public interest and scrutiny of a cipher, we introduce a metric called *Scrutiny Factor*, defined as follows.

$$ScrutinyFactor = \frac{\#citations}{AgeofCipher} \ .$$

The age of the cipher is taken with respect to the current year. The metric assumes a linear growth in the number of citations per year for a cipher. While this is observable in Figure 2, it is not always the case. There may be exceptions, such as the adoption of the cipher as a standard and a growth of a new sector requiring specialized ciphers, such as light-weight or low-latency. Ignoring such deviations, the average citations per year for a cipher is a good indicator of the trust cryptographers and users have on a cipher. This being said, any complete break with time complexity less than 2^{80} dominates public scrutiny.

C. Evaluating Trust in Block Ciphers

We consider 9 cipher algorithms for evaluation. These ciphers were chosen based on their popularity (for example AES) or based on their use in standards (for example CLEFIA [14] and PRESENT [15] are light-weight standards). Others have been designed by reputed organizations (like

TABLE I: Cryptanalytic Attack Comparisons with key size 128 bits. The acronyms in the columns refer as follows. BS: Block Cipher, year: introduction year (first published), attacks: number of published attacks, citations: as per Google scholar (approximate), SF: Scrutiny factor, AC: Adversary capability, DC: Data complexity, TC: Time complexity PF: Penetration factor.

Cipher	Type	BS	Year	Attacks	Citations	SF	Attack	AC	DC	TC	PF
AES	SPN	128	2000	22	8000	380	[18]	CPA	2^{106}	2^{110}	0.7
							[10]	MiTM/CCA	2^{88}	$2^{126.1}$	1
PRESENT	SPN	128	2007	8	2259	161	[19]	CPA	2^{60}	2^{20}	0.73
SPECK	Feistel	128	2013	7	625	78	[20]	CPA	2^{113}	2^{113}	0.53
							[21]	CPA	$2^{63.6}$	$2^{89.4}$	0.52
SIMON	Feistel	128	2013	7	625	78	[22]	CPA	2^{127}	2^{127}	0.72
							[23]	KPA	$2^{127.6}$	2^{120}	0.72
PRINCE	SPN	64	2012	3	598	66	[24]	KPA	$2^{57.95}$	$2^{126.56}$	0.5
							[25]	KPA(MiTM)	1	2^{124}	0.66
							[26]	KPA	$2^{57.94}$	$2^{60.62}$	0.83
CLEFIA	Feistel	128	2007	7	533	38	[27]	CPA	2^{113}	$2^{116.7}$	0.6
							[28]	CPA	$2^{126.83}$	$2^{126.83}$	0.72
MCRYPTON	SPN	64	2005	4	322	20	[29]	MiTM	2^{57}	2^{115}	0.75
PICCOLO	Feistel	64	2011	2	63	6	[30]	MiTM/CCA	2^{24}	$2^{126.79}$	0.9
							[31]	RKA	$2^{117.77}$	2^{118}	0.67
NOEKEON	Feistel	128	2000	1	86	4	[12]	CPA	$2^{20.6}$	$2^{108.1}$	0.31

SIMON and SPECK [8], designed by the NSA). The remaining have been chosen from a world-wide light-weight cipher standardization contest (for example MCRYPTON [13], PICCOLO [16], and PRINCE [17]). Although most of the ciphers considered support multiple block sizes and key lengths, due to space constraints, we restrict our evaluation in this paper to versions with block size and key length 128 bits. The ciphers cover a variety of configurations, including Generalized Feistel networks and SPN structures. We surveyed all published cryptanalytic attacks and rated them according to the metrics described in Section IV-A.

Table I provides a subset of the published attacks on these ciphers ordered on the Scrutiny Factor. As can be seen, the AES cipher is the most scrutinized, with over 8000 citations and 22 attacks. Even though a full attack on AES is feasible in [10], the data complexity of 2^{88} and time complexity of $2^{126.1}$, makes the attack infeasible in practice. This makes AES the most trusted amongst the ciphers considered. PRESENT too has a high Scrutiny Factor, and the attacks are difficult to mount due to the high data complexity. Thus, PRESENT too can be trusted, but not as much as AES. PICCOLO and NOEKEON have the least Scrutiny Factor and therefore the least trusted, even though there are no full attacks on these ciphers. Comparing PRINCE, SIMON and SPECK, they have similar Scrutiny Factor, but, Prince has a powerful attack with a low data complexity. Thus, Prince is less trustworthy compared to Simon and Speck.

V. Performance Overheads of Cipher Implementations

In real-time automotive environments, evaluation of a cipher's performance overheads is as important as the trust estimations. When considering cipher execution, one must differentiate between two main aspects: the overhead concerning the CPU utilization and the additional runtime needed for encryption and decryption in a synchronous execution with respect to the required deadline. The latter is the more critical as we see in applications involving a distributed network, the cipher execution is part of critical event chains with start to end response time, in the worst case, lower than 2ms.

Figure 3 shows a typical event chain from a smart sensor via the control unit to the smart actuator via an automotive bus, like CAN. To ensure a deterministic behavior of the event chain, the tasks on the different devices are synchronized to a common time base and the calculation sequences must comply to the planned time window for each device. t_{01} to t_{04}, illustrate the timing overhead for decryption and encryption of the data to ensure a secure communication. In worst case examples, the timing windows are in the range of less then 500µs, where the phases for reading the input, computation and triggering the output without encryption could last up to 300µs. For the timing considerations for the control unit in the middle of the chain, one must consider the twice the time as the data has to be decrypted before the calculation and then encrypted again. Depending on the interrupt load on the calculation core and delays due to higher prior activities typically the net core execution time of the calculation sequence shall not exceed 70% of the time window. In the example this would mean 350µs implying that the decryption and encryption shall not take longer than 25µs. This example shall give the order of magnitude and will differ from case to case, depending on the problem to solve. Therefore, the target is to have to cipher with the least execution time while guaranteeing an acceptable level of trust. Energy consumption of the micro-controller was not evaluated as in power-train applications it is of less importance, compared to the energy consumption of the power stages used to drive the sensors and actuators.

To understand the tradeoffs between performance and trust, we evaluate the ciphers on three platforms: the Infineon Aurix TC399, Renesas R-Car M3, and TI MSP430. These are

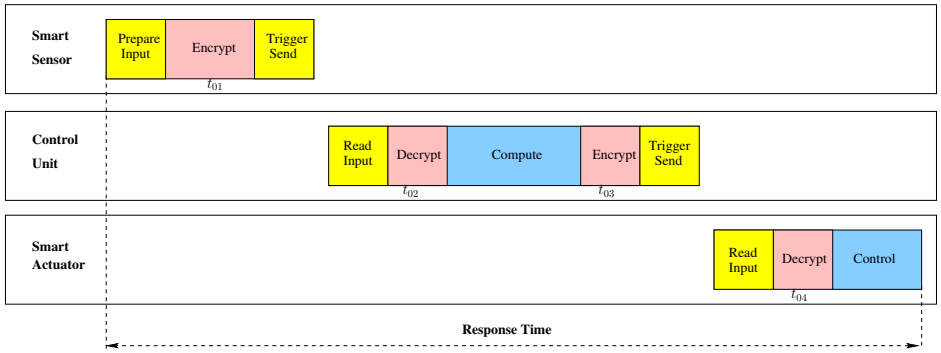


Fig. 3: Event chain for a control flow from a smart actuator via a control unit to a smart actuator.

TABLE II: Feature comparison of the three platforms used in the evaluation. The platforms cover a wide range of common processors. From the 16-bit Texas Instruments MSP430 processors, which are typically deployed in sensors, to the high-end platform Renesas R-Car M3 that run embedded Linux.

Feature	TI MSP430	Infineon Aurix TC399	Renesas R-Car M3	
			Cortex A53	Cortex A57
Size	16-bit	32-bit	32-bit	
Architecture	RISC (vonNeumann)	RISC (Tricore)	RISC (ARM v8.0)	
Pipeline	3-stage inorder	4-stage superscalar	8-stage inorder	15-stage OoO
Cache Memory	Not present	32KB L1-Instruction 16KB L1-Data	32K L1-Instruction 32K L1-Data, 512K L2	48K L1-Instruction 32K L1-Data, 2MB L2
Clock	16MHz	300MHz	1.2GHz	1.5GHz
Operating System	none	none	Embedded Linux	

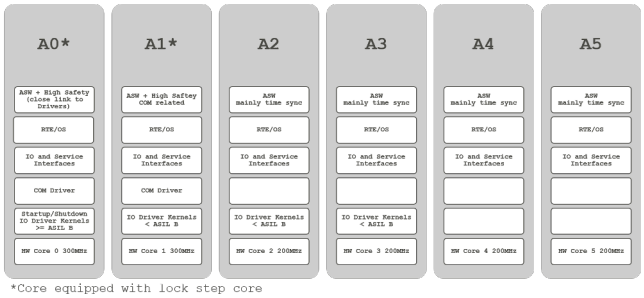


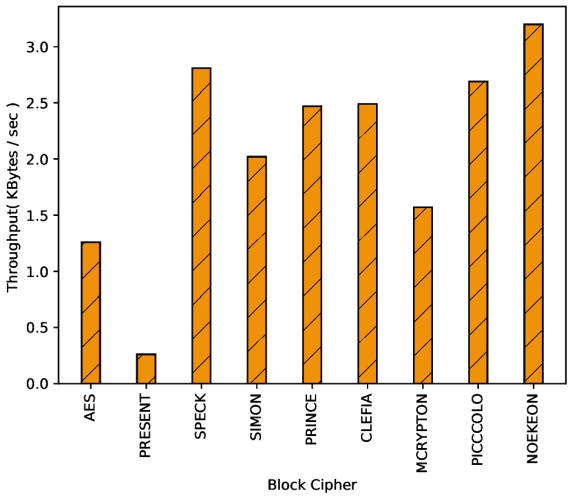
Fig. 4: High level core partitioning of an AURIX TC39 used in an automotive powertrain application.

popular microprocessors for automotive applications chosen to cover a range of platforms from high-performance to sensor networks. Table II compares important parameters in the three platforms. The Infineon Aurix TC399 microcontroller comprises of six symmetrical processing cores, each with a maximum clock frequency of 300Mhz, program flash memory of up to 16MB and over 6MB of integrated RAM.

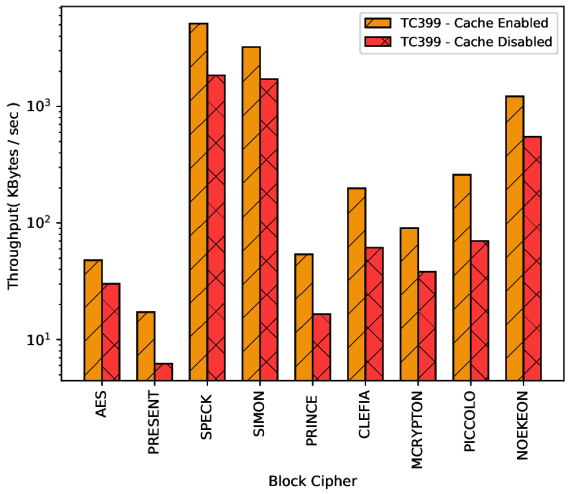
Figure 4 shows a typical partitioning of the software on an Infineon AURIX TC39x [32]. The basic software is distributed on multiple cores and each driver can be accessed from any core by the application software [33] [34]. The communication relevant software parts are located on those cores which are equipped with lock step cores and the communication stacks are distributed to meet the performance requirements in high loaded projects[35]. On these cores the encryption and decryption of security relevant is executed especially for the data which are part of a time critical event chain as shown in Fig. 3. Although the TC39x is equipped with hardware accelerators, practical experience showed that encryption, using the accelerator hardware, takes in total even more time compared to the software algorithm executed on a computational core. This is caused by the communication overhead when triggering the hardware accelerator and the resolving of accesses to this shared resource, if triggered from more than one core.

A large number of factor affect the execution time of block ciphers. Figure 5 shows that Noekeon, SPECK, and PRINCE block ciphers provides the highest throughput on the TI MSP 430, Infineon Aurix TC399, and the Renesas R-Car M3 respectively. We list the important factors that influence execution time and hence the throughput. In all measurements the ECB mode of encryption was used on randomly generated plaintext messages.

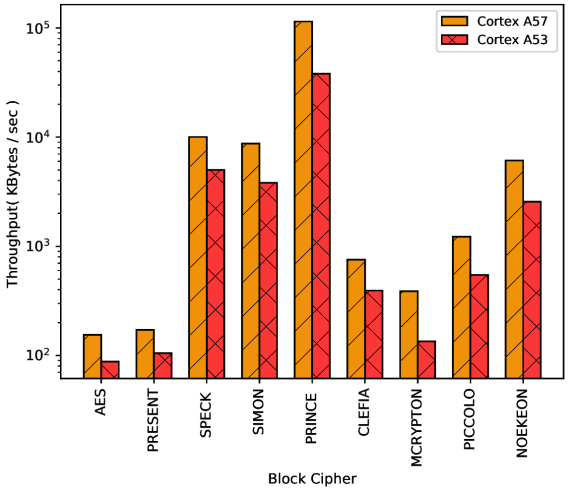
- The execution time of a cipher depends on the cipher algorithm as well as its implementation. A cryptographer may design an algorithm that abets efficient implementations, however, a badly coded realization of the cipher may negate the cryptographer's objectives. Alternatively, implementations may be optimized for memory footprint rather than performance. We have used open source software written in C for all the cipher implementations. Further optimisations may be possible by hand-crafted assembly.
- It is observed that Feistel Networks (SPECK, SIMON PICCOLO and CLEFIA) based block cipher algorithms are in general faster compared to SPN based block ciphers. This is because the operations in each round are fewer in Feistel networks as they work with half the



(a) TI MSP430 clocked at 16 MHz



(b) Infineon Aurix TC399 clocked at 300MHz



(c) Renesas R-Car M3 clocked at 1.2GHz and 1.5GHz

Fig. 5: Throughput for various cipher implementations on the three evaluation platforms.

input at a time. The instructions are thus more easily cached. For example, AES operates on 16 bytes of data per round. The operations modify and perform confusion and diffusion operations on all the 16 bytes. However, an equivalent cipher like CLEFIA, has a Feistel structure and operates on 8 bytes per round. These 8 bytes and the corresponding operations are more easily cached even on the small Aurix processor. Thus, even though, CLEFIA has 18 round (compared to 10 rounds of AES), it executes much faster.

- Some ciphers, like the popular light-weight cipher PRESENT, are well studied by the research community and expected to be small and fast. However, as we see from our results PRESENT is one of the slowest ciphers in the set. This is because ciphers like PRESENT are optimized for hardware and not for software. Their diffusion layer (permutation functions) in each round comprises of bit-operations. While bit-operations are very efficient in hardware, performing bit-operations in software is considerably expensive.
- Cipher implementations mainly comprise of load-store and integer ALU operations. Processors that can efficiently schedule these instructions perform well. For instance, the Infineon Aurix TC399 processor is superscalar with two pipelines – one dedicated for load-store operations while the other is used for ALU operations. This structure is ideal for cipher execution. Load-store operations can be executed in parallel with the ALU operations, leading to a performance much better than one would expect.
- Cipher implementations are extensively data intensive, thus most implementations suffer from the Von-Neumann bottleneck. TI MSP430 has a unified memory and no cache. Thus throughput of ciphers on the TI platform is several times lower compared to the other platforms. Data cache memory in processors, can help boost performance. However, extensively large cache memories are not very useful. This is because, data used by ciphers is localized,

for example within 512 bytes. A lightly-loaded system would therefore not benefit from a large cache.
- A significant contribution to the Von-Neumann bottleneck is due to the cipher's SBoxes. These provide the non-linearity in the results and typically implemented using lookup tables. Ciphers such as SPECK and SIMON, do not have SBoxes, but depend on other arithmetic operations to provide the necessary non-linearity. Thus, these ciphers are faster compared to the others that use lookup tables.
- Cipher algorithms are mostly sequential. Operations in a round cannot be performed until the results of the previous round is available [36]. Thus, not much benefits are gained from Out-of-order processors. This is evident from the Renesas R-car M3, where the out-of-order Cortex A-57 core does not have much benefits compared to the inorder and simplified Cortex A-53 CPU core in spite of the higher clock frequency, and a longer pipeline.
- It is not easy to generalize a cipher's performance across platforms. For instance, the PRINCE cipher which has the highest throughput on the Renesas R-Car M3 with respect to the ciphers, has one of the lowest performance in the Infineon Aurix TC399. This also motivates such multi-platform evaluation to understand performance of ciphers across processors.

VI. Caveats

- The use of citations to compute the Scrutiny Factor of a cipher assumes that a large proportion of the attacks and applications of a cipher are published. For instance, an unpublished full break on a cipher will not be considered in the trust metrics.
- The value of 2^{80} is fixed as the practical limits for the time complexity. This is a widely acceptable value given given today's computing capabilities. This limit would need to be revisited periodically with new computing technologies, for example quantum computing.
- The methodology to evaluate trust is based on heuristics and the state-of-art. Trust would need to be evaluated periodically to take into consideration new attacks and trends.

VII. Conclusion

There seems to be an inverse relationship between trust and execution overhead. The safest ciphers like AES and PRESENT have the worst performance on execution. There are two approaches to select a cipher. The first is to choose any of the trusted ciphers, viz-a-viz AES or PRESENT, and extensively optimize it for the given execution platform. This would involve hand written assembly code, hardware specific optimizations, and use crypto-accelerators present in the hardware. The second option is to choose a cipher that has a good trade off between performance and security. We feel SPECK meets that requirement. After AES and PRESENT, SPECK has the highest Scrutiny factor. The attacks too have a Penetration factor of 0.6 to 0.7, which is much better than many of the other ciphers. Execution overheads too are low.

References

[1] A. Greenberg, "Hackers remotely kill a jeep on the highway—with me in it," 2015. [Online]. Available: https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/

[2] V. Kalirajan, R. Mader, and S. Kastner, "Exploration of the real time behavior of event chains by simulation and measurement of "in-vehicle networks"," in *Symposium on International Automotive Technology*. SAE International, sep 2021. [Online]. Available: https://doi.org/10.4271/2021-26-0490

[3] D. Schwartzberg, "In-vehicle data latency: Fast or furious," 2019.

[4] O. Pfeiffer and C. Keydel, "Can security: how small can we go," 2019. [Online]. Available: https://can-newsletter.org/uploads/media/raw/48cedb145acf731e6d8481e787416d5c.pdf

[5] [Online]. Available: https://www.infineon.com/cms/de/product/microcontroller/32-bit-tricore-microcontroller/32-bit-tricore-aurix-tc3xx/

[6] K. Nyberg, "Generalized Feistel networks," in *International conference on the theory and application of cryptology and information security*. Springer, 1996, pp. 91–104.

[7] J. Daemen and V. Rijmen, *The design of Rijndael*. Springer, 2002, vol. 2.

[8] R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, and L. Wingers, "The simon and speck families of lightweight block ciphers," Cryptology ePrint Archive, Report 2013/404, 2013, https://eprint.iacr.org/2013/404.

[9] C. Pei, Y. Xiao, W. Liang, and X. Han, "Trade-off of security and performance of lightweight block ciphers in Industrial Wireless Sensor Networks," *EURASIP Journal on Wireless Communications and Networking*, vol. 2018, no. 1, pp. 1–18, 2018.

[10] A. Bogdanov, D. Khovratovich, and C. Rechberger, "Biclique cryptanalysis of the full aes," Cryptology ePrint Archive, Report 2011/449, 2011. [Online]. Available: https://eprint.iacr.org/2011/449

[11] J. Daemen and V. Rijmen, "AES proposal: Rijndael," 1999.

[12] M. R. Z'aba, H. Raddum, M. Henricksen, and E. Dawson, "Bit-pattern based integral attack," in *Fast Software Encryption*, K. Nyberg, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 363–381.

[13] C. H. Lim and T. Korkishko, "mcrypton - a lightweight block cipher for security of low-cost rfid tags and sensors," in *WISA*, 2005.

[14] T. Shirai, K. Shibutani, T. Akishita, S. Moriai, and T. Iwata, "The 128-bit blockcipher clefia (extended abstract)," in *Fast Software Encryption*, A. Biryukov, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 181–195.

[15] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vikkelsoe, "Present: An ultra-lightweight block cipher," in *CHES*, 2007.

[16] K. Shibutani, T. Isobe, H. Hiwatari, A. Mitsuda, T. Akishita, and T. Shirai, "Piccolo: An ultra-lightweight blockcipher," in *CHES*, 2011.

[17] J. Borghoff, A. Canteaut, T. Güneysu, E. B. Kavun, M. Knezevic, L. R. Knudsen, G. Leander, V. Nikov, C. Paar, C. Rechberger, P. Rombouts, S. S. Thomsen, and T. Yalçin, "Prince - a low-latency block cipher for pervasive computing applications - extended abstract," *IACR Cryptology ePrint Archive*, vol. 2012, p. 529, 2012.

[18] H. Mala, M. Dakhilalian, V. Rijmen, and M. Modarres-Hashemi, "Improved impossible differential cryptanalysis of 7-round aes-128," 12 2010, pp. 282–291.

[19] B. e. a. Collard, "A statistical saturation attack against the block cipher present," in *Proceedings of the The Cryptographers' Track at the RSA Conference 2009 on Topics in Cryptology*, ser. CT-RSA '09. Berlin, Heidelberg: Springer-Verlag, 2009, p. 195–210. [Online]. Available: https://doi.org/10.1007/978-3-642-00862-7_13

[20] I. Dinur, "Improved differential cryptanalysis of round-reduced speck," in *Selected Areas in Cryptography – SAC 2014*, A. Joux and A. Youssef, Eds. Cham: Springer International Publishing, 2014, pp. 147–164.

[21] F. Abed, E. List, S. Lucks, and J. Wenzel, "Cryptanalysis of the speck family of block ciphers," Cryptology ePrint Archive, Report 2013/568, 2013. [Online]. Available: https://eprint.iacr.org/2013/568

[22] N. Wang, X. Wang, K. Jia, and J. Zhao, "Differential attacks on reduced simon versions with dynamic key-guessing techniques," Cryptology ePrint Archive, Report 2014/448, 2014. [Online]. Available: https://eprint.iacr.org/2014/448

[23] H. Chen and X. Wang, "Improved linear hull attack on round-reduced Simon with dynamic key-guessing techniques," Cryptology ePrint Archive, Report 2015/666, 2015. [Online]. Available: https://eprint.iacr.org/2015/666

[24] H. Soleimany, C. Blondeau, X. Yu, W. Wu, K. Nyberg, H. Zhang, L. Zhang, and Y. Wang, "Reflection cryptanalysis of prince-like ciphers," *J. Cryptol.*, vol. 28, no. 3, p. 718–744, Jul. 2015. [Online]. Available: https://doi.org/10.1007/s00145-013-9175-4

[25] A. Canteaut, M. Naya-Plasencia, and B. Vayssière, "Sieve-in-the-middle: Improved mitm attacks," in *Advances in Cryptology – CRYPTO 2013*, R. Canetti and J. A. Garay, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 222–240.

[26] A. Canteaut, T. Fuhr, H. Gilbert, M. Naya-Plasencia, and J.-R. Reinhard, "Multiple differential cryptanalysis of round-reduced prince (full version)," Cryptology ePrint Archive, Report 2014/089, 2014. [Online]. Available: https://eprint.iacr.org/2014/089

[27] Y. Li, W. Wu, and L. Zhang, "Improved integral attacks on reduced-round clefia block cipher," in *Proceedings of the 12th International Conference on Information Security Applications*, ser. WISA'11. Berlin, Heidelberg: Springer-Verlag, 2011, p. 28–39.

[28] C. Tezcan, "The improbable differential attack: Cryptanalysis of reduced round clefia," Cryptology ePrint Archive, Report 2010/435, 2010. [Online]. Available: https://eprint.iacr.org/2010/435

[29] Y. Hao, D. Bai, and L. Li, "A meet-in-the-middle attack on round-reduced mcrypton using the differential enumeration technique," Cryptology ePrint Archive, Report 2013/756, 2013. [Online]. Available: https://eprint.iacr.org/2013/756

[30] Y. Wang, W. Wu, and X. Yu, "Biclique cryptanalysis of reduced-round piccolo block cipher," 04 2012, pp. 337–352.

[31] M. Minier, "On the security of piccolo lightweight block cipher against related-key impossible differentials," in *Proceedings of the 14th International Conference on Progress in Cryptology INDOCRYPT 2013 - Volume 8250*. Berlin, Heidelberg: Springer-Verlag, 2013, p. 308–318. [Online]. Available: https://doi.org/10.1007/978-3-319-03515-4_21

[32] D. Claraz, F. Grimal, T. Ledier, R. Mader, and G. Wirrer, "Introducing multi-core at automotive engine systems," in *ERTS2*, 2014.

[33] R. Mader, G. Winkler, and A. Graf, "Autosar based multicore software implementation for powertrain applications (2015-1-0356)," *SAE World Conference*, 2015.

[34] A. Göbel and D. Claraz, "A multi-core basic software as key enabler of application software distribution," in *ERTSS*, 2018.

[35] S. Kastner and T. Galla, "Unleashing the power of multi-core mcus by communication stack software distribution," in *EMCC - Munich*, 2019.

[36] C. Rebeiro, D. Mukhopadhyay, J. Takahashi, and T. Fukunaga, "Cache timing attacks on clefia," in *Progress in Cryptology - INDOCRYPT 2009, 10th International Conference on Cryptology in India, New Delhi, India, December 13-16, 2009. Proceedings*, ser. Lecture Notes in Computer Science, B. K. Roy and N. Sendrier, Eds., vol. 5922. Springer, 2009, pp. 104–118. [Online]. Available: https://doi.org/10.1007/978-3-642-10628-6_7

Session Th.1.C
Logical Execution Time

Thursday 2nd June

10:00

–

Room Pastel

A dynamic Reference Architecture to achieve planned Determinism for Automotive Applications

Denis Claraz*, Max J. Friese†, Hermann von Hasseln† and Ralph Mader‡
† *Mercedes-Benz AG* Stuttgart, Germany
* *Vitesco Technologies* Toulouse, France
‡ *Vitesco Technologies* Regensburg, Germany

***Abstract*—With the evolution of modern cars towards more distributed architectures, considering also an increase of the data volume, we see the clear need of increased determinism in all types of communications: inter-tasks, inter-core, inter-partitions, inter-ecus. In parallel, to master the classical cost/quality/time-to-market tryptic, platform approaches are needed more than ever, to allow standardization. Our purpose in this document is to describe how to combine standardization and determinism, in the automotive domain, and what should be put in place in term of architecture and design patterns.**
Therefore, we describe our vision and experience of a dynamic reference architecture, as a "real-time framework" for easy development and integration of functions on one side, and for projects configuration on the other side. We explain the different levels of detail required by different users, the variability management required to support different classes of applications. Then, we introduce the need for determinism, and clarify what level of determinism we are talking about and which different approaches used can be used. In particular, we present a new approach that provides determinism not only at runtime, like Logical Execution Time (LET), but also throughout successive development cycles.
In a next section, we present a deterministic reference architecture, that combines the two above mentioned objectives of standardization and determinism. We explain its main principles and the underlying constraints on the design of the functions. Following such an approach, we can achieve a certain degree of planned determinism (in opposition to an "inherited" one).
Finally, we illustrate our proposal with practical cases taken from industrial projects, mostly from powertrain domain controllers.

I. Introduction

From their early days in basic engine control, powertrain electronics have evolved into sophisticated computers for precise combustion management, exhaust after-treatment and drivetrain control. Yet, that forty-year evolution pales in comparison to the transformation of the entire vehicle's electronics and software architecture over the coming decade.
Over the years, new features for improved emission control, vehicle safety and comfort were incrementally introduced into the vehicle via new electronic control units (ECUs). The modern car now already contains as many as 150 ECUs which communicate with each other over in-vehicle networks such as CAN, Ethernet or FlexRay. This incremental expansion of the E/E landscape in a "distributed architecture" has, over the years, become the state of the art at most carmakers. As a result, the amount of software in the car continues to increase exponentially, with some estimates placing the number of lines of code in a premium car at over 700 million between 2025-2030, compared to an already massive 100 million lines of code in 2015.
Architects imagine a simple E/E constellation borrowed from the IT-world as a "North Star": a few centralized servers for computation, and distributed smart actuators for real-time mechatronic control. However, most established car companies have a long legacy of car platforms which must be maintained in parallel with the desire to introduce new features via simple E/E architectures. A single leap from the current distributed architecture of as many as 150 ECUs, towards a server-based architecture centered around a handful of high-performance computers is done most often in intermediate steps, via domain (Powertrain, Chassis, Body, Interior) or and cross domain controllers, which we call master controller. [1]

A. Repartitioning of Functions

While cross-domain integration in a master controller, e.g. responsible for vehicle motion, reduces the number of ECUs in the car, it greatly increases the complexity of software inside the device. The master controller is not only the home of existing functions like energy management, torque management and vehicle stability functions, but also contains a slew of new functions to manage electrification, connectivity, and the impacts of autonomous driving.
This E/E architectural trend affects the partitioning of the functionalities: They are divided in a low level sensing and actuation part and a high level computational part. The low level part will be still in a control device close to the corresponding sensors and actuators with hard real-time requirements, necessary for controlling a brush-less DC motor or a valve, just to mention some prominent examples. The algorithm part will move up to a higher architectural level in the E/E Architecture, like the Vehicle Motion Master Controller. The exchange of the sensor values, set point, etc. between low and high level ECUs will be realized with bus systems like CAN, CAN-FD and FlexRay.

B. Dynamic behaviour

Let us have a brief look on the architecture of automotive, especially powertrain, applications. They are typically based

on an AUTOSAR Classic real-time operating system according to the OSEK standard. The configuration of the OS is done in a way that there are several recurring tasks with different periods from 1ms up to 1000ms which are distributed on several cores of a micro controller. These tasks call runnables subsequently according to the need of the data flow demanded by the causal chain of the function.

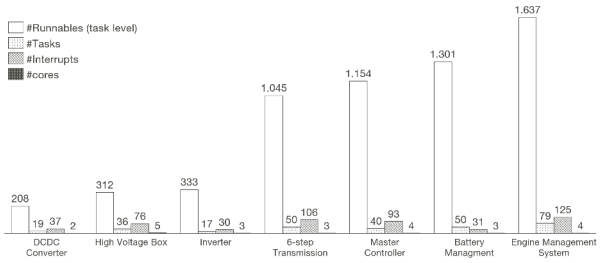


Fig. 1. Number of cores, tasks and runnables of different automotive applications

As one can see in Fig.1 the integration complexity with respect to the dynamic behavior of the system is different across various automotive applications. Typically the number of tasks and runnables increase with the size of the application and reaches its maximum for master-controller, battery management and engine-management-systems.
When distributing functions of these applications over multiple ECUs one needs to carefully consider the timing behaviour of the event chains of the controlled system. Fig.2 illustrates the basic principle of a distributed functionality with an ECU 1, responsible for reading the sensor values and transmitting them via CAN to the Master-Controller.
The higher level algorithmic portion located in the Master Controller will receive the sensor values and use this input to compute a setpoint for the actuator controlled by a transmission- or engine-controller, called ECU2. The setpoint

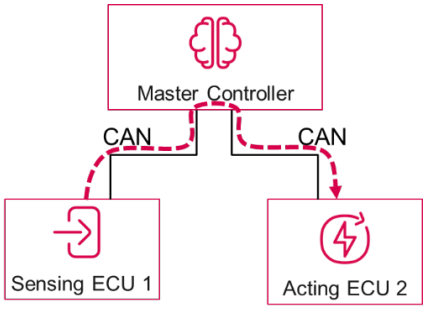


Fig. 2. Schematic of a control flow from a sensing ECU1 via the Master Controller to the acting ECU2, connected via CAN

is transmitted via CAN to the acting ECU which will use this to apply the latest setpoint value to the actuator. The timing between reading the sensor value and applying the setpoint for the actuator must fulfill the requirements of the overall function which needs to be realized in the timing constraints given by the physics or the mechanics.

C. System integration and validation

These distributed functions will be developed by independent teams, working on software for the different ECUs with different update and release cycles. From one release to another the desire of the System Integrators is to have a control on the changes and best be able to reduce the systems test cases to a minimum while still guaranteeing the correct functional behaviour and a safe operation of the system. Therefore mechanisms are needed which implement a deterministic behaviour to reduce the necessary tests for the validation and enable the release of subsystems. In the following chapters we will manly focus on mechanisms which can be applied for a single device in the communication network to ease integration in the more complex devices like master controller or engine management systems, which are typically based on multi core controllers supporting different safety partitions.

II. Determinism

The software technologies used in this context need to support this deterministic behaviour for static and dynamic aspects of the software system architecture. In the last years AUTOSAR Classic platform has been enriched in this direction. For the control of freedom from interference (FFI) safety partitions have been introduced in the AUTOSAR Classic standard. More recently, the release 20-11 [2] introduced the so called software clusters (SWCL) as independent development and build units with a defined interface layer: the software cluster connect to the rest of the system. This allows the development, validation and update of one software cluster, independently from the rest of the system. From a static architectural point of view the development and release of subsystems is covered by these approaches. Concerning the real-time behaviour of the system, an independence of the clusters needs also to be ensured: the update of a cluster should not impact the dynamic behavior of the others. This concerns not only the task behavior itself, but also the data communication between the clusters. In addition, different core to cluster mappings are considered: 1:1, 1:n, m:1, m:n, as shown in Fig.3

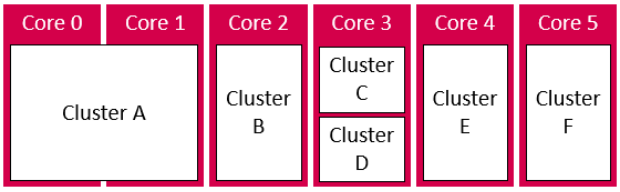


Fig. 3. Actual distribution of Clusters on Cores

A. What is determinism in the dynamic behaviour

In such distributed context, our main interest is the inter-task data communication, when a data is transmitted or received (written, read). In case of multi-core architecture, we want to avoid 2 types of problems: concurrent accesses to the same data (consistency), and unstable data age (determinism).

The first problem happens when the two competing tasks attempt "simultaneously" to write (on one side), and read (or write, on the other side) the same data or data set. This "data consistency" issue (in the sense of stability and coherency) is solved by a buffering strategy, similar to AUTOSAR RTE/IOC [3] [4], and is not in the focus of this paper.

The second problem happens when the relative positions of the tasks jitters over the time, generating an instability of the number of writes between two successive reads, or vice-versa. For instance, in Fig.4, the 2 communicating tasks have a same period of 5ms, and also a deadline of 5ms: They can execute anywhere inside the shaded area (within the period), depending on the local situation on CPU load, interrupts, blocking semaphores, other tasks running, etc... Therefore their relative order is not guaranteed, and by consequence, the actuator function (blue task) may respond with an unstable delay to an input signal variation (yellow task):

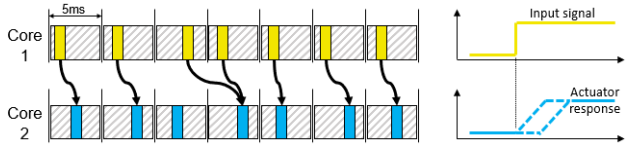


Fig. 4. Impact of Task jitter on determinism of flow, and response time of function

This "non-deterministic" data age can be solved by applying different scheduling and inter-task communication strategies, that will be detailed in the next chapter. As shown here, our main goal is to remove this uncertainty on the tasks communication paths, i.e. to ensure a correct sequence of causal chains.

As added benefits of deterministic communication, we can mention 2 aspects:

- Possibility to parallelize independent parts of algorithms, setting "rendez-vous" where data communication happens before the next parallel slot (chains of "fork join" patterns.) [5]. In particular, the Logical Execution Time paradigm (LET, see II-B3) has been foreseen as a mean to enable multi-core introduction in automotive systems already in 2016 [6] and in 2018 [7]. Furthermore, it was introduced in AUTOSAR Timing Extension [8] in 2020.
- Mean to set time-budgets to tasks and to increase the decoupling between parts of the SW: as long as a task stays within his pre-defined time-interval, there is no impact on the other ones.

In our case (AUTOSAR OS, online scheduling), deterministic scheduling is not used to control the worst case execution time (WCET) of the task (avoiding concurrent access to shared resources like busses, memory, peripherals, ...), like required in offline scheduling strategies.

Finally, there can be different points of view on determinism, and on implementation level, independently of the chosen strategy, an uncertainty remains at what absolute point in time a given data is transmitted or received.

B. Different implementation strategies

In this paragraph, we describe different implementations of determinism, that may fit to the requirement, depending on the needed precision. These strategies (applied on successive projects, as described in section V) are:

1) Strategy 1: Short deadlines (Fig.5):
A basic and simple determinism can be reached by setting Task deadlines, periods, and offsets in a way that the communicating tasks will never overlap. For instance, 2 Tasks of period 5ms on different Cores can have a deterministic communication if they have a short deadline (e.g. DL=2ms) and an offset (e.g. O=2.5ms) between them. Their execution domains (shaded areas) do not overlap each other, and therefore an order is guaranteed. One has just to ensure that each Task fits to its timing requirements, by setting the appropriate system configuration (priority, ...).

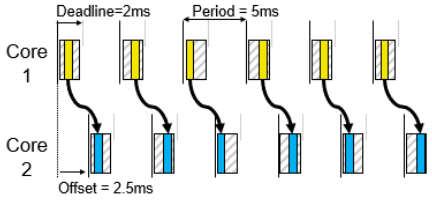


Fig. 5. Short deadline tasks with offsets ensure a deterministic flow

2) Strategy 2: Time Determinism (TD) algorithm (Fig.6):
In the overall schedule of the TD tasks, data communication of each task is done in a gap left empty before its next period starts. The deadline of a TD-task is therefore shorter than its period: At maximum, it is equal to its period minus a fixed gap reserved for the bi-directional communication. A periodic task is created to communicate the data for all TD tasks. Inside each TD task, local buffers are used (similar to AUTOSAR implicit communication), instead of directly reading the global data. Each communication task publishes the buffers of the preceding TD task into the global data ("Terminate"), and copies the global data into local buffers of the succeeding TD task ("Release"). Therefore, the real position of the task within its deadline does not matter. What matters is the position of the communication gaps vs. each other. With such principle, the jitters of the 2 functional tasks have no impact on the data age, as long as they finish before the communication task: a data computed in the cycle N of a task will be available for the other task in the cycle N+1.

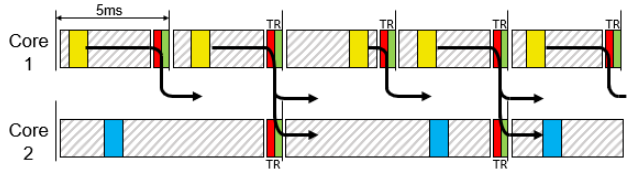


Fig. 6. TD communication done in end of cycle task. A terminate (T) driver publishes the data (computed in the previous cycle) to the global memory, and a release (R) driver fetches the data from the global memory for the next cycle computations. This example shows clearly that the flow between the fast (yellow) task & the slow one (blue) is perfectly deterministic, independently of the jitters of the tasks in their respective intervals.

3) Strategy 3: Logical Execution Time (LET) paradigm (Fig.7):

In the LET paradigm [9], tasks execute anywhere within their designed time-interval (like in previous strategies), and the communication between them is done at the limits of the interval, using dedicated tasks (called "Release" and "Terminate"). The real determinism of the communication depends on how precise is the execution of these communication tasks. In the LET theory, the communication is done at each interval border, in a "zero time". But in the real implementation, the runtime for the data communication needs to be considered, and it must be ensured that the communication is done at the limits of - but inside - the interval. This constraint requires a careful planning of the intervals: For instance, enough margin needs to be reserved in the interval for the task (considering its WCET), but also for the communication (release + terminate) [10] [11]. Also, conflicts (overlaps) between releases and terminates of different intervals shall be avoided, as they may end-up in conflicting accesses to the same data, with the resulting loss of determinism and coherency.

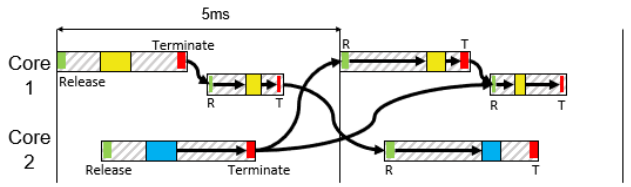


Fig. 7. LET communication: Different intervals/tasks on different cores. Communication is done at limits of each interval in the Release & Terminate drivers.

TD and LET principles are very similar: In both cases, dedicated tasks are in charge of the inter-task communication. In both cases, an interval is defined for each functional task, which represents the time slot inside which the task is allowed to execute. In TD, the intervals are in practice nearly equal to the periods, whereas it should not be necessarily so. In LET, the intervals are much shorter than the period, and several intervals are set for the same period, whereas it should not be necessarily so. In TD, the actual communication of the data is done outside of the task frame, when the next (periodic) communication tasks executes. In LET, the actual communication of the data is done at the limit (inside) the interval.

4) Strategy 4: Planned LET (Fig.8):

The objective of this strategy is to ensure determinism of the communication between different tasks of different clusters. It also ensures a stability of this communication along the project development cycle, despite the successive upgrades: the communication between clusters is kept as planned.

The LET communication described above ensures determinism of communication between tasks at runtime. But in case of distinct clusters, each one with its own tasks, it cannot be ensured that the communication is still done at the same position, when one of the 2 clusters gets updated (e.g. re-flashed). Even if the architectures (number of tasks, periods, offsets, lengths) of all clusters are frozen (which might not be the case), it is possible that in the new upgrade of cluster B (e.g. on core 3), the production of the inter-cluster data is moved from the 1st to the 2nd interval. Consequently, the behavior of the cluster A (on cores 1 & 2) would be affected by this modification of cluster B, if the tasks of the different clusters would directly communicate. Here, LET alone would bring a certain determinism at runtime, but would not offer any determinism over development cycles. To avoid this effect, cross-cluster communication tasks are created, that define communication channels between the clusters, and that are fixed on system design level, in term of position, and data to be transmitted. In other words, we have a component model for the dynamic aspects of the architecture. In the example Fig.8, we show that for cluster A distributed on 2 cores, that the (green) importing task ("XccIn") can even be implemented on a different core than the (red) exporting task ("XccOut").

Finally, this concept has 2 additional benefits:

First, it compensates one weakness of the LET paradigm: its single interval scope: With LET, each interval is treated as an own execution container, for which a high degree of data integrity is ensured (determinism, stability, coherency). But in reality, a complete system is based on several - or even many - intervals. and some of them consume the same information. Then, LET does not answer to the question of data integrity for a complete set of tasks. For instance, one might have the need that all intervals of same period (e.g. 10ms) share the same value for a data coming from another period (e.g. 10ms) or cluster. We might even need to define, in a series of successive intervals, which one should work with the latest value, while the others may work with an older information ("fast causal chain"). With Xcc communication, the producer cluster decides at which instant it communicates certain information to all consumers, and each consumer decides for itself at which instant it imports the information. There is a total decoupling between the actual computation/use of the data and its communication.

As second side effect, the Xcc communication reduces the conversion overhead between the clusters, compared to a direct task-to-task communication: The import of a data from another cluster is done only once, even if this data is used in many tasks (intervals).

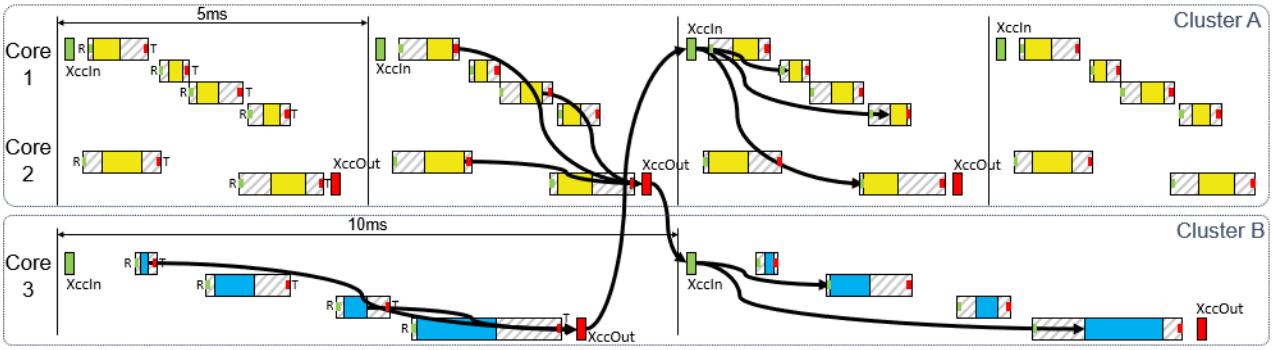


Fig. 8. Cross cluster communication: Clusters A & B communicate at pre-defined & fixed points (Xcc tasks), which are independent of the internal details of each cluster. The Xcc tasks are in charge of distributing (respectively collecting) the inter-cluster data to the different consuming (resp. producing) tasks inside the cluster. LET communication is applied for the internal flow of the cluster.

This strategy is in use in our newest program, which is described further in the chapter V.

C. For which data?

As seen in the previous paragraph, our motivation for determinism is the removal of the uncertainty of the data communication path between tasks, due to task jitters. Achieving a fixed age is a clear advantage of such scheduling, like LET, but there is a price to pay: First, the CPU utilization is not as optimized as it could be in a conventional system; Then, the duration of the Release/Terminate at the limits of the LET interval may become significant in proportion to the interval duration, in case of big data flows.

Therefore, there is a need of a selective approach, where determinism is applied only on a subset of critical data. For the other data, classical data integrity (e.g. stability, or coherency) approach is sufficient, and more optimized.

For instance, the LET approach could be limited to safety critical data, or for signal flows where the instability has system impact. Typically, it may depend also on the nature of the data. Control data (booleans, state machines, indexes, counters, ...) may have a stronger negative impact on the functional behavior in case of non-determinism, than more ”continuous” data, which have low gradients. Finally, this is the consumer function of the signal, which knows better, whether a non-deterministic behavior of its inputs may have or not a negative effect on the strategy.

In the case of Xcc communication, the objective is rather to concentrate the communication flow between the clusters in a few a-priori and stable defined channels, to reduce the impacts of upgrading one cluster. The challenge is here to properly map the exchanged data on the right Xcc communication tasks. On the consumer-cluster side, the different intervals and frequencies where the imported data is used have to be considered, together with the availability and location of XccIn Tasks. The idea is to have one single point of import, even if there are several consuming tasks/frequencies. On the producer-cluster side, a similar approach has to be followed, and finally the flow cross clusters has to be designed on overall system level, ensuring a coherency of the communication channels between the clusters.

For sure, such an approach is easier when the clusters are decoupled, and exchange few information. In case of highly coupled clusters with several hundreds of interfaces, the a priori definition of all communication channels seems impossible, and in this case, a mixture between top-down & bottom-up approaches need to be applied.

III. Reference Architecture

Objectives of a reference architecture:

To facilitate the SW reuse, it is necessary that the reusable components and the reusing projects are developed according to a common framework. This framework comprises many facets, like a functional partitioning, a layered architecture, design patterns, coding rules, and more. One particular facet of this framework is a dynamic reference architecture, that ensures a correct real-time behaviour of the functions and projects.

Technical content:

This reference architecture defines standard events, operating system tasks, as well as some transverse configuration items. Events are abstract artifacts that define timing properties like periods, deadlines, activation pattern (sporadic, periodic, aperiodic, ...), available phases (abstract way of specifying a sequence constraint [12]), the behavior in case of system (core synchronous) transition [13] [5], and other behavioral details. Tasks are concrete artifacts that define not only the basic OS configuration (core, priority, multi-activation, use of cooperative resource, . . .), but also extensions like chaining to other tasks (if any), detailed LET configuration, memory section, asil level and more implementation details. Tasks are linked to events by a n:1 relationship, in the sense that n different tasks can implement the same event, while one event is implemented by at least one task. If different tasks implement the same event, then they comply individually and collectively to the event’s constraints. For instance, if 2 tasks of the same event are chained, the complete chain fulfills

the timing constraints of the event (in particular period & deadline).

In total, the complete definition of an event requires around 10 parameters, and even some more for a task (Fig.9).

Event parameters
Type (Trigger / Transition)
Activation pattern (sporadic / aperiodic / periodic)
Period
Deadline
Unit (ms / °crk)
Phases
Exclusive group (functional exclusions)
MFQL (behavior in case of burst)
AUTOSAR configuration
Owner (responsible module)
Description

Tasks parameters
Related Event
Core
Priority
Type (LET / TD / BET)
Cooperative group(s)
Multi-activation
Auto-start
Length/duration (LET/TD)
Offset (LET/TD)
Chaining (BET)
Memory section
AUTOSAR configuration
Description

Fig. 9. Parameters of events and operating system tasks. They are defined centrally in the reference architecture, to ensure a good synergy between functions and projects.

Functional content:
Functionally, the reference architecture contains different categories of events and related tasks:

- Angle based events and tasks are necessary for engine control applications, which have the most complex architectures. We can find here events of different periods (top dead center, engine rotation, camshaft rotations, ...), different phasing (different angular phases vs. top dead center). These events can be periodic or aperiodic, in all cases depending on the engine rotation speed. There are also engine related transitions, like engine stalling or cranking events, which require specific function initialization.
- Time based events and tasks are ranging from 1ms to 1000ms. For some of them, different deadlines are provided. For instance, most of 10ms calculations use a 10ms deadline, but for some other, a short deadline of e.g. 3ms is necessary. For this, different events, and different sets of tasks are provided in the reference.
- Kernel events and tasks for the infrastructure / Basic Software (BSW), as well as some function dedicated tasks (communication, watchdog, safety, background, CPU load, etc...). These tasks are also part of the reference, but are in general not open for Application Software (ASW) integration.
- Finally, initialization events and tasks are provided, that correspond to transitions between different ECU modes: reset, shutdown, entry or exit into RUN or POST-RUN modes, etc ...

At the end, the complete reference for a four core engine control system contains 80 events & 200 tasks.

How to use for functions:
Function developers refer to standard events to specify the timing requirements (therefore integration constraints) of their functions. They cannot refer to standard tasks for the same, as it would tight them too much to a dedicated project configuration, and therefore reduce their re-usability. As mentioned above, if a project implements different tasks for the same event, any of these tasks is guaranteed to respect the timing constraints of the event. Therefore, the timing properties of a function will be respected in the project as long as it is integrated in a task implementing the specified event.

In particular, core allocation is not a requirement of the function. Eventually, an affinity versus a certain core property (availability of double precision FPU, lock-stepped core, ...) or versus other functionality (”be on the same core than ...”) might be a valid requirement, but has to be used carefully as it might end-up in a non solvable situation.

Seen from function development perspective, the reference architecture provides sufficient information to allow the specification of the function’s integration constraints. These constraints are precise enough to ensure a correct dynamic behaviour of the function, and abstract enough to comply with diverse integration project, based on different microcontrollers (therefore available cores), integrating different functions (thereof customer plugins), having different system configurations (hybrid vs. pure combustion engines vehicles, diesel vs. gasoline, engine control vs. domain controller,etc..), etc...

How to use for projects:
Considering the high number of tasks required (up to 200 on a 6 cores application), and the need of overall coherency, configuring such a project is a significant effort, even if it has just to select into a large library of existing tasks described in all implementation details.
Therefore the reference architecture provides a series of architecture variants (”reference architecture projects”), which correspond to our main use cases, like e.g. 3 & 4 cores engine management system, 3 cores transmission systems, or single core selective catalyst reduction system. The left pie-chart in Fig.10 shows the percentage of projects using one or the other variant, over a selection of 70 projects in development. We see, for instance, that only 21% of the projects are not based on any variant, due to their atypical architecture.

In addition to the case-relevant event and tasks, each of these variants comprises transverse configurations, that can only be set at multi-task or multi-event level. This is the case, for instance, of sets of tasks that need to be enabled or disabled at once at given phases of the ECU life.

Therefore, a project architecture is based (reuse by reference) on the variant, which is closest to its configuration. From this variant, the project can:

- reuse any artifact from the reference as is
- ignore any artifact of the reference (not needed)
- adapt any artifact of the reference (deviation)
- create its own artifact w/o restriction.

In any of these cases, the project is responsible to ensure that the timing requirements of the events are respected, for any of the underlying tasks.

Finally, the most usual case is that the project at Vitesco Technologies reuse one variant and all embedded artifacts with very few specializations, as shown in Fig.10: Nearly 80% of all tasks used by our selection of projects are reused from the reference architecture without any modification.

IV. Reference Architecture for Determinism

A. Reference Architecture for Time Determinism

The reference architecture described in the previous chapter is a living practice, and fits to a well-established product group such as combustion engine control. With the introduction of electrification and new domain-oriented architectures, where more determinism is required, this reference needs to be adapted. The new reference architecture will be simpler in the sense that there will be less complex events (related to engine rotation), but a simple timing event previously described by a period, offset and deadline, will now in addition be split into a series of LET intervals each one defined by an additional offset and a duration.
The challenge is then to define a priori, w/o the full knowledge and background of potential user functions, a standardized organization of these intervals, that is kept stable over the development loops and across projects.
This proceeding is contrary to most of the current LET projects, where the architecture has to be regularly re-adjusted to satisfy fast communication chains (a bottom-up approach where the architecture is adapted to the functions, and not vice-versa).

Below are listed some principles of this LET reference architecture, that we believe can satisfy our upcoming projects.

1) Distribution over the period: In order to balance the CPU load, the intervals are not "grouped" at the beginning of the period, but rather mostly equally distributed over the complete period. For instance in case of 5 intervals of 2ms length and 1000ms period, instead of concentrating them in a reduced slot of approximately 10 to 20ms, they are distributed over the complete 1000ms period. This principle provides flexibility in the later evolution of the architecture, as it gives space for moving or expanding existing intervals, or adding new ones, without reworking the complete scheduling.

2) Ratio interval length vs. WCET of task: We know that we need a minimum margin between the WCET of the task, and the length of its interval, to let enough time for interrupts, higher priority tasks, and for the release & terminate operations. But ensuring a minimum margin does not mean that this minimum value has to be applied: any longer interval is also valid. Therefore, the ratio between the interval length and the WCET of the task has to be above a minimum threshold, but does not have a maximum. Having a long interval, even if we know the WCET of the task will be much shorter is not an issue, and reduces the "stress" of the scheduling, by letting time for other tasks to execute. The important point here is that the position of data transfer, at the limits of the interval, is fixed, and in accordance with the communication needs between tasks. For sure, this ratio depends also on the number of intervals for the given period.

3) Minimum length of interval: A very fine granularity of intervals lengths requires a high resolution of the LET system, and therefore a high CPU load overhead. Finally, this concerns not only the lengths of the intervals, but also the offsets, and the periods.

4) Maximum number of intervals: The number of intervals for a given period and core has to be limited. For instance, even on small periods like 100 or 1000ms, we limit ourselves to a reduced set of intervals, even if we could introduce a lot. The reason here is to limit the integration possibilities, keep the architecture relatively simple, and reduce the OS overhead.

5) Overlap between intervals of different periods: In general, for a given core, the intervals are organized to avoid overlapping situations. Nevertheless, keeping this principle for all periods is particularly problematic when periods are quite different. For instance avoiding overlap between a 1000ms interval and a 5ms -even 1ms- interval introduces the high constraint that even for tasks of slow period, therefore relaxed deadlines, the interval length must be kept very short to avoid such overlaps. To avoid such constraints 3 groups of periods are defined, inside which no overlap can occur. On the opposite, cross different groups, an overlap is fully possible. Therefore, a 5ms interval will overlap a 1000ms interval. This is not contradicting the LET paradigm, and the communication between both tasks will be done at each intervals limits.

6) Core independence: Having identical architectures on different cores have different advantages. First, it simplifies the overall architecture. Second, it allows the design of the functions independently of the core, providing an important flexibility at integration time. Another advantage is the possibility to implement "fork-join" patterns, where necessary. For this reason, we have designed a reference architecture with 2 types of clusters/cores: fast and slow clusters, to cover different use cases. But 2 fast (respectively slow) clusters have exactly the same architecture.

7) Cohabitation with non-LET tasks: The LET architecture must reserve sufficient empty space for non LET tasks, from the kernel, or from non-deterministic application SW. This non-LET part of the SW usually has more relaxed timing constraints, and will get executed in the "holes" of the LET scheduling.

8) Constraints vs. Xcc tasks: For most of the periods, one XccIn and one XccOut are reserved, for inter-cluster communication. In some cases (fast frequencies), no cross cluster communication will be done. In some other (highly loaded frequencies), a double Xcc communication is planned, to allow the insertion of fast communication paths. By default the Xcc communication tasks are added at the begining (for XccIn) and at the end (XccOut) of the period ("chain of LET intervals").

As result of these principles, we defined a standard architecture, for which we give an abstract in Fig.11.

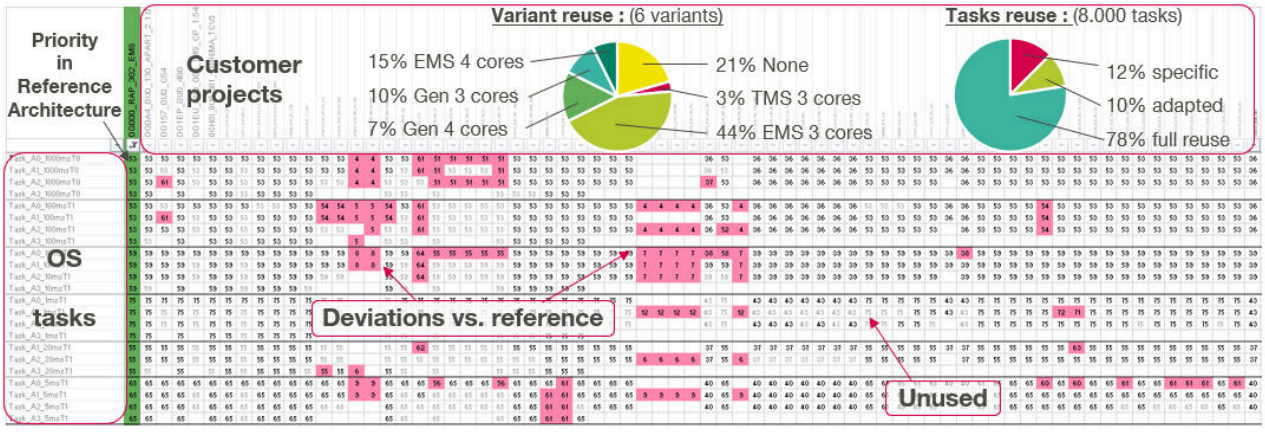


Fig. 10. Comparison of OS tasks priorities between a selection of 70 projects and the reference architecture. Each line represents a task from the reference. Each column represents a project. The green column gives the priority defined in the reference. A red cell indicates that the project deviates from the reference. A grey-shaded cell indicates that the task is not used.

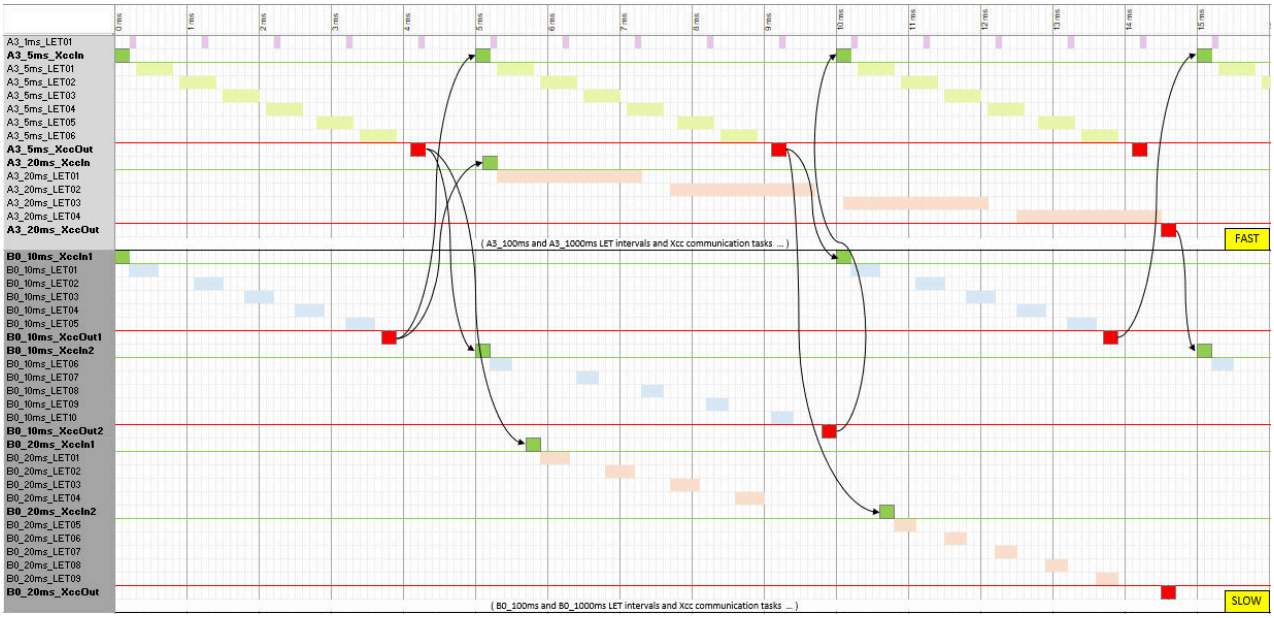


Fig. 11. A simplified LET-RA shown with two flavors: “Fast” and “Slow”. Each flavor can be mapped onto a core, and be used to implement a Software-Cluster, or parts of a Software-Cluster. On the “Slow” flavor there are two sets of LET intervals with their own Xcc-tasks, which can be used for different functional causal chains. On the B0 cluster, Xcc “communication channels” (i.e. extra Xcc tasks) are inserted in the middle of the 10ms and 20ms chains, to satisfy some short causal chains. Note that this is a gross simplified picture.

B. Requirements for, and impact of RA on function design

As described in the next chapter ‘Practical Example’, the strategy up to now was in essence to adapt existing static and dynamic architectures to legacy functions and their software architecture. The drawbacks of such an approach have been outlined above, and a more planning is in need here.

With the introduction of a Reference Architecture for Time Determinism it is clear that some restrictions and rules on the functions design level are necessary. This is by no means something new, as for example the Autosar component model also puts some constraints on the function design. For Time Determinism and the introduction of LET, which results in a component model for the dynamic aspects of the architecture, similar constraints needs to be formulated.

As already stated, without the full knowledge of potential user functions, it is a real challenge to define the Reference Architecture. But, there is a lot of implicit information contained in legacy projects and from foreseeable development of user functions, which can be used to derive some minimal assumptions and constraints for function design.

Some fundamental basics of a function design include:

1) In the case of a periodic functions, the frequency with

which these functions shall be called. It is helpful to avoid frequency dividers, because these would unnecessarily enlarge LET intervals and might lead to a waste of time resources. Although, to minimize complexity, it is also of importance to confine the number of task (or LET interval) periodicities. This is a tradeoff, and depends clearly of the kind of software project and user functions. In our experience there is generally enough information available to decide for a set of periodicities.

2) In the case of event-triggered functions, the information on a worst case response time of a causal chain in which these functions are embedded is needed. If planned to embedd event-triggered functions into LET-intervals, it should be coordinated with the overall set of periodicities, and if over- or undersampling is appropriate.
3) In general, the embedding of functions in one or more causal chains, together with minimal latencies requirements these causal chain have to fulfil,
4) or, at least, the predecessor and successor relations of functions which define execution order constraints.
5) The description of the dependency of functions on other functions in order to build up a data dependency graph. Such data dependency graph are used to optimize implementations of causal chains or resolve concurrency issues on the LET-interval structure. The data dependency graphs is not only needed in order to verify requirements of causal chains in a Software-Cluster, or among Software-Clusters, but may also be necessary in cases where a Software-Cluster has to be mapped onto more than one core (parallelization).
6) A classification to which SW-Cluster functions are assigned to.
7) An estimate of the resource consumption of functions. These include memory consumption, the the amount of data to be buffered for determinism (in or among SW-clusters: therefore a detailed analysis of the data dependency graphs are necessary, see (5)), estimates of (worst-case) execution times in order to map functions onto LET-intervals. It is also of crucial importance to have information on the volatility of the runtime of functions. In most practical cases, static WCET analyzes are not feasible, but information on the functional content often helps to classify these functions.
8) There are also a number of non-functional requirements which have to be considered. Among them are a classification of functions according to ASIL-levels. This might lead to restrictions on the mapping fo ASIL-relevant functions to cores of multi-core controller which a have Lock-Step core behind. Another important restriction is the placement of functions related to signal mappings and network diagnostics (middleware), which should be placed with respect to the capability to distribute basic software functions onto cores. There is also an important restriction coming from the Autosar Software architecture: atomic Software components must be mapped onto one core, i.e., the runnables are not allowed to be distributed among cores.

This list has to be understood as a minimal set of user function properties, or properties of a given Software architecture. On the other hand, this list can be used as a minimal set of impact properties for the design of user function.

Depending on the Software architecture of user functions it might be appropriate to define several "flavors" of task structures of a Reference Architecure, which then can be used on different cores of a target multi-core controller. With this basic information at hand, the following required information can be gathered:

1) which "flavour" of the RA should be instantiated on each core of the target controller,
2) an analysis and mapping of functions onto tasks or LET intervals of a SW-Cluster (e.g., corresponding to existing dynamical concepts), to fulfil inter-SW-Cluster communication,
3) an analysis and placement of intra-Cluster-Communication (Xcc tasks), in order to fulfil causal chain requirements.

V. Practical Example

In this section we give a brief history of how multi-core architectures have become part of the development for the central powertrain controller at Mercedes-Benz Cars. Following this, we discuss the demands of the upcoming generations and finally we give an outlook for future generations which provides additional motivation to pursue LET reference architectures.

In the following, we refer to five different generations of central powertrain controllers: Gen_A (launched around 2012), Gen_B (launched around 2016), Gen_{C0} (launched around 2020), Gen_{C1} (launched around 2020), and Gen_D (to be launched around 2024).

A. Evolution of the Dynamic Architecture of the Central Powertrain Controller

The development of the dynamic architecture of the central powertrain controller has been influenced by multiple factors. Firstly, the hardware architecture has an impact as it determines the number of available cores and the communication between software allocated on these cores. Secondly, the software architecture plays an important role. A monolithic architecture (cf. [14]) can not be freely distributed, wheres a component-based architecture with well-defined interfaces also gives more flexibility for the scheduling. It stands to reason that the amount of signals exchanged via the interfaces also impacts the scheduling and therefore the dynamic architecture. The more signals are communicated, the more scheduling constraints potentially exist. In Fig.12 the increase in the amount of signals exchanged within the OEMs application software is depicted. More precisely, the figure shows signals exchanged with high frequencies between generations of the central powertrain. Please note, that Gen_{C1} is installed in electric vehicles (EV) and therefore has less signals as no combustion engine is controlled. Starting from Gen_B, the determinism is used

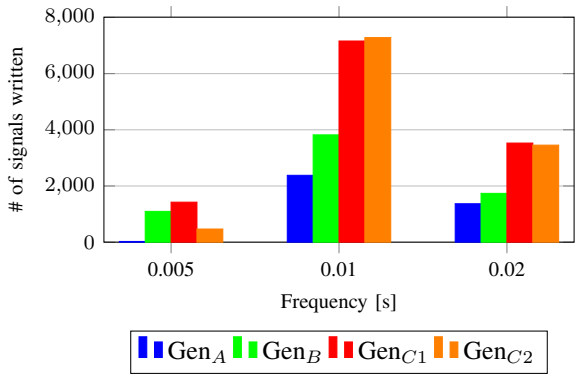


Fig. 12. Signals Written with Frequency 5, 10, and 20ms in the ASW of different generations of the Central Powertrain Controller

for the communication of 5 and 10ms signals. Finally, there are external factors influencing the scheduling. An example is development of *connected cars* [15] as it indirectly influences the dynamic architecture because it imposes new demands, e.g. for service-oriented communication.

1) Gen$_A$: This is the first generation taking the role of a domain controller. It was using a single-core controller with a multi-task system which was scheduled using a priority-based preemptive algorithm. For this, basic data consistency methods were introduced.

2) Gen$_B$: In this generation, tasks are still scheduled using a priority-based preemptive scheduling. Since it is the first generation with a multi-core controller, in Gen$_B$ the first cautious utilization of multi-core was introduced. Tasks were distributed on the different cores with limited use of task chaining techniques. These techniques allow rule out any overlap of certain tasks across two cores. See also page 3 strategy 1.

3) Gen$_{C0}$: In Gen$_{C0}$ the use of multi-core scheduling has been developed further. To cope with the arising challenges different mechanism for time determinism have been introduced. While still using the same priority-based preemptive scheduling, and mostly reusing existing OS-tasks from the previous generation, but implementing data determinism mechanism for tasks with fast periods. Although this is not strictly following the LET approach, it was the first time communication mechanisms following the LET semantics were implemented. Furthermore, synchronization of the clocks of some of the powertrain ECUs has been implemented to orchestrate network communication with other controllers within the domain. See also page 3 strategy 2.

4) Gen$_{C1}$: In this generation, time determinism was rolled out for almost all application-level tasks. For the first time a LET-based scheduling for the application software was used in this ECU to ensure for data determinism, enable load better balancing by parallelization, and implementation of causal chains by design. To achieve this, a detailed data dependency analysis for the design of causal chains was developed [6]. See also page 4 strategy 3.

5) Gen$_D$: With Gen$_D$ a cluster-concept is introduced allowing more flexible development. This also comes with more sophisticated methodologies for architectural work for both, the static software as well as the dynamic software architecture. On the *static* as well as on the *dynamic* side, architects are working with component models. While the *static* side architects can rely on the well-established AUTOSAR component model, for the *dynamic* side new concepts have been developed. More precisely, a reference architecture with SL-LET-like [16] is implemented on the dynamic side, which we call *planned* LET. The primary objective is to further improve stability of the development process during different phases of a project. See also page 4 strategy 4.

B. Future of the of the Central Powertrain Controller

As depicted in Fig.12, the amount of signals communicated with fast frequencies steadily increases with every generation of the central powertrain controller. This is also expected to happen from Gen$_C$ to Gen$_D$. In addition, the introduction of the cluster-concept allows to configure different configurations consisting of varying clusters. In the maximum configuration we expect an increased demand of deterministic communication by 25 to 30 percent compared to Gen$_{C1}$. A considerable share of this increases comes from the EV part in Gen$_{C2}$, however, it is also possible that more functionalities are integrated in new clusters.

VI. Conclusion and Outlook

In the paper we have shown the evolution of deterministic scheduling and data communication approaches of the past decade in the powertrain domain. From single core to multi core, from a project architecture to a reference architecture. We have shown how a reference architecture for the dynamic aspects of an embedded automotive system is applied in practice at Vitesco Technologies.

The evolution of the central power-train controller at Mercedes-Benz Cars clearly shows the gradual establishment of LET-based data determinism mechanisms. Initially, the goal was to utilize multi-core architectures in a safe manner, With LET now being successfully implemented, we strive for an independent dynamic software architecture with LET frames as the components.

Especially the application of a dynamic reference architecture in combination with the AUTOSAR Flexibility concept will enable an independent development and release of SW-clusters and so it will help to manage complex embedded automotive software systems. As a next step these approaches can be extended to in-vehicle networks in the sens of a System Level LET.

References

[1] R. Mader, G. Winkler, T. Reindl, and N. Pandya, "The cars electronic architecture in motion: The coming transformation," in *42nd International Vienna Motor Symposium*, 2021.

[2] AUTomotive Open System ARchitecture. (2020, 11) Requirements on system template. [Online]. Available: https://www.autosar.org/fileadmin/user_upload/standards/classic/20-11/AUTOSAR_RS_SystemTemplate.pdf

[3] D. Claraz, F. Grimal, T. Ledier, R. Mader, and G. Wirrer, "Introducing multi-core at automotive engine systems," in *ERTS2*, 2014.

[4] AUTomotive Open System ARchitecture. (2017, 12) Specification of rte software. [Online]. Available: https://www.autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_SWS_RTE.pdf

[5] R. Mader, "Design pattern in automotive multi-core embedded realtime environment and their support by scheduling strategies," in *Parallel Heidelberg*, 2017.

[6] J. Hennig, H. von Hasseln, H. Mohammad, S. Resmerita, S. Lukesch, and A. Naderlinger, "Poster abstract: Towards parallelizing legacy embedded control software using the LET programming paradigm," in *2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), Vienna, Austria, April 11-14, 2016*. IEEE Computer Society, 2016, p. 51. [Online]. Available: http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=7460013

[7] A. Biondi and M. D. Natale, "Achieving predictable multicore execution of automotive applications using the LET paradigm," in *IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2018, 11-13 April 2018, Porto, Portugal*, R. Pellizzoni, Ed. IEEE Computer Society, 2018, pp. 240–250. [Online]. Available: https://doi.org/10.1109/RTAS.2018.00032

[8] AUTomotive Open System ARchitecture. (2020, 11) Specification of timing extensions. [Online]. Available: https://www.autosar.org/fileadmin/user_upload/standards/classic/20-11/AUTOSAR_TPS_TimingExtensions.pdf

[9] C. M. Kirsch and A. Sokolova, "The logical execution time paradigm," in *Advances in Real-Time Systems (to Georg Färber on the occasion of his appointment as Professor Emeritus at TU München after leading the Lehrstuhl für Realzeit-Computersysteme for 34 illustrious years).*, S. Chakraborty and J. Eberspächer, Eds. Springer, 2012, pp. 103–120. [Online]. Available: https://doi.org/10.1007/978-3-642-24349-3_5

[10] R. Mader, "Implementation of logical execution time in an autosar based embedded automotive multi-core application," in *Dagstuhl Seminar 18092*, 2018.

[11] M. Alfranseder, S. Kuntz, M. Kardos, and R. Mader, "Logical execution time in the automotive environment," in *Embedded Software Engineering Conference - Sindelfingen*, 2018.

[12] D. Claraz, S. Kuntz, U. Margull, M. Niemetz, and G. Wirrer, "Deterministic Execution Sequence in Component Based Multi-Contributor Powertrain Control Systems," in *ERTSS 2012*, 2012.

[13] M. Alfranseder, R. Mader, T. Krapf, M. Niemetz, J. Mottok, and C. Siemers, "An efficient partitioning strategy for runnables in weakly dependent tasks on embedded multi-core systems," in *ERTS2*, 2014.

[14] M. Staron, *Automotive Software Architectures*. Springer International Publishing, 2017, vol. 1. [Online]. Available: https://doi.org/10.1007/978-3-319-58610-6

[15] R. Coppola and M. Morisio, "Connected car: Technologies, issues, future trends," *ACM Comput. Surv.*, vol. 49, no. 3, pp. 46:1–46:36, 2016. [Online]. Available: https://doi.org/10.1145/2971482

[16] K. Gemlau, L. Köhler, and R. Ernst, "Efficient run-time environments for system-level LET programming," in *Design, Automation & Test in Europe Conference & Exhibition, DATE 2021, Grenoble, France, February 1-5, 2021*. IEEE, 2021, pp. 749–754. [Online]. Available: https://doi.org/10.23919/DATE51398.2021.9474257

The synchronous Logical Execution Time paradigm

Fabien Siron*†, Dumitru Potop-Butucaru†, Robert de Simone†, Damien Chabrol* and Amira Methni*
*Krono-Safe, Massy, France
†Université Côte d'Azur, Inria, France
*firstname.lastname@krono-safe.com
†firstname.lastname@inria.fr

***Abstract*—Real-Time industrial systems are not so much of those that have to perform tasks incredibly fast, but in a time-predictable manner; they rather focus on meeting previously specified timing requirements in a provable way. Consequently, time must be taken into account from the very start of the design. However, exact timing constants may not be available yet in early design stages as they may depend on the target. In answer, formalisms based on the Multiform Logical Time have been introduced to abstract real-time durations. The Synchronous-Reactive (SR) approach introduced a discretized abstraction of time on which computations happen logically instantaneously. Contrary to SR, Logical Execution Time (LET) mandates to specify the actual logical duration a task has to fulfill. This allows a more efficient compilation, at the price of a lower expressiveness. Classical LET (i.e. as introduced in *Giotto/TDL*) sticks to uniform pseudo-physical time, i.e. based on one logical clock mapped to the real-time. In this paper, we introduce a new paradigm called *synchronous Logical Execution Time* (sLET) that builds upon both SR and LET paradigms. It keeps the idea of *logical durations* coming from the LET paradigm, while having logical instants based on logical clocks. This extends the expressivity of LET, as time is totally abstracted as sequences of events. The various schedulings provide physically timed versions that, while having distinct non-functional properties (in terms of performance mostly), remain mutually functionally equivalent (in the logical time realm). A particular instance, where computations are executed "in a single instant", and then time is advanced (as in classical event-driven simulation), can lead to a direct translation into synchronous formalisms (in our case *Esterel*). We started inquiring how this could open new ways of verification and analysis on PsyC programs.**

I. Introduction

The design of embedded control software calls for stringent real-time constraints due to their permanent interaction with the physical environment. Such real-time systems are usually safety-critical because of the context in which they are used (avionic, automotive ...). Therefore, their designs should be verified with the highest possible level of confidence. Various formalisms based on the concept of *Multiform Logical Time* have been introduced to abstract real-time durations which are usually not known at the design phase, as they might be target-dependent. Then, specific analysis, called schedulability analysis (or time safety analysis) ensure that physical computations satisfy their logical time constraints.

Among those formalisms, the *Logical Execution Time* (LET) paradigm [1] brings a compromise between the strong expressiveness of the synchronous-reactive approach (SR) [2] and the efficiency of traditional task scheduling. Contrary to the SR model, computation takes time and is bounded by a fixed logical duration usually known before the computation happens. However, in SR model, because computations are considered instantaneous, time can be refined using multiple logical clocks. LET is usually based on a unique clock representing the progress of time. Consider the following temporal interval where the bounds are described by `sync`:

```
sync 1 ms; f(); sync 1 s;
```

Depending on the semantics of the language, the interval might last one second or up to the next second tick. The former semantics, usually used by LET based languages, would mean that both ms and s refer to the same clock (i.e. $1\,s$ being just a short for $1000\,ms$) while the latter semantics, corresponding to synchronous languages, mean that both ms and s are logical clocks related with an affine relation.

In industrial projects, while classical synchronous languages such as Lustre or Scade fit perfectly the need to describe the functional behavior of the system, Synchronous LET, which is implemented by the industrial language *PsyC* [3], actually focus on a different level of the system life-cycle, namely, the software integration level. This approach is quite classical (see [4], [5] and[6]) and divides the functional design of the system from its non-functional design, that is, how different functional components (either designed with synchronous languages or manually) are integrated. This multilayer approach is depicted in figure 1. This is, however, still a challenge today to verify that properties specified at system-level (i.e. in the specification) still holds at integration level.

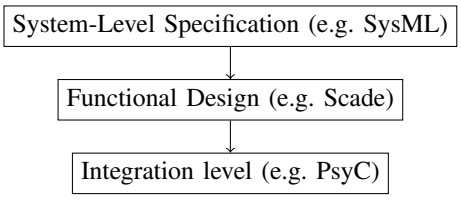


Fig. 1: Typical Software Life-Cycle of industrial real-time systems

In this paper, we introduce a new paradigm called *synchronous Logical Execution Time* (sLET) that builds upon both

the SR model and LET. It keeps the idea of *logical durations* coming from the LET paradigm while having logical instants based on logical clocks. This extends the expressivity of LET as time is totally abstracted as sequences of events. Thus, it inherits 1) the simple compilation and fine-grained schedulability analysis coming from LET and 2) the synchronous semantics that is well-suited for formal verification.

After an overview of existing Multiform Logical Time models, we introduce sLET as an extension of LET in section II. To illustrate sLET, we then give an overview of the formal semantics of a subset of the industrial language *PsyC* [3] in section III, developed by the company Krono-Safe, and then, we give translation rules to the synchronous language Esterel in section IV. While *PsyC* is compiled using techniques similar to the ones of LET, formal verification techniques coming from synchronous languages such as *Esterel* could then be re-used. Such approach is illustrated in section V where *PsyC* and Scade interact in a classical use-case showing a flight control system.

II. Multiform Logical Time Models

Synchronous languages introduce the notion of Multiform Logical Time, through logical clocks, so they could better be called Polychronous time. A logical clock counts the successive ticks of any relevant event. Still, their operational semantics expands the behaviors in terms of a grandmother clock, the discrete reaction step measuring the instant. On the other hand, initial LET formalisms rely on such a totally ordered discrete time, and introduce quantitative durations measured in it; although they may use pseudo-physical unit names (milliseconds, nanoseconds,...) for it, the compliance with actual physical implementation is only a wish (or a requirement), that will have to be checked later when the latter becomes available. We now extend on these notions, and the abstraction they allow for fast and human-legible logical design, cleanly split from the later physical implementation (and without back and further adjustements hopefully, a main purpose of these formalisms). The purpose is to position the sLET formalism as combining multiform logical time and the ability of user-defined logical clocks, together with quantitative durations in which a certain amount of computation can be spread in any instants of a given interval, providing it does not overflow it.

A. The Synchronous-Reactive Model (SR)

The Synchronous-Reactive model, implemented by synchronous languages [2], totally abstracts execution time to focus on logical instants, allowing both determinism and concurrency. For that, computations react simultaneously and instantaneously with the tick of a global common logical clock as shown in figure 2a, which can be further refined to give multiple logical clocks. The synchronous hypothesis, then, ensures that if every computation is bounded by the next reaction, then physical time can be safely ignored as shown in figure 2b. Combining synchrony and concurrency allows a very expressive formal model while keeping it very simple [2]. Hence, SR model is well adapted for formal verification.

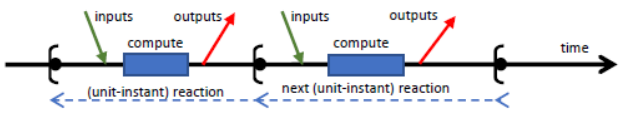
SR model has been introduced in languages such as *Esterel* [7] which is mainly imperative and *Lustre* [8] which is declarative. The compilation of synchronous languages is, however, quite complex. First, instantaneous communication makes the compilation for parallel platforms (i.e. multicore, distributed) quite hard and can produce causality issues. Secondly, execution time is usually limited to the reaction time of the system. Thus, applications with non negligeable execution time (i.e. also called the long period problem) can be rejected during the schedulability analysis. Nonetheless, languages based on the synchronous model such as *Prelude* [6], can adress this problem, but are usually limited to specific patterns such as a multi-periodic synchronous model.

B. The Logical Execution Time Model (LET)

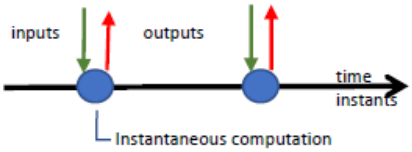
The Logical Execution Time paradigm abstracts physical time through a sequence of instants, similarly to the synchronous approach. However, contrary to the latter, computation takes time and is not considered instantaneous. For that, each computation must fit in a logical interval, called LET interval [1]. Furthermore, communications are only made on the boundaries of LET intervals. Inputs are read at their beginning and outputs are made visible to other tasks at their termination (see Figure 2c). As communication is only done on predefined instants, computations behave like they always take the same time, which is the LET interval duration (see Figure 2d). As such, the determinism property is ensured.

The LET paradigm has been initially introduced in the *Giotto* [1] language in which each task correspond to a periodic LET interval. The language can also express global modes that allows to change from a given set of task frequency to an other. *Timing Definition Language* (TDL) [9] extends *Giotto* via a decomposition in concurrent modules. Each of them defines a set of task and their frequency, similarly to *Giotto*, but a module can also change its local mode independently of other modules. *Giotto* has also been extended with events in the *xGiotto* language [10] through the mechanism of event scoping. In all these approaches, the bounds of LET intervals are either defined using chronometric pseudo-physical durations or events (i.e. in *xGiotto*). This paper proposes a more abstract and homogeneous model of time, that is, the use of multiple logical clocks.

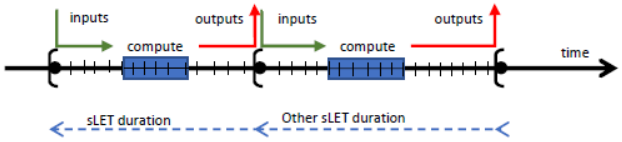
In a way, LET extends logical time with the concept of logical durations. This allows a precise schedulability analysis due to an increased execution time variability. In turn, this also makes compilation to parallel platforms easy and causality issues of the SR model are avoided. However, as LET forbids



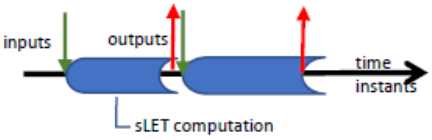
(a) Physical view of the Synchronous-Reactive model



(b) Logical view of the Synchronous-Reactive model



(c) Physical view of the (synchronous) Logical Execution Time model



(d) Logical view of the (synchronous) Logical Execution Time model

Fig. 2: Multiform Logical Time models: physical vs logical views

instantaneous communications, its theoretical expressivity is reduced compared to the SR model.

C. sLET as a synchronous extension of LET

Synchronous Logical Execution Time (sLET) extends the classical LET paradigm with the concept of logical clocks. As such, interval bounds can be triggered by clock ticks while in classical LET, interval duration is usually defined by a constant fixed duration. However, as in LET, communication can only happen on interval bounds, which ensure temporal determinism. Being closer to the SR model, sLET is actually fully compatible with the synchronous hypothesis, that is, computations can happen instantaneously on the activation date of the interval as long as outputs are delayed to the end of the interval. This model is actually a generalization of the *Psy* model introduced in the OASIS framework since the 90's [3].

A logical clock, in the sense of Lamport [11], abstracts time through a series of events called *clock ticks*. The sLET paradigm adds to LET a finite set of logical clocks $\mathbb{C}$ on which LET interval boundaries can be based. We will call *clock event* an event of the form $n \times c$ where $n \in \mathbb{N}^*$ and $c \in \mathbb{C}$. This event basically means that we have to wait n ticks of the clock c. An interval termination is then defined with a *clock event* which may also define the activation of a next interval. As an example, if an interval termination is defined with $n_t \times c_t$, then after its activation, it should wait for n_t ticks of the clock c_t. Note that if all intervals use the same clock, then clock events actually express fixed constant durations with respect to that clock which is similar to the classical LET model.

sLET keeps with the LET idea of imposing at an early phase of logico-functional design some fixed interval durations at which boundaries the I/O external behaviors will be supposed to occur. Then there is flexibility on when exactly computations are scheduled, as long as they remain inside these bounds. Next, we will introduce the PsyC languages foundations, bringing syntax to the sLET view. In particular, PsyC allows conditional durations, in alternative "if-then-else" behaviors. Note however that classical LET, as defined in *Giotto*, can actually be expressed easily with sLET, that is, using only one logical clock mapped to real-time. The converse is not true. Indeed, even considering that all logical clocks are derived using affine relation from a unique global clock, sLET interval might have different duration depending on the current global state.

III. The PsyC language: abstract syntax and formal semantics

A. Language description

In this section, we give a brief description of *PsyC*, a language developed by the company Krono-Safe, as an illustration of the sLET paradigm. This language is implemented in the ASTERIOS product which provides a set of tools to design safety-critical real-time software. Such applications can then be certified at the highest level of criticity for the avionic domain (DAL-A, DO-178C). ASTERIOS and *PsyC* are inherited from the OASIS and PharOS projects coming from the CEA. Initially, *PsyC* (for *Parallel SYnchronous*), in OASIS, was presented as a timed-triggered approach for safety-critical real-time software under a model called *Psy* [3]. This model is actually very close to LET, and synchronous LET generalize both of them. The modern version of *PsyC* is actually better characterized with synchronous LET due to the existence of multiple logical clocks.

A *PsyC* application is composed of a fixed set of tasks called *agents* that are formed of sLET intervals. The content of *agents* is composed of C code that is extended with special instructions like `advance` n `with` c which specifies the boundaries of sLET interval. Informally, its semantics is to advance the logical time of n ticks of a clock c. Such clocks have to be defined in the application as an affine relation with

respect to another clock. They are actually two kinds of logical clocks in *PsyC*, *sources*, which are defined externally, and *clocks*, which basically refine *sources*. In practice, most of the applications define only one *source* called *realtime* which is mapped on real time. The Figure 3 shows an example of an agent using multiple clocks and a conditional computation in the *PsyC* language. A possible timeline is shown in figure 4. Such pattern with multiple clocks allowed by sLET couldn't be expressed, as is, in the classical LET model.

```
source realtime;
clock c2 = 2 * realtime;
clock c3 = 3 * realtime;

agent Ag(uses realtime, starttime 1 with c2)
{
  body start /* infinite loop */
  {
    /* computation A */
    advance 1 with c3;
    if (/* condition */) {
      /* another computation B */
      advance 1 with c2;
    }
  }
}
```

Fig. 3: Example of a PsyC agent

Inter-agent communication is performed through a dedicated channel called *Temporal Variable*. Multiple agents can read a *temporal variable* but only one can write into it. According to sLET, inputs and outputs in a sLET interval can only be made on sLET interval bounds. Hence, data emitted by an agent is made visible (i.e. the temporal variable is updated) on the instant described by the next `advance` instruction. Futhermore, *Temporal Variables* also have another layer of sampling (defined with a clock) and values are read with the expression $\$[n]var$ which denotes the n^{th} last sampled value.

The Figure 5 describes the abstract syntax of a subset of *PsyC* which is necessary to introduce the formal semantics described in the next section. Only specific *PsyC* constructions are described and the syntax of C expressions is omitted for simplicity. Also, we add a specific statement skip that does nothing (i.e. in the C syntax, this could represented as a semi-colon). This will be helpful in the semantics to express that a statement doesn't have any work left to do. Futhermore, both the `starttime` and the source parameter of the agent are omitted because the former could be defined as an `advance` and the latter could be deduced from the clocks. However, we assume that an agent could only use clocks that are derived from a unique source. Nonetheless, different agents might use different sources.

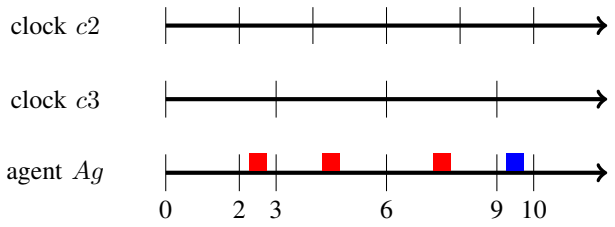


Fig. 4: Possible timeline of example in fig 3. A Red frame represent computation A while a blue frame represent computation B.

$$
\begin{array}{lcl}
\textit{application} & ::= & \textit{decl}^* \\
\textit{decl} & ::= & \mathsf{source}\ c \\
 & | & \mathsf{clock}\ c_1 = n_1 * c_2 + n_2 \\
 & | & \mathsf{temporal}\ id\ =\ value\ \mathsf{with}\ c \\
 & | & \textit{agent} \\
\textit{agent} & ::= & \mathsf{agent}\ id\ \textit{body}^* \\
\textit{body} & ::= & \mathsf{body}\ id\ \textit{stmt} \\
\textit{stmt} & ::= & id\ :=\ exp \\
 & | & \mathsf{skip} \\
 & | & \mathsf{advance}\ n\ \mathsf{with}\ c \\
 & | & \textit{stmt}_1\ ;\ \textit{stmt}_2 \\
 & | & \mathsf{if}\ (exp)\ \textit{stmt}_2\ \mathsf{else}\ \textit{stmt}_3 \\
 & | & \ldots
\end{array}
$$

Where $n, n_1 \in \mathbb{N}^*$, $n_2 \in \mathbb{N}$, v is the initial value of the temporal variable and c_i are clock identifiers.

Fig. 5: Abstract syntax of our *PsyC* subset

B. Formal Semantics

In this section, we introduce the formal semantics of a subset of the *PsyC* agents thanks to term rewriting. Basically, a term can be rewritten sucessively, with respect to a given set of formal rules, to form a symbolic execution that serves as an oracle. The rules are given using the Structural Operational Semantics (S.O.S.) approach introduced by Plotkin [12] and later adapted to synchronous languages [7]. The idea behind S.O.S. is to give the rewriting relation of a term with respect to the rewriting of its parts. This is usually defined through a set of inference rules. The behavioral semantics of Esterel extends S.O.S. through the use of an integer encoding the type of the transition, i.e. whether the transition takes time or is instantaneous [7] (i.e. fits entirely inside an instant reaction).

As described above, the semantics of *PsyC* is described in this paper using two relations, one that makes time progress, and one that is not explicitely timed, and should be considered as conceptually guaranteed to fit in the time interval closed by the next time-progress behavior. This vision can be considered as borrowed from discrete-event simulation models. While the semantics allows many later schedules for the internal behaviors before next time advance (a property exploited by the *PsyC* compiler to optimize their time allocations according to other criteria), one may also consider the (valid) interpretation where all computations are made in the initial instant, than

will allow the translation to synchronous languages of the next section.

Consider that the configuration of an agent (i.e. its state) can be described as a pair $\langle E, T\rangle$ where E is the assignements of its local variables and T the assignments of its displayed temporal variables (i.e. its outputs). Based on this, we define an instantaneous transition as following:

$$\langle E, T\rangle \vdash s \longrightarrow < E', T' > \vdash s'$$

This transition denotes that a statement s can be rewritten in s' instantaneously while the configuration $< E, T >$ is updated to $< E', T' >$ with respect to s. Based on this definition, we can define the semantics of the first basic statement, the assignation:

$$\langle E, T\rangle \vdash x := exp \longrightarrow \langle E', T\rangle \vdash \mathsf{skip} \qquad \text{(assign)}$$

where $E' = Update(E, v, exp)$.

The $assign$ rule evaluates its expression and updates its local variables accordingly using a function that we call $Update()$. This statement is logically instantaneous because it is executed at the beginning of a sLET interval to read the correct input values. Also, it is reduced to skip as there is no work left to do. The $advance$ rule is a little bit more complicated as this statement makes time progress. Recall however that clocks used in an agent by advance statements are all derivated from a unique source. Thus, we first instantaneously rewrite the $advance$ statement with respect to its parameters (i.e. number of ticks and clock) to a counter of source ticks that we define with the pseudo statement *Remains*. This pseudo statement takes an absolute duration in parameter (with respect to the agent source) that is computed with the function $Duration$ with respect to the $advance$ parameters, and the current source date.

$$\langle E, T\rangle \vdash \mathsf{advance}\ n\ \mathsf{with}\ c \longrightarrow \langle E, T\rangle \vdash \textit{Remains}(N) \qquad \text{(advance)}$$

where $N = Duration(n, c, date)$

The *Remains* pseudo statement defined above can then be used to make time progress. For that, we define a new transition that takes one time unit of the agent source (symbolized by the double arrow shape):

$$\langle E, T\rangle \vdash s \Longrightarrow < E', T' > \vdash s'$$

Remains is then defined with two different rules. The first one, *Remains-1*, decreases the counter when its greater than 1. When the counter is equal to 1, *Remains* is rewritten to skip as there is no work left to do and temporal variables are updated. As other agents can only read the latter (and not the agent local environment), this actually means that outputs can only be made visible at the end of a sLET interval, which is actually the semantics of the sLET paradigm.

$$\langle E, T\rangle \vdash \textit{Remains}(N) \Longrightarrow \langle E, T\rangle \vdash \textit{Remains}(N-1) \qquad \text{(Remains-1)}$$

if $N > 1$

$$\langle E, T\rangle \vdash \textit{Remains}(1) \Longrightarrow \langle E, T'\rangle \vdash \mathsf{skip} \qquad \text{(Remains-2)}$$

where $T' = UpdateOutputs(T, E)$.

The rules defined above can be composed quite easily. First, when there are executed in sequence, multiple instantaneous transitions can be represented with one big instantaneous transition and when they are followed by a non instantaneous transition, they can be represented with a big non instantaneous transition. In other words, the semantics of an agent can be represented with only big non instantaneous temporal transitions, which is interesting to show the expected temporal behavior of the agent. Second, the rules defined above only show simple statements, but actually, this can be extended very easily to control flow statement (e.g. condition statement) given that the type of transition is propagated correctly.

To illustrate a little bit the semantics above, we show a very basic example of a sLET interval with an output variable called tv. Let's consider the following PsyC statements: $tv := 2$; advance 2 with $c2$ Let's assume that $c2$ is a strictly periodic clock with period 2 and offset 0, and the current date of the agent is odd. Then, with respect to the semantics rules, we have the following execution:

$$\begin{aligned}
&\langle tv =?, tv =?\rangle \vdash tv := 2 ;\ \mathsf{advance}\ 2\ \mathsf{with}\ c2 \\
&\longrightarrow \langle tv = 2, tv =?\rangle \vdash \mathsf{advance}\ 2\ \mathsf{with}\ c2 \\
&\longrightarrow \langle tv = 2, tv =?\rangle \vdash \textit{Remains}(3) \\
&\Longrightarrow \langle tv = 2, tv =?\rangle \vdash \textit{Remains}(2) \\
&\Longrightarrow \langle tv = 2, tv =?\rangle \vdash \textit{Remains}(1) \\
&\Longrightarrow \langle tv = 2, tv = 2\rangle \vdash \mathsf{skip}
\end{aligned}$$

This execution shows that the communication is correctly updated at the end of the interval and its duration is actually 3 as it depends on the current date (e.g. if the date was even, then the interval should have lasted 4 ticks). Also, one can notice that such pattern is not possible in classical LET languages (e.g. *Giotto*, *TDL*) as they usually can only represent duration on a single clock.

Construction	PsyC syntax	Esterel Translation
Clock	$clock\ c = n_1 * c_p + n_2$	$await\ immediate\ n_2;$ $loop$ $\quad emit\ c$ $each\ n_1\ c_p$
Temporal Variable	$temporal\ tvar\ =\ init\ with\ c$	$signal\ tvar = init\ in$ $\quad run\ Sampler[tvar/In, tvar_0/Out, c/Clk]\ \|\|$ $\quad ...$ $endsignal$
Local Variable	$int\ var\ = init;$	$var\ private\_var\ :=\ init\ in\ ...$
Assignation/expression	$tvar = exp;$	$private\_tvar\ :=\ exp;$
Temporal variable reading	$\$[0]tvar$	$tvar_0$
Condition	$if\ (exp)\ s1\ else\ s2$	$if\ (exp)\ s1\ else\ s2$
Advance	$advance\ n\ with\ c$	$await\ n\ c; emit\ tvar(private\_tvar)$
Body (cyclic case)	$body\ p$	$loop\ p\ end$
Agent composition	$agent\ ag1\ ...agent\ ag2\ ...$	$run\ ag1[...]\ \|\|\ run\ ag2[...]\ \|\|\ ...$

Fig. 6: Some Esterel translation patterns (var is a private variable and $tvar$ is a (shared) temporal variable)

IV. Consequences

A. Synchronous semantics of sLET via Esterel translation

To show the direct relation between sLET and the SR model, we can encode the semantics of *PsyC* into a synchronous language like *Esterel*. We choose *Esterel* because its syntax is quite close to *PsyC*, that is, both languages are control-flow and imperative. Basically, the approach is to focus on the logical behavior of synchronous LET and to totally abstract actual executions, i.e. to consider them (logically) instantaneous. Thus, of course, this kind of simulation is not representative of the actual operational behavior where each computation takes time. Nonetheless, sLET (similarly to classical LET) allows multiple valid schedules that are mutually equivalent, that is, that form an equivalence class with respect to logical clock ticks. The synchronous semantics is then actually a particular instance where in a sLET interval:

1) all computations are executed during the first instant, thus inputs are read synchronously to the start of the interval;
2) time is advanced to the end of the interval;
3) outputs are produced synchronously to the end of the interval.

This abstraction should be sufficiant to express properties at the model level as actual computations should not impact temporal properties of a sLET design. While this approach focus on synchronous LET, as the latter is a generalization of the classical LET model, this approach could be adapted easily to other LET languages.

Esterel is an imperative synchronous language control-flow based. In particular, it allows imperative concurrency, that is, handling multiple instruction pointers at a same instant and preemption where a computation can be aborted when a given signal is present. The only communication means available in *Esterel* is through instantaneous signals which can have a status, *present* or *absent*, and a value. A signal which only has a status is said to be *pure*, otherwise, it is said to be *valued*. The instructions are basically divided into two categories similarly to *PsyC*:

- the instantaneous instructions, like $emit\ s$ which emits a given signal s;
- and temporal instructions, like $pause$ or $await\ s$ which respectively wait for the next instant and for the next instant where signal s is present.

For a full definition of the Esterel language, we invite the reader to consult [7].

In a *PsyC* application, different elements are actually composed in a synchronous parallel fashion. Assuming that *PsyC* clock ticks are represented as *Esterel* pure signals (i.e. either the clock ticks or not), we have to represent clock generation, temporal variable sampling and agent behavior. As explained at the beginning of section III, a *PsyC source* clock is actually an external logical clock, and is thus represented as an *Esterel* pure signal input. Clock generation basically refines a clock tick, i.e. a pure signal, based on an affine relation. A temporal variable is represented as a persistent valued signal (i.e. which is always present) and is sampled on the signal corresponding to its clock. And finally, an agent is almost translated as is. The control-flow is translated in its equivalent form in *Esterel* and basic *body* patterns can be translated either to sequence or to infinite loop depending on the next body to be executed. We do not cover advanced *body* patterns in this paper for simplicity but this could be adapted with some more advanced *Esterel* control-flow patterns. All variables of an agent, either local to the agent or temporal variables, are translated to *Esterel*

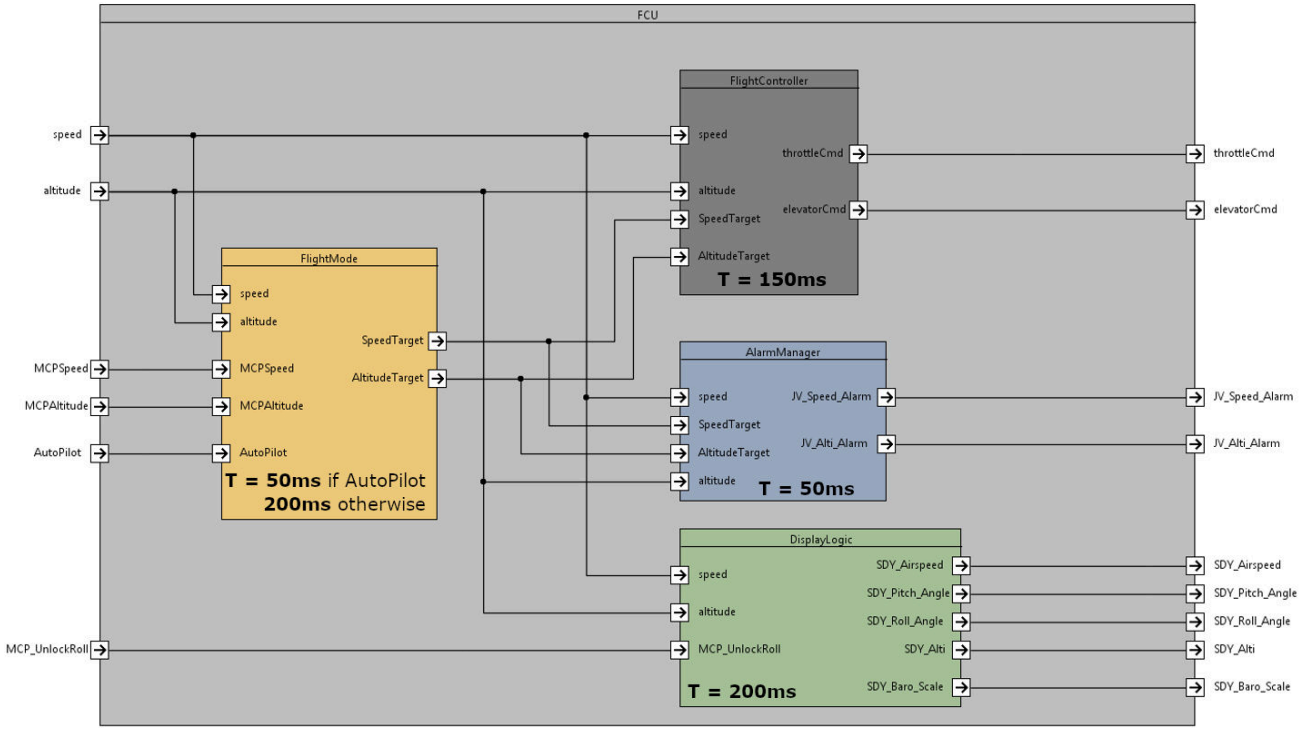


Fig. 7: The Flight Control System use-case

variables. The *advance* statement is translated to the *await* statement followed immediately by an update of the temporal variables, that is, the valued signal of each temporal variable in output of the agent is updated with its local variable value. This is actually the main difference from the synchronous model, communication cannot be updated instantaneously. All the translation patterns described above are sketched in the table 6.

B. Synchronous Observers and Formal Verification

The main interesting consequence of the *Esterel* translation is to apply techniques and tools coming from the synchronous community. Among them, formal verification has been successfully applied on the synchronous approach due to two main things: 1) the synchronous composition semantics, the state space is usually quite reduced comparing to asynchronous approaches and 2) symbolic model-checking allows to scale verification techniques even more, using BDD or SMT techniques. Futhermore, in synchronous languages, properties that are usually modeled using temporal logics, can also be modeled directly in the language using the so-called synchronous observer approach [13]. Observers only observe the state of the application and defines which state is acceptable or not. Classicaly, a signal *Error* is emitted if an error is detected and verification is reduced to a reachability analysis on this signal.

In *PsyC*, the typical kind of property that we might want to check applies on the different temporal variables rhythms. Typically, due to the sampled buffered semantics of temporal variable, if the producer and the consumer are not at the same rhythm, produced values can be lost or consulted values might be duplicated. While the second scenario is usually not a problem, the first one might be one. Of course, this kind of property is quite simple, but this opens the possibility to more evolved properties like freshness constraints of temporal variables or end-to-end latencies.

V. Use-Case

A. Description

As highlighted in the introduction, languages based on LET, such as *PsyC*, focus on the architecture description at integration level of real-time systems. In other words, they allow to express how multiple functional components connect and what is their real-time behavior. In particular, it is possible to express multi-rate behavior. Typically, in industrial applications, a synchronous language such as Scade, is used to design the functional components of the system and a language like *PsyC* can be used to specify the real-time software integration of these components.

To reflect this approach, the following use-case is based on an example found in the Scade Suite software. Consider a basic

Flight Control Software in which you expect the following behavior:

- given an altitude and a speed command, a flight controller implements control laws for altitude and speed with respect to altitude and speed sensors;
- depending on the flight mode, i.e. AutoPilot or Manual, the commands given to the flight controller is either the input commands or the commands of the previous instant;
- regulation alarm is raised if the altitude sensor (respectively the speed sensor) is too far from the altitude command (respectively the speed command).

The system architecture is shown in the figure 7. Basically, each component correspond to a function detailed above. Each functional component can either be implemented manually or with a *Scade* sheet. To illustrate a little bit more this approach we show in figure 8 the *Scade* sheet corresponding to the mode handling. The mode handling is modeled with a finite state machine with only two states, the *AutoPilot* state and the *Manual* state corresponding to the *AutoPilot* mode and *Manual* mode described at the beginning of the section. The control laws are based on PID controllers but we will not dive into the details as it is beyond the scope of this paper.

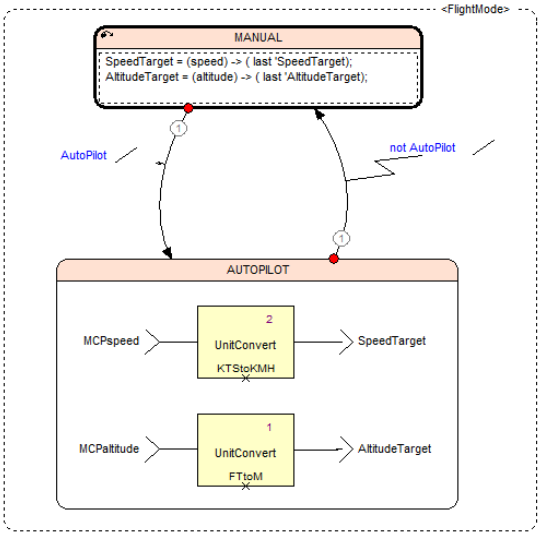


Fig. 8: Mode Controller of FCS

For simulation purpose, the example also contains a plane simulation that takes throttle and elevation commands (from the controller), simulates the actuator response as well as the aerodynamics behavior and yield updated speed and altitude sensors values (i.e. to the controller). Running such a simulation in *Scade* typically ignores real-time constraints and every components run at the same rate. While this is totally justified for simulation, it's is not necessarily the case for compilation. Indeed, in practice, different components might have different real-time behaviors, depending on available resource, as well as different mapping constraints (typically for multicore target). Thus, in the next section we will show how *PsyC* can be used to integrate seamlessly multi-rate functional components.

B. Architecture Description in PsyC

We choose a very simple mapping where each components described above are directly mapped to an individual agent. Futhermore, each communication signal is mapped to an individual temporal variable. Of course, while only one agent can write a temporal variable, multiple agents can read it. Thus, a temporal variable is not duplicated for each reader component. The temporal behavior of a component in an agent is the following:

- The reset function of the component generated from Scade is executed once with a given phase constraint (not specified here) and;
- The cycle function of the component generated from Scade is executed indefinitely *after* the reset function, on a given rate.

As an illustration, the code in 9 describes the temporal behavior of the controller component. The inputs and outputs of the cycle function are transfered with the help of global variables which is the usual calling convention with code generated from Scade.

```
agent FlightController(starttime 1 with clk_starttime)
{
  body start {
    next cycle;
    FlightController_reset_FlightControl();

    advance 1 with clk_10000_0 ;
  }
  body cycle /* infinite loop */ {
    altSensor = $[0]temporal_altitude;
    speedSensor = $[0]temporal_speed;
    speedSTarget= $[0]temporal_speedTarget ;
    altTarget= $[0]temporal_altitudeTarget ;

    FlightController_FlightControl();

    temporal_throttleCmd = throttleCmd;
    temporal_elevatorCmd = elevatorCmd;

    advance 1 with clk_150000_0 ;
  }
}
```

Fig. 9: Agent corresponding to the *FlightController* component

Chosing the right clocks is primary in the integration flow. First, the clocks should be sufficiently fast to ensure the stability of the different functional behaviors (typically control laws). Second, the clocks should be not too fast to ensure that the platform has enough resource to execute the system.

The expected clock periods are described in the figure 7. We consider that *FlightController* is a heavier task and hence, has a period three times bigger (e.g. $150ms$) than the other tasks that run with a period of $50ms$.

As explained in section III, *PsyC* also allows more advanced control flow patterns. As an example, the agent *FlightMode* may need to relax the deadline when the *AutoPilot* mode is disabled as specified in 7. This is illustrated in the listing 10. This kind of pattern doesn't have any impact on the functional behavior (i.e. commands are unchanged in manual mode), except during mode transition, and this allows to relax the cpu load.

```
agent FlightMode(starttime 1 with clk_starttime)
{
  body start {
    next cycle;
    FlightMode_reset_FlighControl();

    advance 1 with clk_10000_0;
  }
  body cycle /* infinite loop */ {
    /* From sensors */
    altitude = $[0]temporal_altitude;
    speed = $[0]temporal_speed;
    /* From user commands */
    AutoPilot = $[0]temporal_AutoPilot;
    MCPspeed = $[0]temporal_MCPspeed;
    MCPaltitude = $[0]temporal_MCPaltitude;

    FlightMode_FlightControl();

    temporal_speedTarget = SpeedTarget ;
    temporal_altitudeTarget = AltitudeTarget;

    if (AutoPilot) {
      advance 1 with clk_50000_0 ;
    } else {
      advance 1 with clk_200000_0 ;
    }
  }
}
```

Fig. 10: Agent corresponding to the *FlightMode* component with conditional deadline

C. Esterel Translation

The *PsyC* architecture given above allows efficiant compilation. However, as illustrated, the right clocks has to be used to respect the temporal behavior. For that, one can use the synchronous semantics of *PsyC*, that is, its translation to Esterel to analyze the application's behaviors. Of course, theorically, such analyze could be made in the Scade application simulating the multi-rate behavior. However, we think that such approach should be made at the integration level. Futhermore, the whole application might not be available in Scade, i.e. either coming from different teams or written manually.

The listing 11 shows the translation of the *Mode* agent. We remove the function calls for simplicity but not the communication mechanism. In the main loop, *altitude* is read instantaneously then, some computation is made and the *AltitudeTarget* is set to a private variable. Finally, after the `advance`, that is, the `await` here, this private variable is made visible through the `emit` statement. Special communication events are also added to show when communication is read and when communication is produced in this module. This will be needed later for verification.

```
module FlightMode:
[
  await clk_starttime;
  /* FlightMode_reset_FlightControl(); */
  await 1 clk_10000_0;

  loop
    emit FlightMode_Read;
    altitude := ?temporal_altitude_0;
    speed := ?temporal_speed_0;
    AutoPilot := temporal_AutoPilot_0;
    MCPspeed := temporal_MCPspeed_0;
    MCPaltitude := temporal_MCPaltitude_0;

    /* FlightMode_FlightControl(); */

    private_altitudeTarget := AltitudeTarget;
    private_speedTarget := SpeedTarget;

    if autoPilot then
      await 1 clk_50000_0;
    else
      await 1 clk_200000_0;
    emit temporal_altitudeTarget(
      private_altitudeTarget);
    emit temporal_speedTarget(
      private_speedTarget);
    emit FlightMode_Write;
  end loop
]
```

Fig. 11: Translation of *FlightMode* agent in Esterel

D. Synchronous Observers

The typical properties we might want to check with synchronous observers on such system are clock tick event ordering and clock tick event synchronizations. Typically, in *PsyC*, if the ticks of two tasks are alternating, and assuming temporal variable is at the speed of the producer, then, no value is lost. In a clock algebra such as CCSL, the predicate $c_1 \sim c_2$ states that the ticks of the logical clocks c_1 and c_2 alternate. Such constraints can be easily implemented in Esterel observers as shown in [14]. Thus, verification of such properties is then reduced to a reachability analysis on a signal representing an error state that we will call *Error* in this section.

In our example, let's consider that we want to ensure that all commands values are taken into account once by the *Alarm* agent. This functional chain is thus composed of the *FlightMode* agent, the *SpeedTarget* and the *AltitudeTarget* temporal variables, and the *Alarm* agent. Considering that the corresponding tick events have the same name, we have the following constraints to check to ensure that no values are lost:

$$\begin{aligned} \textit{FlightMode\_Write} &\sim \textit{SpeedTarget} \wedge \\ \textit{FlightMode\_Write} &\sim \textit{AltitudeTarget} \wedge \\ \textit{SpeedTarget} &\sim \textit{Alarm\_Read} \wedge \\ \textit{AltitudeTarget} &\sim \textit{Alarm\_Read} \end{aligned}$$

Based on [14], an *Esterel* observer for *Alternate* can be derivated. To fit our purpose, alternation strictness (whether two synchronous tick events could be synchronous or not) should be adapted to our communication mode. That is, the transition from event A to B in $A \sim B$ can be instantaneous but not the transition from B to A. The verification of the constraints described above can be written in instanciating the `Alternate` observer for each of the constraint with the help of parallel composition:

```
run Alternate[signal FlightMode_Write / A,
              SpeedTarget / B,
              Error/Error]; ||
run Alternate[signal SpeedTarget / A,
              Alarm_Read / B,
              Error/Error]; ||
run Alternate[signal FlightMode_Write / A,
              AltitudeTarget / B,
              Error/Error]; ||
run Alternate[signal AltitudeTarget / A,
              Alarm_Read / B,
              Error/Error];
```

In our scenario, this property is verified only when the *AutoPilot* mode is set as the *Alarm* agent might read multiple times the same values of the *FlightMode* agent when *AutoPilot* is disabled. The observer could be adapted easily to enforce alternation only when the mode is set. Also, note that this property does not enforce that all clocks should be synchronous (i.e. at the same rate). In fact, the agent *Alarm* might have some offset with *FlightMode* still without losing any value.

VI. Conclusion

This article presents a new paradigm called sLET that builds upon both the synchronous and the LET paradigm. Such unification allows to re-use formal analysis techniques coming from synchronous languages while keeping the simple compilation and efficient schedulability analysis of LET. sLET (as the classical LET paradigm) allows multiple valid schedulings that are mutually equivalent (in the logical time realm), as long as all of the computations fits in their sLET intervals (the *PsyC* compiler actually using one of these solutions also satisfying actual physical computation durations). The synchronous semantics is then actually a particular instance where computations are executed "in a single instant" and outputs are delayed. Consequently, this shows that one can use analysis based on the synchronous approach to reason on languages based on LET (e.g. *Giotto*, *xGiotto*, *TDL*) at the logical level; in particular, it unveils the possibility to re-use synchronous formal verification techniques for these languages. The sLET paradigm is implemented by the *PsyC* language which is part of the ASTERIOS framework, produced by KRONO-SAFE.

This work opens twofold perspectives: firstly, express more advanced properties, such as end-to-end latencies or data freshness, in a simpler way; secondly, synchronous verification techniques should be adapted more thoroughly to be efficient. In particular, (s)LET intervals are composed of instants that only make time progress and thus, can be optimized for verification. Finally, while our main focus is formal verification, other well known techniques can be adapted to languages based on (s)LET, such as model-in-the loop simulation, automatic test generation or test coverage.

References

[1] C. Kirsch and A. Sokolova, "The logical execution time paradigm," in *Advances in Real-Time Systems*, 2012.

[2] A. Benveniste and al, "The synchronous languages 12 years later," *Proceedings of the IEEE*, vol. 91, pp. 64 – 83, 2003.

[3] V. David and al, "Safety properties ensured by the oasis model for safety critical real-time systems," in *SAFECOMP*, 1998.

[4] P. Caspi, A. Curic, A. Maignan, C. Sofronis, S. Tripakis, and P. Niebert, "From simulink to scade/lustre to tta: a layered approach for distributed embedded applications," *ACM Sigplan Notices*, vol. 38, no. 7, pp. 153–162, 2003.

[5] S. Bliudze, X. Fornari, and M. Jan, "From model-based to real-time execution of safety-critical applications: Coupling scade with oasis," in *Embedded Real Time Software and Systems (ERTS2012)*, 2012.

[6] J. Forget and al, "A Multi-Periodic Synchronous Data-Flow Language," in *11th IEEE High Assurance Systems Engineering Symposium*, 2008.

[7] G. Berry, *The Constructive Semantics of Pure Esterel*, 1996.

[8] D. Pilaud, N. Halbwachs, and J. Plaice, "Lustre: A declarative language for programming synchronous systems," in *14th Annual ACM Symposium on Principles of Programming Languages*, vol. 178, 1987, p. 188.

[9] W. Pree and J. Templ, "Modeling with the timing definition language (tdl)," in *Automotive Software Workshop*. Springer, 2006, pp. 133–144.

[10] A. Ghosal, T. Henzinger, C. Kirsch, and M. Sanvido, "Event-driven programming with logical execution times," vol. 2993, 2004.

[11] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Commun. ACM*, vol. 21, no. 7, pp. 558–565, Jul. 1978.

[12] G. Plotkin, "A structural approach to operational semantics," *J. Log. Algebraic Methods Program.*, 2004.

[13] N. Halbwachs and P. Raymond, "Validation of synchronous reactive systems: From formal verification to automatic testing," in *5th Asian Computing Science Conference on Advances in Computing Science*, 1999.

[14] C. André, "Verification of clock constraints: CCSL Observers in Esterel," INRIA, Research Report, 2010. [Online]. Available: https://hal.inria.fr/inria-00458847

Session Th.2.A
Formal Methods & Certification

Thursday 2nd June

11:30

–

Amphithéâtre

A Bottom-Up Formal Verification Approach for Common Criteria Certification: Application to JavaCard Virtual Machine

Adel Djoudi, Martin Hána, Nikolai Kosmatov, Milan Kříženecký, Franck Ohayon
Thales
France & Czech Republic

Patricia Mouy, Arnaud Fontaine
ANSSI
France

David Féliot
CEA-Leti
France

I. Introduction

The quality and security of critical software have become nowadays a major concern. The Common Criteria (CC) for Information Technology Security Evaluation [1] provide an international standard for computer security certification. Its highest assurance levels EAL6–EAL7 require a formal Security Policy Model (SPM) and an associated mathematical proof of security properties (i.e. confidentiality, integrity). Thales recently conducted a formal verification of a JavaCard platform module [14] in a novel EAL6 certification project of a smart card product. This certification project was evaluated by CEA-Leti (an evaluation center, or ITSEF) with the supervision of ANSSI (the French national cybersecurity agency, and the French certification body).

Historically, since the verification of real-life code was not feasible for large industrial projects, the certification usually followed a top-down approach, where a separate abstract model was used to verify the specified security properties, and then refined to the code. A classical approach of applying formal verification on a JavaCard platform consists in building a high-level formal model of target sub-modules. The need to bridge the gap between the formal model and the implementation and to provide stronger guarantees for the real-life code was reported by experts [2].

In our work, we adopt a novel bottom-up methodology relying on verification of the real-life code of a JavaCard Virtual Machine using the Frama-C verification platform [3]. We expressed all specified features and properties as annotations in a formal specification language, called ACSL (ANSI C Specification Language), inserted directly in the source code. We annotated over 7,000 lines of C code in ACSL, and over 50,000 proof goals were generated and formally proved by the tool. An earlier paper [4] focused on technical and scalability issues of the proof without addressing the certification methodology. In this paper, we focus on methodology aspects: we describe this bottom-up approach, discuss its benefits and challenges and compare it to previous top-down approaches.

II. Common Criteria Certification Process

A. Overview of Common Criteria Evaluation

The international standard ISO/IEC 15408—which defines the Common Criteria (CC)—is an international agreement on security evaluation of IT products. It contains a Common Evaluation Methodology (CEM) describing the general evaluation process from EAL1 up to EAL5. SOG-IS[1] is the European mutual recognition agreement that was concluded in 2010 and involves ten countries. For EAL6 and EAL7, formal methods have to be used, but the CEM does not detail how to use them to demonstrate the respect of the security properties. For this reason, the CEM is completed by an additional interpretation by the National Certification Body. Within the SOG-IS, only three countries mutually emit and recognize certificates up to EAL7: France, Germany and Netherlands. Each of them defines its own interpretation for EAL6 and EAL7 (AIS-34 in Germany and Note 12 [5] in France). Recognition agreements beyond the EU (between more than 30 countries) are defined in the Common Criteria Recognition Arrangement (CCRA).

[1]See https://www.sogis.eu/

A Common Criteria evaluation is initiated by the owner of the product to evaluate. The product owner (generally, an industrial) establishes a contract with an approved evaluation center (ITSEF) and registers the evaluation with the Certification Body. At the end of the evaluation, an Evaluation Technical Report (ETR) is produced by the ITSEF to be reviewed by the Certification Body.

The national certification bodies recognized to deliver EAL6–EAL7 certificates do not prescribe the use of any particular formal method or tool [6]. Up to now, only methods relying on high-level models in B and Coq were recognized by ANSSI in the Common Criteria context. A guidance document about the use of formal methods was published by ANSSI [7]. A more recent guidance written by ANSSI and INRIA research teams was published but with a focus on Coq [8] with a dedicated paper [9], which details the rationale of the guidelines and requirements from ANSSI. The respect of these guidelines has to be verified during the evaluation process.

The adoption of a new formal method or tool requires a pilot evaluation. It implies a tripartite work between the developer, the ITSEF and the certification body, and additional effort to inspect formal assurance.

For an evaluation, the developer has to supply the following evidence:

- the source code of the Formal Security Model (SPM),
- an explanatory document presenting a description of the model, a complete list of the associated hypotheses (explicit ones but also implicit ones due to modeling choices), their justification and their consistency,
- explicit links between the model, the security target and the security properties (further discussed below),
- a clear justification of the level of confidence for the method and tools used, and
- all the necessary information to allow the evaluator to reproduce the proof(s).

B. Security Specification

A Common Criteria certification of a product helps the customer to determine whether the security of a product is sufficient to meet their needs and to ensure that the security properties are satisfied. For the product owner, the Common Criteria help to identify security issues, define security objectives, establish security requirements relying on a standardized catalog and then define a precise Target of Evaluation (TOE) that is usually only part of the entire IT product. The security specification plays an important role in the certification process.

The Common Criteria offer a catalog of Security Functional Requirements (SFRs) and Security Assurance Requirements (SARs) [1]. The SFRs define the TOE security characteristics. The SARs define confidence degree in the enforcement of the security objectives of the TOE. The assurance level is increased by increasing the scope, depth and rigor of the evaluation effort in six assurance classes: Security Target Evaluation (ASE), Development (ADV), Guidance (AGD), Life Cycle Support (ALC), Tests (ATE) and Vulnerability Assessment (AVA). At EAL6–EAL7 levels the ADV assurance requirement class mandates to build a formal security policy model (SPM) as the most rigorous way to identify and eliminate ambiguous, inconsistent, unenforceable or contradictory security policy elements [1]. For instance, the Common Criteria Action Element **ADV_SPM.1.1D** mandates the developer to identify the security policies that should be formally modeled. Note that the National Certification Body provides guidance for the interpretation of such statements [10] in order to satisfy the target security objectives. The identification of the security policy implies a list of SFRs to be formally modeled.

C. Application to JavaCard

JavaCard system is a well-known security-critical product. Many JavaCard products have been subject to Common Criteria evaluation. The Security Target Evaluation class (ASE) enforces the definition of a Security Target (ST) for an identified product (e.g. a particular JavaCard product). A Security Target is an implementation-dependent statement of security needs. It states what is to be evaluated before the evaluation is performed (and thus helps to understand after the evaluation what was actually evaluated). The Security Target may claim conformance (strict or demonstrable) to a **Protection Profile** for a generic TOE type (such as a JavaCard system) [11]. A protection profile provides a standardized statement of Security Policies to be tailored according to the defined scope of evaluation in Security Targets. Over the years, the JavaCard System Protection Profile has established as one of the most recognized smartcard industry reference and is typically mandated in tenders or requested explicitly by customers. For instance, the Open Configuration Protection profile of JavaCard systems defines the **Firewall Security policy/aspect** [11] as follows:

"**#.FIREWALL**: The Firewall shall ensure controlled sharing of class instances, and isolation of their data and code between CAP files (that is, controlled execution contexts) as well as between CAP files and the JCRE context. An applet shall not read, write, compare a piece of data belonging to an applet that is not in the same context, or execute one of the methods of an applet in another context without its authorization."

The protection profile also instantiates this security aspect with a security objective (**O.Firewall**) and provides a rationale for the list of SFRs to be satisfied in order to meet this objective. The Security Target of a product may then instantiate the Common Criteria requirement **ADV_SPM.1.1D** as follows:

"**ADV_SPM.1.1D**: The developer shall provide a formal security policy model for the **Firewall Security Policy**."

and provide a rationale (according to Common Criteria security components ASE_OBJ and ASE_REQ) for the (sub)set of SFRs that are formally modeled in order to meet this requirement. In the sequel of this document, we consider the two following (simplified) SFRs for illustration purposes:

SFR1 The Target of Evaluation Security Functions shall enforce the Firewall access control policy to provide restrictive default values for security attributes that are used to enforce the security policy. The objects' security attributes of the access control policy are created and initialized at the creation of the objects. Afterwards, these attributes are no longer mutable.

SFR2 The Target of Evaluation Security Functions shall enforce the following rule to determine if an operation among controlled objects is allowed: the **Currently Active Context** may freely perform any memory access operation upon any object whose **Lifetime** attribute has value "Persistent" **only if** the **object's owning Context** attribute has the same value as the **Currently Active Context**.

Further details about the interpretation of these SFRs are provided in the following sections. In particular, we illustrate the mapping of these SFRs to our formal model in Section V.

III. Bottom-Up Approach based on Deductive Verification

A. JavaCard Virtual Machine

JavaCard applets are compiled to bytecode, which is executable in the JavaCard Virtual Machine (JCVM). A binary file (called CAP file), loadable to the platform, encapsulates mainly the bytecode together with class definitions. It may contain several Java packages and applets. A unique context is associated to each CAP file during loading to the card. Prior to loading a CAP file, a ByteCode Verifier (BCV) is run off-card to perform a static analysis (type-level abstract interpretation) of the applets [12]. This ensures that the code does not attempt to perform ill-typed operations that may bypass security protections ensured by the JCVM. Indeed, the virtual machine ensures bytecode interpretation and offers higher-level, more secure abstractions of data than the hardware processor, such as object references instead of memory addresses. Memory access operations on the stack, heap or code area are performed by the JCVM for each interpreted bytecode.

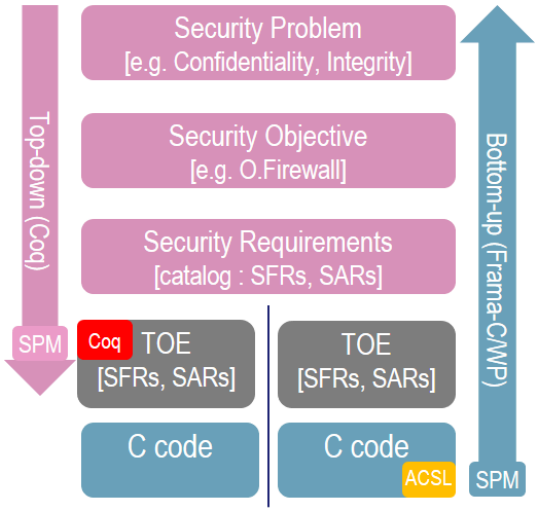


Fig. 1. Top-down/bottom-up approaches.

The JCVM firewall enforces runtime-protection of applet security properties against major security concerns: developer mistakes and design oversights allowing sensitive data "leakage" from the applet owning the data to another applet without explicit permissions [13]. In general, the firewall blocks access to the data in one CAP file to an applet in another CAP file (having different execution contexts), except some well-defined exceptions (such as global arrays, ArrayViews or class variables).

B. Target of Evaluation

The target of evaluation (TOE) of our project includes a set of Security Functions (TSF) that support the `O.Firewall` Security Objective enforcement. The implementation of these Security Functions in our project consists in a large subset of C functions of the JCVM. This subset features all possible functionalities of a single JCVM run that ensures the execution of any applet being selected. The JCVM specification [14] describes the data life cycle of an applet: transient deselect data is erased during owning application deselection, transient reset data is erased only when the smart card is reset, and persistent data is preserved anytime. Data and the associated life cycle of an undefined number of applets are carefully considered in our Security Policy Model (SPM) in order to ensure isolation properties. The SPM is deemed to comply with security functional requirements (SFRs) of the `O.Firewall` security objective defined in the JavaCard System Protection Profile [11], according to which "an applet shall not read, write, compare a piece of data belonging to an applet that is not in the same context, or execute one of the methods of an applet in another context without its authorization".

C. Formal Security Policy Model (SPM)

Before this work, ANSSI's requirements for formal methods in the context of Common Criteria evaluations were only successfully fulfilled using B and Coq methods. This project was part of a pilot evaluation due to the adoption of a new formal method based on Frama-C [3], a verification platform for C code. Frama-C uses ACSL (ANSI C Specification Language) [15], a formal specification language for C programs. It allows the user to specify annotations that express the expected program properties. The Frama-C/WP plugin can then be used to prove them for each function of the code: this technique is called *(modular) deductive verification*. In order to build the formal security policy model (SPM), we follow a bottom-up approach, in which the C code implementation is enriched with annotations instead of merely making a separate model and a formal representation of a set of SFRs being claimed (see Fig. 1). The security properties are proved to be enforced by the actual implementation design. Examples of ACSL annotations and security properties are given in the next section.

IV. Formal Specification of Contracts and Security Properties

A. Function Contracts

Deductive verification with Frama-C/WP requires that each C function be annotated with a *(function) contract* expressed in ACSL language. Such a contract contains *preconditions* (in **`requires`** clauses), which express properties of program variables that must be respected before the function is called, and *postconditions* (in **`ensures`** clauses), which express properties that must be ensured after the function terminates. A special kind of postconditions is expressed by **`assigns`** clauses, which give the list of variables that the function is allowed to modify. All other variables cannot be modified by the function. Each function should then be proved by Frama-C/WP to respect its contract.

```
1 /*@ requires \valid(p1) && \valid(p2) && \separated(p1,p2);
2     ensures \old(*p1) == *p2 && \old(*p2) == *p1;
3     assigns *p1, *p2; */
4 int swap(int *p1, int *p2){
5   int tmp = *p1;
6   *p1 = *p2;
7   *p2 = tmp;
8 }
```

Fig. 2. A C function `swap` which permutes the values referred to by two given pointers `p1` and `p2`, and its ACSL contract.

All functions in our Formal Security Policy Model are annotated by ACSL contracts (see [4] for detailed examples). For lack of space, we illustrate an ACSL contract in Fig. 2 for a very simple C function `swap` that swaps the values `*p1` and `*p2` referred by the two pointers given as function arguments. Line 1 expresses a precondition stating that input pointers `p1` and `p2` must refer to *valid* memory locations, that is, locations that can be safely accessed, and that these locations are *separared*, that is, the underlying bytes are disjoint. The validity is necessary here to ensure the absence of runtime errors, also known as *undefined behaviors*. The separation assumption is necessary to avoid overwriting some bytes of `*p2` when modifying `*p1`, and vice versa. Line 2 expresses a postcondition: the values after the function returns are swapped. Keyword **`\old`**`(e)` used in a postcondition allows

```
/*@ // === A security property: object headers remain intact ===
  predicate object_headers_intact{L1, L2} =
    \forall integer i, off; 0 <= i < \at(gNumObjs,L1) &&
      \at(gHeadStart[i],L1) <= off < \at(gHeadStart[i],L1) + 8 ==>
      \at(ObjHeader[off],L1) == \at(ObjHeader[off],L2); */
```

Fig. 3. Example of a security-related predicate expressing that object headers are not modified between program points `L1` and `L2`.

```
/*@ // === Metaproperties: persistent object data written/read only by the object owner ===
  meta \prop,\name(persi_objects_integrity),\targets(\ALL),\context(\writing),
    ( \forall integer i; 0 <= i < gNumObjs && !gIsTrans[i] && ObjHeader[gHeadStart[i] + 0] != JCC ==>
      \separated(\written,PersiData+(gDataStart[i]..gDataEnd[i])) );

  meta \prop,\name(persi_objects_confidentiality),\targets(\ALL),\context(\reading),
    ( \forall integer i; 0 <= i < gNumObjs && !gIsTrans[i] && ObjHeader[gHeadStart[i] + 0] != JCC ==>
      \separated(\read,PersiData+(gDataStart[i]..gDataEnd[i])) ); */
```

Fig. 4. Metaproperties for integrity and confidentiality of persistent object data.

one to refer to the value of expression `e` before the call. Finally, line 3 states that `*p1` and `*p2` are the only memory locations the function is allowed to modify.

B. Security Properties Expressed as Predicates

Some security properties (typically, for integrity) can be specified in ACSL as invariant properties maintained by relevant functions and directly proved by Frama-C/WP. We illustrate one of such properties, expressed by predicate `object_headers_intact` shown in Fig. 3, stating that object headers are not modified between program points `L1` and `L2`. For lack of space, we give here only the main ideas of this definition and refer the reader to [4] for detailed explanations. This predicate states that for any allocated object (represented by its index `i`) and for any offset `off` within the object header, the byte at offset `off` in the object header of object `i` has the same value in program points `L1` and `L2`. A typical usage of this predicate is to include the postcondition **`ensures`** `object_headers_intact{Pre, Post};` in the contract of every function. This postcondition ensures that the object headers are not modified by the function between program points `Pre` and `Post`, which refer to the states before and after the function call.

C. Security Properties Expressed as Metaproperties

Other properties (for confidentiality or some cases of integrity) are stated as *metaproperties* [16]. The main principle of a metaproperty is to state a global property for a specified set of target functions and a specified context. Two examples of key metaproperties are shown in Fig. 4. Again, we give here the main ideas of their definition, a more detailed presentation being available in [4]. Metaproperty `persi_objects_integrity` states that the data of a persistent object (represented by its index `i`) cannot be modified unless the current context (`JCC`) is the object owner (which is stored at offset 0 in the object header). Similarly, metaproperty `persi_objects_confidentiality` states that the data of a persistent object (represented by its index `i`) cannot be read unless the current context is the object owner.

Each metaproperty is instantiated by the Frama-C/MetAcsl plugin into assertions in relevant program points in all target functions. For example, the first metaproperty has the writing context, therefore the corresponding property must be checked each time when a memory location is modified. So an assertion is automatically added by the Frama-C/MetAcsl plugin at all those memory locations, where the metavariable `\written` is replaced by the written memory location. The proof of the resulting assertions ensures that the metaproperty is globally respected by the code.

The proof of real-life code in our project requires a careful combination of several ingredients (see Fig. 6): macros, companion ghost code, global preservation properties in addition to lemmas and proof scripts. Macros reduce redundancy in specifications and facilitate updates and maintenance. Ghost code is mainly used to describe the memory model and to offer an alternative encoding of low-level operations, amenable to automatic provers. This combination made it possible to efficiently reason about non-trivial code fragments involving bitwise operations without the use of external interactive tools (e.g. Coq) with a high level of automatic proof.

V. Traceability Between CC Requirements and Implementation

a) General Approach: As described in Section II, Security Objectives and related Security Functional Requirements are summarized in a Protection Profile describing a particular product type. In this project, the Security Target is related to the Protection Profile for JavaCard System [11]. To establish a correspondence between formal (SPM) and informal concepts (ST), the developer must establish and describe the links between them, as mandated by [10]. In fact, ANSSI and BSI (the German national certification authority) have driven the use of formal methods in Common Criteria evaluations with the publication of guidance for developers and evaluators (Note-12 [10] and AIS34 [17]). Formal analyses in CC context consist in giving a proof that the TOE Security Functions correctly implement the expected security objectives. Several "representations" of the TSFs are provided as shown in Fig. 5:

- SPM: the Security Policy Model contains only the mechanisms directly supporting Security Objectives enforcement,
- FSP: the Functional Specification of the implementation where low-level details are abstracted away,
- TDS: the TOE design or a simplified version of the implementation,
- IMP: the most concrete representation.

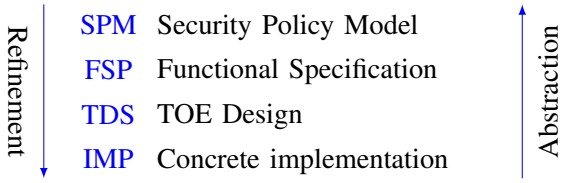


Fig. 5. Representations of the TSFs

The CC Assurance Development (ADV) components : Security Policy Model (ADV_SPM), Functional Specification (ADV_FSP), Target of evaluation Design (ADV_TDS), Implementation representation (ADV_IMP) provide a list of requirements to be fulfilled by each of the Security Function representations. Developers are also expected to establish a "formal equivalence" of these various representations of the TSF. The primary objective is to formally establish the correctness of the SPM w.r.t. security objectives. The rationale is to authorize reasoning at an abstract level (SPM) and to propagate the result toward the implementation (IMP).

The lack of proof of refinement until the implementation is the rub with the top-down strategy [18], [19], [20]. This is all the more unfortunate since most of the time the formal model is provided a posteriori, only for CC evaluation, and is not used to guide the design of the implementation.

Since our verification approach is based on formal properties written in ACSL annotations directly on the implementation, we link all informal concepts to formal ACSL annotations. Fig. 6 summarizes how the CC Assurance Development (ADV) components (Implementation representation (ADV_IMP), Target of evaluation

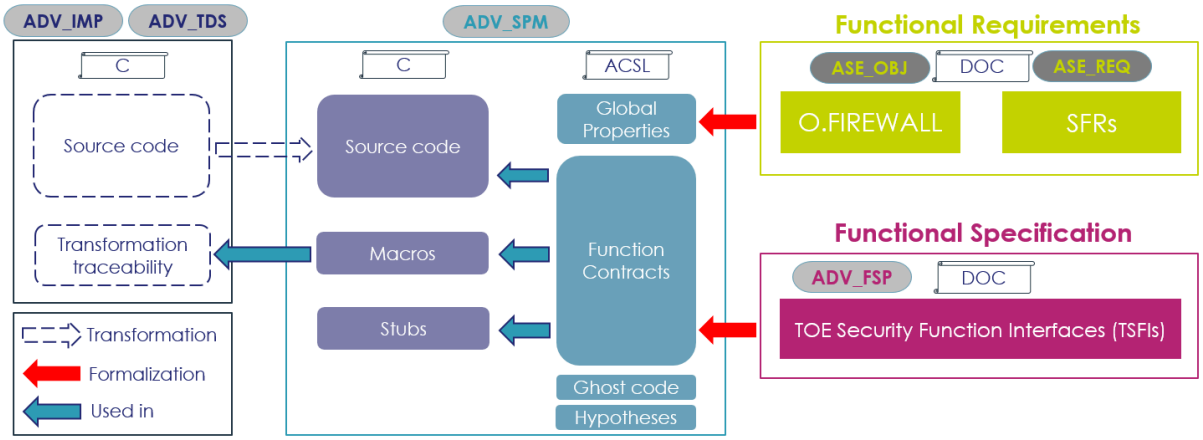


Fig. 6. Structure of the SPM with respect to CC requirements.

Design (ADV_TDS), Security Policy Model (ADV_SPM), Functional Specification (ADV_FSP)) are structured with respect to the actual product source code.

The proposed bottom-up approach intrinsically encompasses the refinement from the functional specification through the design to the implementation that is usually required in top-down methodologies.

b) Mapping Security Objectives and Requirements: The Security Objective defined in the Security Target (in our case, `O.FIREWALL`) is mapped to global ACSL annotations expressing the required isolation properties (either as ACSL predicates used as invariants, or as metaproperties for Frama-C/MetAcsl). A second mapping is then done for Security Functional Requirements (SFRs), describing the characteristics of the model (i.e. the contained functionality) for which the Security Objective is proved. Both mappings (Security Objectives and SFRs) are helpful to clearly articulate the security behavior chosen to be modeled, in other words, to clearly define the scope of the Security Policy Model (SPM). Security Functional Requirements are linked against some ACSL annotations in the SPM, it can be for example the contract of the firewall function or assignment specification of important variables decisive for firewall result (current context or context of objects). To make the mapping easier to evaluate, separate tables are provided to "translate" the Common Criteria terms of low granularity (e.g. Sharing, Currently Active Context) into their counterparts inside the Security Policy Model. It helps the evaluator to check the correctness of the mappings for particular SFRs based on these terms. For instance, SFRs introduced in II-C are mapped to their formalization in ACSL (depicted in Fig. 3 and 4) as follows:

SFR1 As sensitive security attributes are stored in object headers, global invariants are used to prove that the headers of objects are intact during the entire VM run. The predicate `object_headers_intact{Pre, Post}` used in a function contract, ensures that the content of object headers is the same at the end of the function as it was before executing the function. Thus, preserving this predicate throughout all relevant function calls during the VM run ensures the integrity of sensitive security attributes.

SFR2 Metaproperties `persi_objects_integrity` and `persi_objects_confidentiality` target integrity and confidentiality of object data respectively. In case the owner of a persistent object is different from the active context (`JCC`), any accessed memory location must be separated from the body of this object (see also Section IV-C).

VI. Specification Effort and Proof Results

JCVM C code		ACSL Annotations				
		User provided annotations			Generated by MetAcsl	Generated by RTE
# Functions	# Loc C	# Loc Ghost	#Loc Metaproperties	# Loc other ACSL annotations	# Loc ACSL	# Loc ACSL
381	7,014	162	350	35,480	396,603	2,290

Fig. 7. Specification effort for real-life code.

Applying deductive verification on large real-life industrial code requires a lot of care in order to avoid or at least mitigate scalability issues. In this project, we managed to ensure the scalability of our approach despite of the big size of the analyzed C code (7014 lines of C code split into 381 functions). As low-level operations are difficult to handle by automated provers, it was necessary to make abstraction of such low-level operations in a sound way. For that purpose, we introduced 162 lines of ghost code, carefully chosen to help automatic provers. As shown in Fig. 7, 35,480 lines of user-provided ACSL annotations were required in order to formally specify the security policy of the targeted virtual machine. However, we succeeded to reduce this effort up to 13,432 lines of ACSL annotations thanks to the usage of parameterized macros that gather redundant annotations. The effort is still considerable. 396,603 lines of ACSL annotations were automatically generated from 36 metaproperties (written in 350 lines of ACSL) only by Frama-C/MetAcsl. 2,290 lines of ACSL annotations were automatically generated by Frama-C/RTE.

Our proof scales reasonably well with an increasing number of proof goals. In particular, thanks to the translation of metaproperties into annotations that do not overload proof contexts, the metaproperty-based approach scales very well, despite a great number of generated annotations. Overall, it takes almost 3h30m to prove 52,198 proof goals.

99% of proof goals are automatically discharged by automatic provers. However, an important manual effort is required to maintain the proof of remaining proof goals when the code or the formal model are updated.

VII. Related Work

A classical approach of applying formal verification on a JavaCard platform relies on a high-level formal model [18], [19], [20]. Several case studies have adopted this approach. An executable formal semantics of the JavaCard Virtual Machine (JCVM) and the ByteCode Verifier (BCV) is proposed in [21] with 15,000 lines of specification in the Coq proof assistant. An operational semantics of a language modeling the JCVM behavior is proposed in [22], [23]. Authors of [24] describe a refinement-based approach, relying on the Coq proof assistant, to show that a native JavaCard API function fulfills its specification. In general, in such approaches, the traceability of formally proven properties may require an important effort to be justified because of the gap between the formal model and the source code [2]. Among tools designed and/or used for the purpose of providing formal guarantees about JavaCard platform security properties (but not in a Common Criteria context) we can list: Key [25], KRAKATOA [26] and Caduceus [27].

This work is also broadly related to other projects in which real-world software is verified. For instance, formal verification of the seL4 microkernel (comprising 8700 lines of C and 600 lines of assembly) was performed in a certification context [28]. Heitmeyer et al. [29] report on evidence for a Common Criteria evaluation of an embedded software system, which uses a separation kernel (of over 3,000 lines of C and assembly code). Although the separation kernel enforcing data separation was annotated with pre- and postconditions in the style of Hoare and Floyd, the machine-checked proof is not directly applied on the C code. A mechanized formal proof is performed on a Top Level Specification (TLS) of the separation-relevant behavior of the kernel. A correspondence between the annotated code and the TLS was established separately.

In general, while the usage of formal methods is, indeed, a costly validation technique, it is often seen as counterproductive to the current industrial development processes, even for having an advantage over competitors. Especially when the obtained certificates need to be renewed regularly, in order to try to cope with the dynamic landscape of a product's life-cycle [30]. We believe that our approach is well-suited to optimize, albeit still high, the investments to reach EAL7 certifications using formal methods. Further enhancements of our approach for automatic generation of Common Criteria documentation like in [32], [33], [34], may be a step further for a better integration in current industrial development processes.

VIII. Discussion and Lessons Learned

A. From the Developers' Point of View

An important drawback of top-down certification approaches, based on a high-level model (e.g. in Coq) is the traceability issue: the difficulty to relate the model with the real-life code. It can be complex to ensure that the model faithfully represents the behavior of the code. Another issue is the maintainability of the model for the developers: changes can be difficult to integrate and can require a very significant review of the whole proof.

a) Benefits: The presented bottom-up approach facilitates traceability since the SPM in that case is (a subset of) the real-life code, with the same structure (same functions, variables, data structures, etc.). Another benefit of the proposed approach is a better maintainability (in particular, in case of minor code updates or scope extensions) and a more straightforward extension from EAL6 to EAL7. Compared to a top-down approach, where significant model and proof changes are often required for more complex properties or a larger scope, integration of new properties or functions for an EAL7 certification in the presented approach can more significantly rely on the proof performed for the EAL6 level. Small design changes can easily be integrated in order to check if the security properties are affected.

The proposed bottom-up verification approach strongly benefits from automation, which is particularly important for a large industrial product. The link between the SPM and the real-life code in our project is explicit and can be automatically exploited by various tools. For example, they include syntactic code comparison and identification of possible differences—code transformations—between the SPM and the real-life code. It is important to efficiently identify and review such transformations (used in the verified code of the SPM e.g. to avoid some tool limitations or to realize some scope restrictions). Construction of a control-flow graph can help to identify the function hierarchy,

in particular, to automatically distinguish fully verified functions (with a contract and a body), functions that are included as stubs (with a contract and a declaration but without a body) and excluded functions.

Furthermore, most security-related ACSL annotations in our approach are generated automatically from a few high-level security properties (stated as *metaproperties* [16]) by the Frama-C/MetAcsl plugin and then verified by Frama-C/WP. In this project, over 22,000 assertions are automatically generated by Frama-C/MetAcsl from (only!) 36 manually written metaproperties. Similarly, properties on the absence of *runtime errors* (RTE, also known as *undefined behaviors*) are generated automatically by another plugin, Frama-C/RTE, before being verified by Frama-C/WP.

Another major advantage of the approach is that it strongly relies on automatic proof. In the presented project, about 99% of proof goals are proved automatically by the Alt-Ergo solver or by the Frama-C/WP plugin (and its internal simplification engine Qed). A huge effort would be required to prove them interactively (that is, basically, manually) in a proof assistant. Finally, the proposed approach is suitable for a continuous integration process and it is planned to use it in the future in a continuous integration environment.

b) Challenges and Points of Attention: Those benefits also come with challenges for developers. An EAL6 certification with a bottom-up approach takes a more significant effort, already going closer to the implementation than actually required by the Common Criteria. Source-code level formal verification can be sensitive to tool scalability issues. Indeed, the tool has to deal simultaneously with high-level program properties and low-level properties (such as absence of runtime errors, presence of casts and bit-level operations), that can lead to a large number of relatively complex properties. A specific expertise and a good understanding of the capacities of the chosen verification tools and automatic solvers are required for the developers. Advanced tools like Frama-C/WP offer interactive proof features (proof scripts) that help the developer to finish most complex proofs.

A significant effort of manual annotation of the code is another challenge. In the presented project, $\sim$12,000 lines of annotations were written manually for $\sim$7,000 lines of C code (i.e. a factor of 1.7). Some of them rely on carefully chosen macros to avoid annotation redundancy, without using it the developers would have to write $\sim$35,000 lines of annotations (i.e. a factor of 5 compared to the C code). Another challenge is an efficient and meaningful organization of annotations and global properties—sometimes not obvious in modular verification—that can have an impact both on the capacity to prove and to define the mapping (see Section V).

B. From the Evaluators' Point of View

a) Benefits:

1) the main benefit of the bottom-up approach for an evaluator is the immediate understanding of the formal entities, such as the modeling of the heap and the stack in a JCVM, because the program[2] has the same representation as the JCVM implementation, which has already been evaluated (ADV IMP);
2) as SFRs are directly represented in the program as formal annotations, the correspondence from the SFRs to the model can be easily understood by the evaluator, in particular to check that SFRs are modeled precisely enough to allow the verification of the security objectives;
3) only a single model needs to be reviewed because all the security and functional properties can be verified by the same model built on top of the implementation;
4) there is no refinement, no abstract level, no relation between multiple models to be evaluated;
5) the formal modeling of the subsystems is implicitly provided by the program;
6) apart from code transformations, the verified program has been written by a separate team (i.e. not the modeling team) which makes the relevance of the model easier to evaluate than when properties are verified on a dedicated model (in particular, a purely logical model);
7) the evaluator can directly check that well-known attack paths (e.g. type confusions) are not ignored by the model;
8) the bottom-up approach perfectly fits into the continuation of the evaluation process, unlike the top-down approach that involves a new (formal) design, which is more difficult to evaluate because of the traceability issue (as mentioned in Section VIII-A).

[2]The term "program" refers here to the C code part of the model, without the ACSL annotations and metaproperties.

b) Limitations and Points of Attention: The limitations of the bottom-up approach are caused by the code complexity which is directly transferred to the model making the whole proof too complex to be fully reviewed by the evaluator. However, deductive verification ensures that properties are correctly propagated between functions in the callgraph. If a property is verified in the contract of a caller, deductive verification ensures that the contracts of the callees are complete enough to verify the property at the caller level. Therefore, the evaluator can have confidence in the proof, as long as the properties are correctly defined, and the program is correctly modeled in terms of code transformations, ghost code, and hypotheses. Hence, the main points of attention are the following.

- *Properties:* The bottom-up approach may make the properties more complex to evaluate because they directly rely on the implementation. Metaproperties, which are defined globally and whose number is limited, can be reviewed more easily but their statement remains as complex as the low-level annotations.
- *Code transformations:* In some cases, the real-life code complexity cannot be fully supported by the existing tools, and requires manual transformations of the code. While some bugs in code transformations can be detected by the logical part of the model, some other bugs can lead to missing states and be hard to detect. An exhaustive manual review of the code remains difficult. Therefore, code transformations should rigorously follow a precise methodology even for the (apparently) simple cases.
- *Ghost code:* Companion ghost code increases the complexity of the model, but also helps the evaluator to understand how the proof is conducted. The evaluator should detect non-companion ghost code that is used to create new concepts in the model either to express properties that cannot be directly expressed with the concepts of the implementation, or to simplify the model. The evaluator needs to check that these new concepts are valid and consistent with the implementation.
- *Hypotheses:* Hypotheses can be introduced as preconditions of the entry point function, or contracts of stubs (whose code is not provided so their contracts are admitted). Stubs can include functions from the implementation removed for simplification, or new functions, specifically declared to introduce some local hypotheses at some points in the program. Local hypotheses make the understanding of the chain of reasoning and the detection of contradictory hypotheses more difficult. Additional tool-related hypotheses (like memory model assumptions in Frama-C/WP) also require specific attention of the evaluators.

C. Conclusion

The bottom-up approach brings many benefits to the certification process in terms of model understanding and confidence in verified properties. It helps to reduce the gap between the formally proved properties and the implementation, and should facilitate the step from EAL6 to EAL7 for the developers. Modern verification tools like Frama-C/WP and Frama-C/MetAcsl are capable to deal with real-life code after only a limited number of code modifications. The application of bottom-up approaches can require some adjustments in the existing terminology and certification guidelines (e.g. [10]) released by certification bodies. Indeed, historically, they were designed with top-down approaches in mind, and their application on the bottom-up approach requires some clarifications. The existing evaluation methodology has to be extended with additional tasks for a careful analysis of properties, code transformations, ghost code and hypotheses.

REFERENCES

[1] "Common criteria for information technology security evaluation. part 3: Security assurance components," CCMB-2017-04-003, Tech. Rep., 2017. [Online]. Available: https://www.commoncriteriaportal.org/files/ccfiles/CCPART3V3.1R5.pdf

[2] B. Beckert, D. Bruns, and S. Grebing, "Mind the gap: Formal verification and the common criteria (discussion paper)," in *Proc. of the 6th International Verification Workshop (VERIFY 2010)*, ser. EPiC Series in Computing, vol. 3. EasyChair, 2010, pp. 4–12.

[3] F. Kirchner, N. Kosmatov, V. Prevosto, J. Signoles, and B. Yakobowski, "Frama-C: A software analysis perspective," *Formal Asp. Comput.*, pp. 1–37, 2015.

[4] A. Djoudi, M. Hána, and N. Kosmatov, "Formal verification of a JavaCard virtual machine with Frama-C," in *the 24th Int. Symp. on Formal Methods (FM 2021)*, vol. 13047. Springer, 2021, pp. 427–444.

[5] SGDN, "Application note: Target evaluation's security policies formal modelling," 587/SGDN/DCSSI/SDR - NOTE/12.1, Tech. Rep., 2008. [Online]. Available: https://www.ssi.gouv.fr/uploads/2014/11/NOTE-12-Modelisation-formelle-SPM-EN.pdf

[6] C. Lavatelli and G. Tétu, "Formal models for high assurance: why and how," in *International Common Criteria Conference*, 2020. [Online]. Available: https://www.internetoftrust.com/wp-content/uploads/2021/02/ICCC-2020-SPM_v1.0_20201118.pdf

[7] SGDN, "Remarques relatives à l'emploi des méthodes formelles (déductives) en sécurité des systèmes d'information," SGDN/DCSSI/SDS/LTI[1], Tech. Rep., 2008. [Online]. Available: https://www.ssi.gouv.fr/uploads/2014/11/ssi_formelle.pdf

[8] ANSSI-INRIA, "Requirements on the use of Coq in the context of common criteria evaluations," SGDN/ANSSI/SDE/DST/LSL, Tech. Rep., 2020. [Online]. Available: https://www.ssi.gouv.fr/uploads/2014/11/anssi-requirements-on-the-use-of-coq-in-the-context-of-common-criteria-evaluations-v1.0-en.pdf

[9] Y. Bertot, M. Dénès, V. Laporte, A. Fontaine, and T. Letan, "The use of Coq for common criteria evaluations," *The Sixth International Workshop on Coq for Programming Languages (CoqPL 2020), part of POPL 2020*, 2020.

[10] SGDN, "Note d'application : Modélisation formelle des politiques de sécurité d'une cible d'évaluation," 587/SGDN/DCSSI/SDR - NOTE/12.1, Tech. Rep., 2008. [Online]. Available: https://www.ssi.gouv.fr/uploads/2014/11/NOTE-12-modelisation-formelle.pdf

[11] Oracle, "Java Card system – open configuration protection profile, version 3.1," Oracle, Tech. Rep., 2020. [Online]. Available: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Zertifizierung/Reporte/ReportePP/pp0099V2b_pdf.pdf;jsessionid=6C3F5A7FB5FA0D928A1C310C1C0EF1CE.internet462?__blob=publicationFile&v=1

[12] X. Leroy, "Bytecode verification on Java smart cards," *Software: Practice and Experience*, vol. 32, no. 4, pp. 319–340, apr 2002.

[13] Oracle, "Java Card Platform: Runtime Environment Specification, Classic Edition, Version 3.1," Oracle, Oracle, Tech. Rep., feb 2021. [Online]. Available: https://docs.oracle.com/javacard/3.1/related-docs/JCCRE/JCCRE.pdf

[14] ——, "Java Card Platform: Virtual Machine Specification, Classic Edition, Version 3.1," Oracle, Oracle, Tech. Rep., feb 2021. [Online]. Available: https://docs.oracle.com/javacard/3.1/related-docs/JCVMS/JCVMS.pdf

[15] P. Baudin, P. Cuoq, J.-C. Filliâtre, C. Marché, B. Monate, Y. Moy, and V. Prevosto, *ACSL: ANSI/ISO C Specification Language*, 2021. [Online]. Available: https://www.frama-c.com/download/acsl.pdf

[16] V. Robles, N. Kosmatov, V. Prevosto, L. Rilling, and P. Le Gall, "Methodology for specification and verification of high-level properties with MetAcsl," in *9th IEEE/ACM International Conference on Formal Methods in Software Engineering (FormaliSE 2021)*. IEEE, 2021, to appear.

[17] BSI, "Application notes and interpretation of the scheme (AIS), AIS 34, version 3," Bundesamt für Sicherheit in der Informationstechnik (BSI), Bundesamt für Sicherheit in der Informationstechnik (BSI), Tech. Rep., 2009. [Online]. Available: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Zertifizierung/Interpretationen/AIS_34_pdf.html

[18] B. Chetali and Q.-H. Nguyen, "About the world-first smart card certificate with EAL7 formal assurances," *The 9th International Common Criteria Conference*, 2008.

[19] ——, "Industrial use of formal methods for a high- level security evaluation," in *Proc. of the 15th International symposium on Formal Methods (FM 2008)*, 2008.

[20] D. Bolignano, "Formal proof of a secure OS full trusted computing base (invited paper)," in *Proc. of the 2nd International Workshop on Enabling Trust through OS Proof and Beyond (Entropy 2019), co-located the 4th IEEE European Symposium on Security and Privacy (EuroS&P 2019)*, 2019.

[21] G. Barthe, G. Dufay, L. Jakubiec, B. P. Serpette, and S. M. de Sousa, "A formal executable semantics of the JavaCard platform," in *10th European Symposium on Programming on Programming Languages and Systems, (ESOP 2001), Held as Part of the Joint European Conferences on Theory and Practice of Software (ETAPS 2001)*, ser. LNCS, vol. 2028. Springer, 2001, pp. 302–319.

[22] I. A. Siveroni, "Operational semantics of the Java Card Virtual Machine," *The Journal of Logic and Algebraic Programming*, vol. 58, no. 1-2, pp. 3–25, jan 2004.

[23] M. Éluard, T. Jensen, and E. Denne, "An operational semantics of the Java Card firewall," in *Smart Card Programming and Security, International Conference on Research in Smart Cards (E-smart 2001)*, ser. LNCS. Springer, 2001, vol. 2140, pp. 95–110.

[24] Q.-H. Nguyen and B. Chetali, "Certifying native java card API by formal refinement," in *The 7th IFIP WG 8.8/11.2 International Conference on Smart Card Research and Advanced Applications (CARDIS 2006)*, ser. LNCS. Springer, 2006, vol. 3928, pp. 313–328.

[25] W. Mostowski, "Fully verified Java Card API reference implementation," in *Proceedings of the 4th Interenational Verification Workshop in connection with CADE-21. CEUR Workshop Proceedings, Vol-259*, 2007.

[26] C. Marché, C. Paulin-Mohring, and X. Urbain, "The KRAKATOA tool for certification of Java/JavaCard programs annotated in JML," *The Journal of Logic and Algebraic Programming*, vol. 58, no. 1-2, pp. 89–106, jan 2004.

[27] J. Andronick, B. Chetali, and C. Paulin-Mohring, "Formal verification of security properties of smart card embedded source code," in *International Symposium of Formal Methods (FM 2005)*, ser. LNCS, vol. 3582. Springer, 2005, pp. 302–317.

[28] G. Klein, J. Andronick, K. Elphinstone, G. Heiser, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch, and S. Winwood, "seL4," *Communications of the ACM*, vol. 53, no. 6, pp. 107–115, jun 2010. [Online]. Available: https://dl.acm.org/doi/10.1145/1743546.1743574http://web1.cs.columbia.edu/$\sim$junfeng/09fa-e6998/papers/sel4.pdf

[29] C. L. Heitmeyer, M. Archer, E. I. Leonard, and J. McLean, "Formal specification and verification of data separation in a separation kernel for an embedded system," in *Proc. of the 13th ACM conference on Computer and communications security (CCS 2006)*. ACM Press, 2006, pp. 346–355.

[30] S. P. Kaluvuri, M. Bezzi, and Y. Roudier, "A quantitative analysis of Common Criteria certification practice," in *TrustBus 2014: Trust, Privacy, and Security in Digital Business*, 2014, pp. 132–143. [Online]. Available: https://www.eurecom.fr/fr/publication/4438/download/rs-publi-4438.pdf

[31] "The common criteria - certified products list - statistics." [Online]. Available: https://www.commoncriteriaportal.org/products/stats/

[32] P. Heck, M. Klabbers, and M. van Eekelen, "A software product certification model," *Software Quality Journal*, vol. 18, no. 1, pp. 37–55, 2010. [Online]. Available: https://link.springer.com/article/10.1007/s11219-009-9080-0

[33] F. Tuong and B. Wolff, "Deeply integrating C11 code support into Isabelle/PIDE," *Electronic Proceedings in Theoretical Computer Science*, vol. 310, pp. 13–28, dec 2019. [Online]. Available: https://arxiv.org/pdf/1912.10630.pdf

[34] S. Bezzecchi, P. Crisafulli, C. Pichot, and B. Wolff, "Making agile development processes fit for V-style certification procedures," *arXiv preprint arXiv:1905.06604*, may 2019. [Online]. Available: http://arxiv.org/abs/1905.06604

Obtaining DO-178C Certification Credits by Static Program Analysis

Daniel Kästner, Markus Pister, Christian Ferdinand
AbsInt Angewandte Informatik GmbH. Science Park 1, D-66123 Saarbrücken, Germany

Abstract

Static analysis has evolved to be a standard method in the software development and verification process. Its formal method, Abstract Interpretation, is one of verification methods covered by the Formal Methods Supplement DO-333 of the DO-178C standard. Static program analysis can contribute to numerous verification goals of DO-178C at various stages of the development process. The main focus of static analysis methods are non-functional software quality hazards, e.g., violations of coding guidelines, violations of software architecture constraints, violations of resource bounds such as stack overflows and real-time deadlines, runtime errors, and data races. This article gives a brief overview of abstract interpretation and its applications to detect different classes of safety hazards. We will review the requirements of DO-178C/DO-333, from High-Level Requirements to requirements for verification of Executable Object Code, and pinpoint aspects that can be covered by static analysis methods. The article concludes with illustrating the relevant requirements for DO-330-compliant tool qualification of static analysis tools.

Keywords: DO-178C, DO-330, DO-333, certification, static analysis, abstract interpretation, tool qualification

1 Introduction

Some years ago, static analysis meant manual review of programs. Nowadays, automatic static analysis tools have been established in modern software development processes as they offer a tremendous increase in productivity by automatically checking the code under a wide range of criteria. Here, the term *static analysis* is used to describe a variety of program analysis techniques with the common property that the results are only based on the software structure. No execution of the program under analysis is needed. Static analysis can be applied to any kind of program representation, from the model level or the source code level to the executable object code level.

An important distinction of static analysis methods is the complexity of the program properties they aim at determining and the level of rigor at which they operate. In the simplest form, static analysis is focused on the program syntax: purely syntactical methods can be applied to check syntactical coding rules as contained in coding guidelines, such as MISRA C [39], SEI CERT C [44], or the Common Weakness Enumeration (CWE) [46]. They aim at a programming style that improves clarity and reduces the risk of introducing bugs. Compliance checking by static analysis tools has become common practice.

Syntactic rules play an important part in coding standards as they are easy to take into acccount while implementing code, and easy to check. However, ultimately, the objective is to prevent code defects which means that semantical properties have to be considered. To that end semantics-based static analyzers are needed which focus on the program semantics and compute invariants about variable values, pointer targets, etc. This is also relevant for coding standard compliance checking, since all commonly used coding guidelines also include semantical rules.

Depending on the level of rigor, semantics-based methods can be grouped into unsound and sound approaches, the essential difference being that when a sound method reports the property under analysis – such as freedom of runtime errors – as satisfied, this is guaranteed to be true. Abstract interpretation is a formal method for sound semantics-based static program analysis [9]. It supports formal correctness proofs: it can be proved that an analysis will terminate and that it is sound in the sense that it computes an over-approximation of the concrete program semantics. Abstract interpretation always provides full data and control coverage.

As of today, abstract interpretation-based static analyzers are most widely used to determine non-functional software quality properties [20, 17]. On the one hand that includes source code properties, such as compliance to coding guidelines, compliance to software architectural requirements, as well as absence of runtime errors and data races [24]. On the other hand also low-level code properties are covered such as absence of stack overflows and violation of timing constraints [21, 28].

Violations of non-functional software quality requirements often either directly represent safety hazards and cybersecurity vulnerabilities in safety- or security-relevant code, or they can indirectly trigger them. Hence they are invariably addressed by verification obligations in current safety and security norms, such as DO-178C [40], IEC-61508 [13], ISO-26262 [14], and EN-50128 [8].

In this article we will focus on the DO-178C standard [40], its formal method supplement DO-333 [41], and the DO-330 [42] which details the tool qualification requirements to be taken into account. We will review the software requirements and verification goals that are amenable to static analyses and hence identify the certification credits that can be obtained by applying different static analysis methods at different levels of the development process. Sec. 2 walks through the DO-178C norm and highlights the relevant software requirements and verification objectives. Sec. 3 illustrates the methodology of static program analysis and presents the fundamentals of its application

to compute various non-functional software properties. Sec. 4 then summarizes the verification goals supported by the various applications of static analysis. A brief overview of the tool qualification requirements relevant to static program analysis tools is given in Sec. 5, and Sec. 6 concludes.

2 DO-178C / DO-333

The DO-178C [40], published in December 2011, is a revision of DO-178B to take progress in software development and verification technologies into account. In general, the DO-178C aims at providing "guidance for determining, in a consistent manner and with an acceptable level of confidence, that the software aspects of airborne systems and equipment comply with airworthiness requirements." It specifically focuses on model-based software development, object-oriented software, the use and qualification of software tools and the use of formal methods to complement or replace dynamic testing. Each of these key aspects is addressed by a dedicated *supplement* which modifies, complements, and completes the DO-178C core document. The supplements "should be used with and in the same way" as the core document [40]. In this overview we will specifically focus on the DO-178C core document and DO-333 (Formal Methods Supplement to DO-178C and DO-278A) [41].

The DO-178C first discusses general system aspects which are relevant for software development and defines the software life cycle processes, which then are addressed in turn: the software planning, development, and verification processes, as well as the configuration management, quality assurance, and certification processes. The norm also details the software life cycle data and addresses additional considerations, such as tool qualification requirements. In this section we will follow that structure and pinpoint the respective requirements and verification objectives amenable to static analysis techniques. Considerations for formal methods are addressed at the end of this section, tool qualification is addressed in Sec. 5.

2.1 System Aspects

System aspects relevant for software development include functional and operational requirements, performance requirements and safety-related requirements, including design constraints and design methods, in particular partitioning (cf. Sec. DO.2.1 of DO-178C [40]). The norm emphasizes that timing and performance characteristics require special attention since they affect the system software and the software-hardware boundaries and have to be included in the respective information flows.

The DO-178C defines five software levels (criticality levels) ranging from Level A (most critical) to Level E (least critical). According to Sec. DO.2.3 only partitioned software components can be assigned individual software levels by the system safety assessment process. Sec. DO.2.4, Architectural Considerations, states that "if partitioning and independence between software components cannot be demonstrated, all components are assigned the software level associated with the most severe failure condition to which the software can contribute". The standard defines partitioning as a "technique for providing isolation between software components to contain and/or isolate faults and potentially reduce the effort of the software verification process". Among others, a partitioned software component "should not be allowed to contaminate another partitioned software component's code, input/output, or data storage areas", and it "should be allowed to consume shared processor resources only during its scheduled period of execution". Thus, freedom of interference, both in the spatial and the temporal domain, is recognized as an important architectural property.

2.2 Software Planning & Development Process

In Sec. DO.4.4.2, Language and Compiler Considerations, the DO-178C points out that the software verification process needs to consider particular features of the programming language and compiler. In Sec. DO.4.5 the use of Software Development Standards is demanded which include Software Design Standards and Software Code Standards. One of the goals is to "disallow the use of constructs or methods that produce outputs that cannot be verified or that are not compatible with safety-related requirements". The defined Software Development Standards have to be taken into account during software design and coding (cf. Sec. DO.5.2.2 and Sec. DO.5.3.2).

The *Software Design Standards* are defined to focus on algorithmic constraints like exclusion of recursion, dynamic objects, or data aliases (cf. Sec. DO.11.7e). They should also include complexity restrictions like maximum level of nested calls, use of unconditional branches, or number of entry/exit points of code components (cf. Sec. DO.11.7f).

The *Software Code Standards* focus on the programming language. They identify the programming language to be used and should define a safety-oriented language subset (cf. Sec. DO.11.8a). To improve readability (and hence verifiability) they should cover style rules like length restrictions, indentation, and documentation rules (cf. Sec. DO.11.8c), and impose further constraints on code complexity, e.g., regarding the complexity of logical and numerical expressions (cf. Sec. DO.11.8d).

2.3 Software Verification Process

Like the DO-178B, the DO-178C addresses the incompleteness of testing techniques: "Verification is not simply testing. Testing, in general, cannot show the absence of errors". Since formal methods are sound they can completely satisfy some verification objectives while for others additional verification such as complimentary testing may be necessary. Purpose and objective of the software verification process are defined in the same way as in the DO-178B: The purpose of the software verification process is to detect and report errors that may have been introduced during the software development processes. Removal of the errors is an activity of the software development processes. The general objectives of the software verification process are to verify that the requirements of the system level, the architecture level, the source code level and the executable object code level are satisfied, and that the means used to satisfy these objectives are technically correct and complete.

As described in Sec. 2.1 non-functional software properties can affect the system and the software level, and consequently,

they are addressed at all levels of the software verification process. Design constraints may apply to ensure verifiability of the software.

The software verification process comprises reviews and analyses of high-level requirements, low-level requirements, the software architecture, the source code, and requires testing or formal analysis of the executable object code. One common verification objective at all levels is to demonstrate the compliance with the requirements of the parent level. As the system requirements include performance and safety-related requirements non-functional aspects like timing or storage usage can impact all stages. Consequently, the *compatibility with the target computer* is a verification objective among the high-level requirements, the low-level requirements, and at the software architecture level.

According to Sec. DO.6.2 the software verification activities have to address the "accuracy, completeness, and verifiability of the software requirements, software architecture, and Source Code". In particular objective 6.3.2.d for reviews and analysis of low-level requirements demands to ensure that each low-level requirement can be verified. Objective 6.4.3.e demands to ensure that the Software Design Standards were followed during the software design process and that deviations from the standards are justified. Also during review and analyses of source code (Sec. DO.6.3.4) the Software Development Standards have to be addressed. Objective 6.3.4.c which aims at verifiability demands to ensure that no statements and structures that cannot be verified are contained in the Source Code. Objective 6.3.4.d demands to show conformance to the Software Code Standards defined, e.g., that complexity limits have been considered. Deviations from the standards have to be justified.

In Sec. DO.6.3 of [40] the system response time is given as an example of target computer properties relevant for high-level requirements and low-level requirements. Computing the response time requires the worst-case execution time to be known. At the source-code level the objective *accuracy and consistency* explicitly includes determining the worst-case execution time, the stack usage, and runtime errors (memory usage, fixed-point arithmetic overflow and resolution, floating-point arithmetic, use of uninitialized variables). All these characteristics can be checked using formal analysis (cf. Sec. FM.6.3.4 of [41]).

The data and control flow of the software is of crucial importance for the verification of functional and non-functional correctness properties. In the DO-178C, the verification goal 6.3.3.b (Consistency) demands that "a correct relationship exists between the components of the software architecture. This relationship exists via data flow and control flow." It is complemented by the verification goal of Sec. DO.6.3.4.b (Compliance with the software architecture) which demands to "ensure that the source code matches the data flow and control flow defined in the software architecture". Obviously the data and control flow of the implemented software must match the intended data and control flow as specified in the software architecture, and unintended data and control flow must be avoided. In particular, this implies demonstrating freedom of interference between software components in mixed-criticality software. In addition, the data and control flow also determines the required effort for functional testing. In DO-178C, Objective 8 of Annex A Table A-7 requires that "Test coverage of software structure (data and control coupling) is achieved", referencing Sec. DO.6.4.4.2.c which states that structural coverage analysis should "confirm that the requirements-based testing has exercised the data and control coupling between code components". An in-depth discussion of data and control coupling analysis is given in [25].

Worst-case execution time and worst-case stack usage have to be considered at the Executable Object Code level. The reason is that the impact of the compiler, linker, and of hardware features on the worst-case execution time and stack usage has to be assessed. Both can be checked by formal analyses at the Executable Object Code level (cf. Sec. FM.6.7 of [41]). Runtime errors also can be addressed at the Executable Object level, e.g. to deal with robustness issues like out-of-range loop values and arithmetic overflows (cf. Sec. FM.6.7.b of [41]), or to verify the software component integration. The latter implies, e.g., detecting incorrect initialization of variables, parameter passing errors, and data corruption.

At the Executable Object Level complimentary testing is still required, e.g., to address transient hardware faults, or incorrect interrupt handling (cf. Sec. DO.6.4.3. of [40]). Formal analysis performed at the source code level can be used for verification objectives at the executable object code if property preservation between source code and executable code can be demonstrated (cf. Sec. FM.6.7.f of [41]).

2.4 Formal Methods (DO-333)

[41] defines formal methods as "mathematically based techniques for the specification, development, and verification of software aspects of digital systems". It distinguishes three categories of formal analyses: deductive methods, such as theorem proving, model checking, and abstract interpretation. The computation of worst-case execution time bounds and the maximal stack usage are listed as reference applications of abstract interpretation. The importance of soundness is emphasized: "an analysis method can only be regarded as formal analysis if its determination of a property is sound. Sound analysis means that the method never asserts a property to be true when it is not true."

Regarding the applicability of formal methods, the DO-333 states that "formal methods provide comprehensive assurance of particular properties only for those aspects that are formalized in the formal model, so defining the limits of the model is essential." Formal analyses at the source code level have to be based on a formal source code semantics. For formal analyses done at the object code level, the object code becomes a formal model the semantics of which are treated as they are by the target hardware. When formal analysis is used to meet a verification objective, it has to be ensured that each formal method used is correctly defined, justified, and appropriate to meet this verification objective (cf. Sec. FM.6.2.1 of [41]):

- all notations used for formal analysis should be formal notations

- the soundness of each formal analysis method should be justified
- all assumptions related to each formal analysis should be described and justified; for example assumptions associated with the target computer or about the data range limits.

At the Executable Object Level, coverage analysis depends on whether formal methods are used to complement or to replace dynamic testing methods. When any low-level testing is used to verify low-level requirements for a software component then the entire software component will be subject to test coverage analysis consisting of requirements-based coverage analysis and structural coverage analysis. Requirements-based coverage analysis establishes that verification evidence exists for all of the requirements of the system. Structural coverage analysis is necessary since no exhaustive testing is achievable and used to ensure that the testing performed is rigorous and sufficient (cf. Sec. DO.6.4. of [40], Sec. FM.6.7.1 of [41], and Sec. FM.12.3.5 of [41]).

When only formal methods are used to verify requirements for a software component, a different coverage analysis for that component has to be performed, consisting of the following steps (cf. Sec. FM.6.7.2 of [41]):

- requirements-based coverage analysis to determine how well the implementation of the software requirements has been verified.
- complete coverage of each requirement to ensure that all assumptions made during the formal analysis are verified. If the assumptions are all verified then the formal analysis can give complete coverage of each requirement.
- completeness of the set of requirements
- detection of unintended dataflow relationships
- detection of extraneous code, including dead code and deactivated code
- review and analyses of the formal analysis cases, procedures, and results for the executable object code.

3 Static Analysis & Abstract Interpretation

Static analysis often is perceived as a technique for source code analysis at the programming language level, but it can also be applied at the binary machine code level. In that case it does not compute an approximation of a programming language semantics, but an approximation of the semantics of the machine code of the microprocessor.

The theory of abstract interpretation [9] is a mathematically rigorous formalism providing a semantics-based methodology for static program analysis. The semantics of a programming language is a formal description of the behavior of programs. The most precise semantics is the so-called concrete semantics, describing closely the actual execution of the program. Yet in general, the concrete semantics is not computable. Even under the assumption that the program terminates, it is too detailed to allow for efficient computations. The solution is to introduce an abstract semantics that approximates the concrete semantics of the program and is efficiently computable. This abstract semantics can be chosen as the basis for a static analysis. Compared to an analysis of the concrete semantics, the analysis result may be less precise but the computation may be significantly faster. Abstract interpretation based static analyzers have been demonstrated to scale up to industry-size software projects containing millions of line of code [27, 30].

In the remainder of this section we will describe how static analysis, and abstract interpretation in particular, can be applied to code guideline checking, software architecture analysis, runtime error analysis, and to determine worst-case stack usage and worst-case execution times. Code guideline checking, software architecture analysis and run-time error analysis operate at the source code level. Worst-case stack usage analysis and worst-case execution time analysis are performed at the binary level, because they have to take the instruction set and hardware architecture into account. As explained above, a sound analysis computes a safe over-approximation of the concrete semantics and reports any potential defect of the defect classes under analysis:

- For worst-case execution time analysis soundness means that the reported WCET is never below the actual execution time in some execution environment. Overestimations may occur.
- In the same way, the computed stack height must never be below the stack usage in any concrete execution. Overestimations may occur.
- For run-time error analysis soundness means that the analysis never omits to signal an error that can appear in some execution environment. False alarms may occur.

3.1 Code Guideline Checking

Coding guidelines aim at improving code quality and can be considered a prerequisite for developing safety- or security-relevant software. In particular, obeying coding guidelines is strongly recommended by all current safety standards. The norms do not enforce compliance to a particular coding guideline, but define properties to be checked by the coding standards applied. As an example, the ISO 26262 gives a list of topics to be covered, including enforcement of low complexity, enforcing usage of a language subset, enforcing strong typing, and use of well-trusted design principles (cf. [15], Table 1). The language subset to be enforced should exclude, e.g., ambiguously defined language constructs, language constructs that could result in unhandled runtime errors, and language constructs known to be error-prone. As discussed in Sec. 2 the DO-178C is less prescriptive, but mandates coding guidelines nontheless.

There is a variety of code guidelines, in particular for C and C++, which are widely used in industry. The most prominent guidelines for C are MISRA C:2012 [39], ISO/IEC TS 17961

[16], the SEI CERT C Coding Standard [44], and the MITRE Common Weakness Enumeration CWE [46]. Prominent coding standards for C++ include MISRA C++:2008 [38], the SEI CERT C++ Coding Standard [43], the C++ Core Guidelines [7], the Adaptive AUTOSAR C++ Coding Guidelines [4], and the Joint Strike Fighter Air Vehicle C++ Coding Standards [34]. However, as of 2021, the discussion which C++ language features should be admitted to which extent for safety-critical software projects, is in full swing, and there is no clear consensus yet [19].

Most coding guidelines try to make coding rules easy to follow for programmers, and easy to check by automatic tools, hence, many coding rules are formulated at the syntactic level. They can be addressed by static analyzers operating purely on the program syntax.

However, obeying syntactic coding guidelines can reduce the risk of programming errors but not prevent them. Hence, all coding standards also explicitly contain semantic rules. These rules require a deeper understanding of the code as they focus on semantical properties which requires knowledge about variable values, pointer targets etc. To address such rules, and, in consequence, identify semantical code defects, *semantics-based* static analyses can be applied. Many semantical rules are associated with runtime errors due to undefined or unspecified behaviors of the programming language used (cf. Sec. 3.5). All safety norms, including DO-178C, consider demonstrating the absence of such runtime errors explicitly as a verification goal. They typically address general runtime errors (e.g., division by zero, invalid pointer accesses, arithmetic overflows), and additionally consider corruption of content, synchronization mechanisms, and freedom of interference in concurrent execution. This is reflected, e.g., in MISRA C:2012, e.g., by Rule 1.3. (*goal: preventing undefined or critical unspecified behavior*) and Directive 4.1 (*goal: minimization of run-time failures*)). Rule 1.3 and Directive 4.1 are examples for semantical guidance.

In contrast to other coding standards, MISRA C provides a clear classification which rules are based on semantic properties: typically such rules are labeled as system/undecidable. This is important since – due to their inherent undecidability – for semantical rules there cannot always be a correct and precise ”yes” or ”no” answer. As discussed above, there can always be false alarms (false positives), and, in case of unsound analyzers, also missed defects (false negatives). Therefore, developers have to be aware of the class and the operational context of the static analysis tool in use.

3.2 Software Architecture Analysis

All current safety norms require determining the data and control flow in the source code and making sure that it is compliant to the intended control and data flow as defined in the software architecture. In traditional static code analysis, data accesses via pointer variables and control flow by function pointer calls might be missed.

Using sound static analysis based on abstract interpretation, it is possible to guarantee the absence of runtime errors that could cause memory corruption and control flow corruption. Furthermore, it is possible to guarantee that in the analysis, all data and function pointer targets are considered and that the possible data and control coupling is fully captured. This way, a safe approximation of the data and control coupling between software components can be determined. That makes it possible to detect critical data and control flow errors and allows to complement traditional code coverage criteria by the degree of data and control coupling covered by the testing process, helping to identify relevant previously untested scenarios. In addition, freedom of spacial interference between software components can be demonstrated at the source code level [25].

3.3 Stack Usage Analysis

In safety-critical systems, stack overflows can cause catastrophic damage. The run-time stack (often just called “the stack”) typically is the only dynamically allocated memory area. It is used during program execution to keep track of the currently active procedures and facilitate the evaluation of expressions. The maximal stack usage has to be statically known: at configuration time of the system sufficient stack space has to be reserved for each task.

However, the stack height cannot be easily determined from the source code, since it depends on the dynamic call depth of functions, on compiler optimizations, and on link-time optimizations. Overestimating the maximum stack usage means wasting memory resources. Underestimation can lead to stack overflows where memory cells from the stacks of different tasks or other memory areas are overwritten. This can cause crashes due to memory protection violations and can trigger arbitrary erroneous program behavior, if return addresses or other parts of the execution state are modified. In consequence stack overflows are typically hard to diagnose and hard to reproduce, but they are a potential cause of catastrophic failure. One example is the series of accidents caused by unintended acceleration of the 2005 Toyota Camry: the expert witness’ report commissioned by the Oklahoma court in 2013 identifies a stack overflow as most probable failure cause [6, 47].

For safe stack size analysis, it is important to work on fully linked binary code, since the effects of code generation – inserting padding bytes, register allocation, etc. – or link- time optimizations have to be taken into account. Hence the static stack usage analysis cannot be based on the source code but must work on the executable machine code. It approximates the semantics of the machine code of the microprocessor by using an abstract model of the processor architecture. The abstract model does not need to cover the entire state of the microprocessor, only the parts affecting the stack space are needed. The hardware state relevant for worst-case stack analysis includes the processor registers and the memory cells. For a naive analysis, only the stack pointer register is needed, but for precise results it is important to perform an elaborate value analysis on the contents of processor register and memory cells.

In the following we will give an overview of the structure and analysis phases of StackAnalyzer [2], which is an example tool from this category. First, the control-flow graph is reconstructed from the input file, the binary executable. Then a static value analysis computes value ranges for registers and address

ranges for instructions accessing memory. StackAnalyzer reports computed branch and call instructions in case their targets cannot be automatically resolved by its value analysis, as well as unbounded recursions. Missing information such as recursion bounds and call targets can be manually specified in a formal annotation language, *AIS* [21]. Function pointer targets can also be automatically imported from an Astrée analysis. By concentrating on the value of the stack pointer during value analysis, StackAnalyzer computes how the stack increases and decreases along the various control-flow paths. This information can be used to derive the maximum stack usage of the entire task. StackAnalyzer takes the entire application into account and interprocedurally analyzes each call site with its precise stack height. The results of StackAnalyzer are presented as annotations in a combined call graph and control-flow graph. It shows the critical path, i.e., the path on which the maximum stack usage is reached which gives important feedback for optimizing the stack usage of the application under analysis. Experimental results show that the analysis is fast and precise so that only few explicit annotations are needeed [21].

3.4 Worst-Case Execution Time Analysis

In real-time systems the overall correctness depends on the correct timing behavior: each real-time task has to finish before its deadline, hence, reliable bounds of the worst-case execution time (WCET) of real-time tasks have to be determined.

With end-to-end timing measurements timing information is only determined for one concrete input. Due to caches and pipelines the timing behavior of an instruction depends on the program path executed before. Therefore, usually no full test coverage can be achieved and there is no safe test end criterion. Techniques based on code instrumentation modify the code which can significantly change the cache and pipeline behavior (probe effect): the times measured for the instrumented software do not necessarily correspond to the timing behavior of the original software.

One safe method for timing analysis is static analysis by Abstract Interpretation which provides guaranteed upper bounds for WCET of tasks. Static WCET analyzers are available for complex processors with caches and complex pipelines, and, in general, support both single- and multi-core processors. A prerequisite is that good models of the processor/System on-Chip (SoC) architecture can be determined. However, there are modern high performance SoCs which contain unpredictable and/or undocumented components that influence the timing behavior. Analytical results for such processors are unrealistically pessimistic.

A hybrid WCET analysis combines static value, loop and path analysis with measurements to capture the timing behavior of tasks. Compared to end-to-end measurements the advantage of hybrid approaches is that measurements of short code snippets can be taken which cover the complete program under analysis. Based on these measurements a worst-case path can be computed.

In the following we will focus on applications of static analysis to WCET computation and illustrate the basic principles to establish their contribution to verification goals of DO-178C. For an overview of methods and tools for WCET analysis we refer to [48] , and recommend [33] for a general survey on methods for timing analysis on multi-core processors.

3.4.1 Timing Predictability

In general, a system is predictable if it is possible to predict its future behavior from the information about its current state [5]. The primary sources of uncertainty in execution time of an instruction sequence are the program input and the hardware state in which its execution begins. Hardware-related timing predictability can be expressed as the maximal variance in execution time due to different hardware states for an arbitrary but fixed input. Analogously, software-related timing predictability corresponds to the maximal variance in execution time due to different inputs for an arbitrary but fixed initial hardware state.

Even in single-core processors timing predictability is compromised by performance-enhancing hardware mechanisms like caches, pipelines, out-of-order execution, branch prediction and other mechanisms for speculative execution, which can cause significant variations in timing depending on the hardware state. On multi-core architectures, in addition the inter-core parallelism becomes relevant. To interconnect the several cores, buses, meshes, crossbars, and also dynamically routed communication structures are used. In that case, the interference delays due to conflicting, simultaneous accesses to shared resources (e.g. main memory) are the main cause of imprecision.

3.4.2 Fully Static WCET Analysis

Static analysis by Abstract Interpretation can provide guaranteed upper bounds for WCET of tasks. Over the past decades, a standard architecture has emerged [10, 12] which neither requires code instrumentation nor debug information and is composed of the following major building blocks:

Decoding: The instruction decoder identifies the machine instructions and reconstructs the call and control-flow graph. To ensure safety of later analysis results, this graph itself must be safe, i.e., all possible paths that can occur during execution of the program must be represented.

Value analysis: Value analysis aims at statically determining the contents of the registers and memory cells at each program point and for each execution context. The results of the value analysis are used to predict the addresses of data accesses, the targets of computed calls and branches, and to find infeasible paths caused by conditions that always evaluate to true, or always evaluate to false in a specific context.

Micro architectural analysis: The execution of a program is statically simulated by feeding instruction sequences from the control-flow graph to a micro-architectural timing model which is centered around the cache and pipeline architecture. It computes the system state changes induced by the instruction sequence at cycle granularity and keeps track of the elapsing clock cycles.

Path analysis: Based on the results of the combined cache/pipeline analysis the worst-case path of the analyzed code is computed with respect to the execution timing. The execution time of the computed worst-case path is the worst-case execution time for the task.

A tool which implements this architecture is the static WCET analyzer aiT [45]. It is available for a variety of microprocessors including multi-core processors which can be configured in a timing-predictable way to avoid or bound inter-core interferences like Infineon AURIX TC27x [1].

3.4.3 Hybrid WCET Analysis

Hybrid WCET analysis tools combine static context-sensitive path analysis with real-time instruction-level tracing to provide worst-case execution time estimates. By its nature, an analysis using measurements to derive timing information is aware of timing interference due to concurrent execution and multi-core resource conflicts, because the effects of asynchronous events (e.g. activity of other running cores or DRAM refreshes) are directly visible in the measurements.

An example tool is the hybrid WCET Analyzer TimeWeaver [29, 3], which builds on non-intrusive instruction-level tracing: the probe effect is avoided since no code instrumentation is needed. The computed estimates are safe upper bounds with respect to the given input traces, i.e., TimeWeaver derives an overall upper timing bound from the execution time observed in the given traces. Thus, the coverage of the input traces on the analyzed code is an important metric that influences the quality of the computed WCET estimates.

The trace information needed for running TimeWeaver is provided by embedded trace units of modern processors, like NEXUS IEEE-ISTO 5001, Infineon TriCore MCDS, or ARM CoreSight. They allow the fine-grained observation of a program execution on single- and multi-core systems.

The main inputs for TimeWeaver are the fully linked executable(s), timed traces and the location of the analyzed code in the memory (entry point, which usually is the name of a task or function). The analysis proceeds in several stages: decoding, loop/value analysis, trace analysis, and path analysis, most of which are shared with aiT [45]. The main difference is that micro-architectural analysis is replaced by the trace analysis stage:

Trace analysis: The given traces are analyzed such that each trace event is mapped to a program point in the control-flow graph and the trace points and segments are defined. In case a preemptive system has been traced, interrupts are detected and reported. The extracted timing information, i.e., the clock cycles which have been elapsed between two consecutive trace points are annotated to the CFG in a context-sensitive manner [29].

3.5 Run-Time Errors and Data Races

In this section we focus on source-level runtime errors due to undefined or unspecified behaviors of the programming language used. Examples are faulty pointer manipulations, numerical errors such as arithmetic overflows and division by zero, data races, and synchronization errors in concurrent software. Such errors can cause software crashes, invalidate separation mechanisms in mixed-criticality software, and are a frequent cause of errors in concurrent and multi-core applications. At the same time, these defects also constitute security vulnerabilities, and have been at the root of a multitude of cybersecurity attacks, in particular buffer overflows, dangling pointers, or race conditions [23].

Runtime errors are one important cause of software-induced memory corruption in safety-critical systems. The other two main causes are stack overflows, and miscompilation, where the compiler silently generating erroneous code from a correct input program. As described above, with abstract interpretation, the absence of runtime errors and stack overflows can be proven; when using a formally proven compiler like CompCert [22, 18] no miscompilation is possible, hence all main sources of software-induced memory corruption can be covered.

In the following we will give a brief overview of the Astrée analyzer as an example of sound static runtime error analysis tools [31][37]. To achieve high precision Astrée provides a variety of abstract domains covering, e.g. , intervals, octagons, digital filters, finite state machines, and interpolations. The memory domain empowers Astrée to exactly analyze pointer arithmetic and union manipulations and to perform a type-safe analysis of absolute memory addresses. Floating-point computations are precisely modeled while keeping track of possible rounding errors. Astrée also implements a low-level concurrent semantics [35] which provides a scalable sound abstraction covering all possible thread interleavings. The interleaving semantics enables Astrée, in addition to the classes of runtime errors found in sequential programs, to report data races, and lock/unlock problems, i.e., inconsistent synchronization. The set of shared variables does not need to be specified by the user: Astrée assumes that every global variable can be shared, and discovers which ones are effectively shared, and on which ones there is a data race. Since Astrée is aware of all locks held for every program point in each concurrent thread, Astrée can also report all potential deadlocks. The abstract domains are parameterized, which enables users to fine-tune the precision of the analyzer to the software under analysis to minimize the number of false alarms.

In its data and control flow analysis module, Astrée tracks accesses to global, static, and local variables in case those accesses are made outside of the frame in which the local variables are defined (e.g., because their address is passed into a called function). The soundness of the analysis ensures that all potential targets of data and function pointers are taken into account. Function pointer targets are automatically resolved and can be exported in AIS format to support binary-level static analyzers. Astrée's data and control flow reports show the number of read and write accesses for every global, static, and out-of-frame local variable, lists the location of each access and shows the function from which the access is made. All variables are classified as being thread-local, effectively shared between different threads, or subject to a data race. Astrée also suports data and control coupling analysis [26], and can check compliance to commonly used coding guidelines such as MISRA C/C++,

CWE, SEI CERT C/C++, Adaptive Autosar C++, etc. Furthermore, Astrée includes a program slicer, and a user-configurable taint analysis [24].

Practical experience on avionics and automotive industry applications are given in [31][36][32]. They show that industry-sized programs of millions of lines of code can be analyzed in acceptable time with high precision for runtime errors and data races.

4 Coverage of Verification Objectives

In this section we give an overview of the verification objectives of DO-178C to which static analysis methods can be applied. The relevant verification objectives are summarized in Annex-A of [41]. We will explicitly list the sections of the DO-178C and the DO-333 that address verification objectives to which sound static analysis for WCET (worst-case execution time), WCSU (worst-case stack usage), and RTE (runtime errors), as well as simple unsound code guideline checking (CG) can contribute. The section names of the DO-333 match the corresponding sections in the DO-178C, e.g. Sec. FM.6.3.1.c of DO-333 corresponds to Sec. 6.3.1.c of DO-178C, so in the following we will just use the numbering from [41] for simplicity.

In general, worst-case execution time analysis, worst-case stack usage analysis and runtime error analysis contribute to all objectives related to the target environment. Stack usage, response times and execution times are determined by the target computer. They can cause violations of high-level requirements, violations of low-level requirements, incompatibility of the software architecture with the target computer, and they can affect the accuracy and consistency of the source code. Sound static source code analysis enables sound data and control flow analysis which is required to demonstrate consistency between software architecture level and source code level. Detecting runtime errors is required to deal with robustness issues like out-of-range loop values and arithmetic overflows, or to verify the software integration. The latter implies, e.g., detecting incorrect initialization of variables, parameter passing errors, and data corruption.

The sections Sec. FM.6.3.1.c, FM.6.3.2.c, and FM.6.3.3.c address the compatibility with the target computer, hence, sound static analyses for WCET, WCSU, and RTE are relevant for all of them. The timing aspect is emphasized as response times and is explicitly listed as an example in Sec. FM.6.3.1.c and Sec. FM.6.3.2.c. Sound analyzers for WCET, WCSU and RTE can report unreached code and dead code, thus also contributing to Sec. FM.6.3.3.a, Sec. FM.6.3.4.a, and Sec. FM.6.3.4.e. Compliance to low-level requirements (Sec. FM.6.3.4.a) also necessitates the absence of programming defects as reported by sound RTE analysis. The data and control flow analysis provided by sound RTE analysis relates to Sec. FM.6.3.3.b and FM.6.3.4.b. Subsequently, Sec. FM.6.3.4.f explicitly lists determining worst-case execution time, stack usage, and absence of runtime errors as verification objectives.

Sec. FM.6.7 addresses the formal analysis of the executable object code. Sound analysis of WCET and WCSU both contribute to Sec. FM.6.7.e by computing safe bounds to the worst-

Obj.	Ref.	WCET	WCSU	CG	RTE
Table FM.A-3					
3	FM.6.3.d FM.6.3.1.c	✓	✓		✓
Table FM.A-4					
3	FM.6.3.d FM.6.3.2.c	✓	✓		✓
5	FM.6.3.f FM.6.3.2.e			✓	✓
8	FM.6.3.3.a	✓*	✓		✓
9	FM.6.3.c FM.6.3.3.b	✓*	✓*		✓
10	FM.6.3.d FM.6.3.3.c	✓	✓		✓
11	FM.6.3.e FM.6.3.3.d	✓*	✓*	✓	✓
12	FM.6.3.f FM.6.3.3.e	✓*	✓*	✓	✓
13	FM.6.3.3.f	✓	✓		✓
Table FM.A-5					
1	FM.6.3.a FM.6.3.4.a	✓	✓		✓
2	FM.6.3.a FM.6.3.4.b				✓
3	FM.6.3.e FM.6.3.4.c			✓	✓
4	FM.6.3.f FM.6.3.4.d			✓	✓
5	FM.6.3 FM.6.3.4.e			✓	✓
6	FM.6.3.b FM.6.3.c FM.6.3.4.f	✓	✓	✓	✓
7	6.3.5.a	✓	✓		✓
Table FM.A-6					
1	FM.6.7.a FM.6.7.c	✓	✓		✓**
2	FM.6.7.b FM.6.7.c	✓	✓		✓**
3	FM.6.7.d FM.6.7.c	✓	✓		✓**
4	FM.6.7.b FM.6.7.c	✓	✓		✓**
5	6.4.e FM.6.7.e	✓	✓		✓**

* Leveraging the built-in value analysis
** Using CompCert to ensure semantic preservation

Table 1: Coverage of DO-178C verification objectives by sound static WCET analysis (WCET), sound static stack usage analysis (WCSU), static code guideline checking (CG) and sound static runtime error analysis (RTE).

case execution time or stack consumption of the software under certification. The possibility to deeply investigate the behavior of the analyzed software on the assembly level allows to derive information about timing contribution of specific parts of the software but also to trace back contents of memory cells and/or register values to the corresponding source code. Hence sound static binary code analysis as needed for WCET and WCSU also contributes to the paragraphs *a* till *d* in Sec. FM.6.7.

As specified in Sec. FM.6.7.f, relevant properties can also be verified on source code level if property preservation between source and object code level is ensured. For the language C, this can be achieved by using the formally verified compiler CompCert [22]. The code it produces is proven to behave exactly as specified by the semantics of the source C program. Leveraging this, sound static RTE analysis is relevant for all points in Sec. FM.6.7 and also contributes to Sec. FM.6.6.a: program slicing and taint analysis, e.g. , as available in Astrée, support identifying program parts contributing to run-time errors, and program parts affected by data corruption, respectively, hence possibly pinpointing flaws in the Data Item File. Furthermore a type-safe analysis of absolute addresses at the source code level as available in Astrée can detect incorrect hardware addresses, and also reports memory overlaps. The value analysis component of sound binary-level analyzers for WCET and WCSU supports proving that memory accesses are made to the expected memory regions at the Executable Object Level. These analyses contribute to Sec. FM.6.3.5.a which deal with review and analyses of the output of the integration process.

Code guideline checkers, either as standalone tools, or included in sound RTE analyzers such as Astrée typically also compute code metrics, and provide checks to demonstrate conformance to thresholds defined. These capabilities support requirements of the Software Design Standards (e.g., recursion, dynamic objects, call nesting levels) and of the Software Code Standards (e.g., style rules, expression complexity). They also support traceability requirements by appropriate coding rule checks. In summary, they contribute to various verification objectives, in particular the objectives Sec. FM.6.3.3.d, Sec. FM.6.3.3.e, Sec. FM.6.3.4.c, Sec. FM.6.3.4.d, and Sec. FM.6.3.4.e, in general supporting to check the verifiability of software design and implementation.

5 Tool Qualification

Whenever the output of a tool is either part of a safety-critical system to be certified or the tool output is used to eliminate or reduce any development or verification effort for such a system, that tool needs to qualified. DO-178C regulates *when* a tool qualification is to be applied and DO-330 [42] *gives guidance* on the tool qualification requirements.

5.1 Relevant Tool Qualification Levels (TQL)

First, the necessary qualification activities and results have to be identified. For this, DO-330 defines the so-called tool qualification level (TQL). There are five different levels, from the most critical level TQL-1 down to TQL-5.

The TQL is determined by the potential tool impact and the software level. There are three tool impact categories where *Criteria 1* denotes the highest impact and *Criteria 3* the lowest. *Criteria 1* does not apply to static verification tools since it is associated with tools whose output is part of the airborne software. Analysis and verification tools are subject to *Criteria 2/3*. The difference between the latter two categories is whether the output of the tool is used to justify the elimination or reduction of other verification or development activities or not. The following table illustrates how the TQLs are assigned based on *Criteria* and *Software Level*.

Software Level	**Criteria**		
	1	2	3
A	TQL-1	TQL-4	TQL-5
B	TQL-2	TQL-4	TQL-5
C	TQL-3	TQL-5	TQL-5
D	TQL-4	TQL-5	TQL-5

To summarize, the tool qualification levels relevant for static analyzers are TQL-4 and TQL-5. TQL-4 applies when the software under analysis is Level A/B and the tool categorized as criteria 2. In all other cases, TQL-5 applies. In the following, we summarize the qualification material that is required for a TQL-4 qualification, and outline a well-proven example on how this can be supported.

5.2 Tool Qualification Requirements

The tables *T-0* to *T-10* in DO-330 define requirements (called objectives) to the tool qualification. Each table addresses a different process associated with the life cycle of the tool under qualification in order to cover the relevant aspects of the affected processes.

Summarizing the objectives in these tables, the qualification data to be provided for a TQL-4 verification tool boils down to the following. First, tool (operational) requirements have to be specified, i.e. , the intended tool behavior must be defined in an explicit and detailed manner. Second, test cases have to be created which cover all those functional requirements. This includes providing descriptions of test case objectives, execution procedures, expected results, and records of their execution. Unique identifiers for requirements and test cases allow to establish trace data between those data elements which in the end allows to claim coverage of all requirements by successfully passed test case executions. For the static analyzers described in Sec. 3 these requirements are fully covered by so-called *Qualification Support Kits* (QSKs).

In addition to the tool behavior, the qualification material needs to address certain aspects of the tool development processes. The *Tool Qualification Plan* (TQP) gives an overview to the tool qualification as a whole. The *Tool Development Plan* (TDP) includes the objectives, standards, and tool life cycle(s) to be used in the tool development processes. The *Tool Verification Plan* (TVP) is a description of the activities to satisfy the tool verification process objectives. The *Tool Configuration Management Plan* (TCMP) establishes the methods to be used to satisfy the objectives of the TCM process throughout the tool life cycle. The *Tool Quality Assurance Plan* (TQP) establishes the methods to be used to satisfy the objectives of the tool quality assurance process. The execution of all activities defined in

these plans needs to produce evidence records that are archived and part of the qualification data. AbsInt provides these evidences about the applied tool life cycle activities as part of the QSKs.

Typically, static analyzers are categorized as so-called *Commercial Off-The-Shelf* (COTS) tools, basically meaning that the tool developer is different from the tool user. To qualify COTS tools, the tool developer provides the basis for the required qualification material from a "tool developers" perspective. The tool user then needs to either adapt the material or to describe how its tool use maps to the data provided by the developer. For example: the tool user might needs to define how the tool is used and which functionality (from the provided operational requirements) is used.

To provide high confidence in the correct functioning of a tool it is necessary to demonstrate that the tool works correctly in the operational context of its users. This is a common requirement of most current safety standards. The correct functioning of a tool might be affected by the OS version, system libraries installed, software patch levels, etc. For the tools listed in Sec. 3 there is dedicated support for tool users to automatically execute the QSK test cases and automatically create all data required for the certification package.

6 Conclusion

Static analysis has evolved to be a standard method in the software development and verification process. It can be applied to various verification activities required for DO-178C certification, in particular by performing code guideline checking, data and control coupling analysis, interference analysis, worst-case stack and execution time analysis, and runtime error analysis. In this article we precisely identified the verification requirements and objectives that can be covered by static analysis and its formal method, abstract interpretation. For each application of static analysis mentioned above we summarized the underlying analysis concepts and illustrated them with practical examples. We also summarized the required tool qualification activities, and illustrated them with a well-proven example approach to tool qualification.

References

[1] AbsInt GmbH. aiT WCET Analyzer Website. http://www.AbsInt.com/ait.

[2] AbsInt GmbH. StackAnalyzer Website. http://www.AbsInt.com/sa.

[3] AbsInt GmbH. Timeweaver website.

[4] AUTOSAR. Guidelines for the use of the C++14 language in critical and safety-related systems, 2018.

[5] P. Axer, R. Ernst, H. Falk, A. Girault, D. Grund, N. Guan, B. Jonsson, P. Marwedel, J. Reineke, C. Rochange, M. Sebastian, R. von Hanxleden, R. Wilhelm, and W. Yi. Building timing predictable embedded systems. *ACM Transactions on Embedded Computing Systems*, 2013. Accepted.

[6] M. Barr. Bookout v. Toyota, 2005 Camry software Analysis by Michael Barr. http://www.safetyresearch.net/Library/BarrSlides_FINAL_SCRUBBED.pdf, 2013.

[7] H. S. Bjarne Stroustrup. C++ Core Guidelines. https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines [retrieved: Jan. 2020].

[8] CENELEC EN 50128. Railway applications – Communication, signalling and processing systems – Software for railway control and protection systems, 2011.

[9] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proc. of POPL'77*, pages 238–252. ACM Press, 1977.

[10] A. Ermedahl. *A Modular Tool Architecture for Worst-Case Execution Time Analysis*. Phd thesis, Uppsala University, 2003.

[11] C. Faure and V. Delebarre. Automatic proof of freedom from interference with iffree. In *Proccedings of the 10th European Conference on Software Architecture Workshops, Copenhagen, Denmark, November 28 - December 2, 2016*, page 36, 2016.

[12] C. Ferdinand, R. Heckmann, M. Langenbach, F. Martin, M. Schmidt, H. Theiling, S. Thesing, and R. Wilhelm. Reliable and precise WCET determination for a real-life processor. In *Proceedings of EMSOFT 2001, First Workshop on Embedded Software*, volume 2211 of *LNCS*, pages 469–485. Springer, 2001.

[13] IEC 61508. Functional safety of electrical/electronic/programmable electronic safety-related systems, 2010.

[14] ISO 26262. Road vehicles – Functional safety, 2018.

[15] ISO 26262. Road vehicles – Functional safety – Part 6: Product development at the software level, 2018.

[16] ISO/IEC. Information Technology – Programming Languages, Their Environments and System Software Interfaces – Secure Coding Rules (ISO/IEC TS 17961), Nov. 2013.

[17] D. Kästner. Applying Abstract Interpretation to Demonstrate Functional Safety. In J.-L. Boulanger, editor, *Formal Methods Applied to Industrial Complex Systems*. ISTE/Wiley, London, UK, 2014.

[18] D. Kästner, J. Barrho, U. Wünsche, M. Schlickling, B. Schommer, M. Schmidt, C. Ferdinand, X. Leroy, and S. Blazy. CompCert: Practical Experience on Integrating and Qualifying a Formally Verified Optimizing Compiler. In *ERTS2 2018 - Embedded Real Time Software and Systems*, Toulouse, France, Jan. 2018.

[19] D. Kästner, C. Cullmann, G. Gebhard, S. Hahn, T. Karos, L. Mauborgne, S. Wilhelm, and C. Ferdinand. Safety-Critical Software Development in C++. In A. Casimiro, F. Ortmeier, E. Schoitsch, F. Bitsch, and P. Ferreira, editors, *Computer Safety, Reliability, and Security. SAFECOMP 2020 Workshops*, pages 98–110, Cham, 2020. Springer International Publishing.

[20] D. Kästner and C. Ferdinand. Efficient Verification of Non-Functional Safety Properties by Abstract Interpretation: Timing, Stack Consumption, and Absence of Runtime Errors. In *Proceedings of the 29th International System Safety Conference ISSC2011*, Las Vegas, 2011.

[21] D. Kästner and C. Ferdinand. Proving the Absence of Stack Overflows. In *SAFECOMP '14: Proceedings of the 33th International Conference on Computer Safety, Reliability and Security*, volume 8666 of *LNCS*, pages 202–213. Springer, September 2014.

[22] D. Kästner, X. Leroy, S. Blazy, B. Schommer, M. Schmidt, and C. Ferdinand. Closing the gap – the formally verified optimizing compiler CompCert. In *SSS'17: Developments in System Safety Engineering: Proceedings of the Twenty-fifth Safety-critical Systems Symposium*, pages 163–180. CreateSpace, 2017.

[23] D. Kästner, L. Mauborgne, and C. Ferdinand. Detecting Safety- and Security-Relevant Programming Defects by Sound Static Analysis. In J.-C. B. Rainer Falk, Steve Chan, editor, *The Second International Conference on Cyber-Technologies and Cyber-Systems (CYBER 2017)*, volume 2 of *IARIA Conferences*, pages 26–31. IARIA XPS Press, 2017.

[24] D. Kästner, L. Mauborgne, S. Wilhelm, and C. Ferdinand. High-Precision Sound Analysis to Find Safety and Cybersecurity Defects. In *10th European Congress on Embedded Real Time Software and Systems (ERTS 2020)*, Toulouse, France, Jan. 2020.

[25] D. Kästner, L. Mauborgne, S. Wilhelm, and C. Ferdinand. Static Data and Control Coupling Analysis. *Submitted to the 11th European Congress on Embedded Real Time Software and Systems (ERTS 2022)*, March 2022.

[26] D. Kästner, L. Mauborgne, S. Wilhelm, and C. Ferdinand. Static Data and Control Coupling Analysis. In *ERTS2 2022 - Embedded Real Time Software and Systems. To appear*, Toulouse, France, Mar. 2022.

[27] D. Kästner, A. Miné, L. Mauborgne, X. Rival, J. Feret, P. Cousot, A. Schmidt, H. Hille, S. Wilhelm, and C. Ferdinand. Finding All Potential Runtime Errors and Data Races in Automotive Software. In *SAE World Congress 2017*. SAE International, 2017.

[28] D. Kästner, M. Pister, G. Gebhard, M. Schlickling, and C. Ferdinand. Confidence in Timing. *Safecomp 2013 Workshop: Next Generation of System Assurance Approaches for Safety-Critical Systems (SASSUR)*, September 2013.

[29] D. Kästner, M. Pister, S. Wegener, and C. Ferdinand. TimeWeaver: A Tool for Hybrid Worst-Case Execution Time Analysis. In S. Altmeyer, editor, *19th International Workshop on Worst-Case Execution Time Analysis (WCET 2019)*, volume 72 of *OpenAccess Series in Informatics (OASIcs)*, pages 1:1–1:11, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[30] D. Kästner, B. Schmidt, M. Schlund, L. Mauborgne, S. Wilhelm, and C. Ferdinand. Analyze This! Sound Static Analysis for Integration Verification of Large-Scale Automotive Software. In *Proceedings of the SAE World Congress 2019 (SAE Technical Paper)*. SAE International, 2019.

[31] D. Kästner et al. Finding All Potential Runtime Errors and Data Races in Automotive Software. In *SAE World Congress 2017*. SAE International, 2017.

[32] D. Kästner et al. Analyze This! Sound Static Analysis for Integration Verification of Large-Scale Automotive Software. In *Proceedings of the SAE World Congress 2019 (SAE Technical Paper)*. SAE International, 2019.

[33] C. Maiza, H. Rihani, J. M. Rivas, J. Goossens, S. Altmeyer, and R. I. Davis. A survey of timing verification techniques for multi-core real-time systems. *ACM Comput. Surv.*, 52(3), jun 2019.

[34] L. Martin. Joint strike fighter air vehicle c++ coding standards for the system development and demonstration program, 2005.

[35] A. Miné. Static analysis of run-time errors in embedded real-time parallel C programs. *Logical Methods in Computer Science (LMCS)*, 8(26):63, Mar. 2012.

[36] A. Miné and D. Delmas. Towards an Industrial Use of Sound Static Analysis for the Verification of Concurrent Embedded Avionics Software. In *Proc. of the 15th International Conference on Embedded Software (EMSOFT'15)*, pages 65–74. IEEE CS Press, Oct. 2015.

[37] A. Miné et al. Taking Static Analysis to the Next Level: Proving the Absence of Run-Time Errors and Data Races with Astrée. In *8th European Congress on Embedded Real Time Software and Systems (ERTS 2016)*, Toulouse, France, Jan. 2016.

[38] MISRA (Motor Industry Software Reliability Association) Working Group. MISRA C++:2008 Guidelines for the use of the C++ language in critical systems, 2008.

[39] MISRA (Motor Industry Software Reliability Association) Working Group. *MISRA-C:2012 Guidelines for the use of the C language in critical systems*. MISRA Limited, Mar. 2013.

[40] Radio Technical Commission for Aeronautics. RTCA DO-178C. Software Considerations in Airborne Systems and Equipment Certification, 2011.

[41] Radio Technical Commission for Aeronautics. RTCA DO-333. Formal Methods Supplement to DO-178C and DO-278A, 2011.

[42] Radio Technical Commission for Aeronautics. Software Tool Qualification Considerations, 2011.

[43] Software Engineering Institute SEI – CERT Division. SEI CERT C++ Coding Standard.

[44] Software Engineering Institute SEI – CERT Division. *SEI CERT C Coding Standard – Rules for Developing Safe, Reliable, and Secure Systems*. Carnegie Mellon University, 2016.

[45] J. Souyris, E. Le Pavec, G. Himbert, V. Jégu, G. Borios, and R. Heckmann. Computing the worst case execution time of an avionics program by abstract interpretation. In *Proceedings of the 5th Intl Workshop on Worst-Case Execution Time (WCET) Analysis*, pages 21–24, 2005.

[46] The MITRE Corporation. CWE – Common Weakness Enumeration. https://cwe.mitre.org [retrieved: July 2019].

[47] Transcript of Morning Trial Proceedings had on the 14th day of October, 2013 Before the Honorable Patricia G. Parrish, District Judge, Case No. CJ-2008-7969. http://www.safetyresearch.net/Library/Bookout_v_Toyota_Barr_REDACTED.pdf, October 2013.

[48] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström. The worst-case execution-time problem—overview of methods and survey of tools. *ACM Transactions on Embedded Computing Systems*, 7(3):1–53, 2008.

[49] B. Zimmer, C. Dropmann, and J. U. Hanger. A systematic approach for software interference analysis. In *25th IEEE International Symposium on Software Reliability Engineering, ISSRE 2014, Naples, Italy, November 3-6, 2014*, pages 78–87, 2014.

Session Th.2.B

Assurance By Design

Thursday 2nd June

11:30

–

Room Lauragais

Architecture-Supported Audit Processor: Interactive, Query-Driven Assurance

Sam Procter
sprocter@sei.edu.edu
Software Engineering Institute, Carnegie Mellon University
Pittsburgh, Pennsylvania

Jerome Hugues
jjhugues@sei.edu.edu
Software Engineering Institute, Carnegie Mellon University
Pittsburgh, Pennsylvania

ABSTRACT

Establishing that safety-critical systems are actually safe requires a large effort and involves a range of tasks, from conducting preliminary hazard analyses to creating detailed assurance cases. This paper introduces the Architecture-Supported Audit Processor, or ASAP, which generates a number of safety-specific system views that deeply integrate a system's architecture and arguments about its safety. These views are generated interactively and automatically using safety-specific extensions to the Architecture Analysis and Design Language (AADL). Though use of the tooling and views do not require the use of any particular process, they align well with a system-theoretic approach. This paper discusses the background and use of ASAP on a demonstrative example.

KEYWORDS

Model-Based Engineering, System Safety, Hazard Analysis, Architecture Analysis and Design Language (AADL)

1 INTRODUCTION

Safety-critical systems, i.e., those systems whose failure would result in death, injury, or unacceptable financial losses, are becoming increasingly sophisticated and software reliant [10]. Developing confidence in the safe and reliable behavior of these systems requires efforts to assure them. The cost and time required for these assurance efforts is significant, and it has been argued that they are a bottleneck that prevents the rapid deployment of new and improved versions [5]. Additionally, many assurance practices were not designed for modern systems and do not consider the impact of software on safety [10], nor do they take advantage of model-based engineering techniques for the traceability of safety artifacts to system elements.

In this paper, we report on a tool-supported approach that addresses these shortcomings. We propose contextualizing assurance evidence in an environment that supports modern system development practices, and that explicitly links assurance evidence to safety argumentation. We present the *Architecture-Supported Audit Processor* (ASAP), an interactive tool which follows this strategy and supports common assurance activities in an architecture-centric, model-based system development environment. The tool provides an assurance-specific view of a system that builds on previous work on merging modern safety analysis and system architecture [15].

(1) **Research Context** We identify three research ideas from the safety and argumentation communities and discuss how they illuminate a path forward for next-generation hazard anlaysis.
(2) **SAFE Improvements** We propose an approach for performing safety analysis that builds on the *Systematic Analysis of Faults and Errors* (SAFE) hazard analysis technique, which supports model-based, compositional safety argumentation.
(3) **Tool** We describe the inputs, use cases, and outputs of the ASAP tool.

In order to better explain the use of ASAP, we also orient its inputs and outputs in the process of a popular hazard analysis technique:

(4) **Mapping to STPA** We establish an example mapping to STPA [13], a popular system-theoretic hazard analysis. Our mapping leverages system development tasks that are already performed in a typical development and presents safety information in an interactive, navigable format that can be queried.

The remainder of the paper is organized as follows: Section 2 surveys three research topics which informed the design of ASAP. Section 3 describes background concepts, including languages, tooling, and theory this work directly builds on. Section 4 describes ASAP and its application to a small system. Section 5 presents related work. We discuss future work in Section 6 and conclude in Section 7.

2 SURVEY: THREE CHALLENGES TO ASSURANCE

In addition to the technical background, covered in the next section, this work has been informed by three challenges to system assurance, which we describe here.

2.1 Assurance evidence should be contextualized with explicit safety arguments

One of the challenges that makes assurance difficult is that the evidence produced by some assurance strategies may not be what is most relevant for actually determining the safety of a system. Take for example hazard analysis – a common way of assessing the safety of a system [4]. Many hazard analyses consist of a series of steps to be performed at one or more stages in the system development lifecycle (e.g., preliminary design, detailed design, system test, etc.). The outcome of the analysis's steps is evidence that the system is either free from the types of safety issues that the technique is designed to detect or those issues have been mitigated to a point where residual risk is acceptable in light of the system's benefits. Some safety standards rely on this evidence for certification of a system's worthiness for a particular mission.

Rushby, however, has argued that the claims and arguments which rely on that evidence—and upon which those standards are built—are sometimes based on reasoning that is left implicit [18]. He continues by noting that while approaches based on standards as well as structured-argumentation like *safety cases* (also referred to more generally as *assurance cases*) both have their strengths and weaknesses, standards-based approaches are "slow-moving and conservative." This aligns, to some extent, with arguments made by Leveson, who writes that many hazard analyses and system safety standards are fundamentally outdated in their approach [12]. In Leveson's view, a common error is to evaluate a system's *reliability* and use that as a stand-in for the system's *safety*. This approach, she continues, was more valid when safety-critical systems were largely hardware based: individual component failures were much more likely to be the ultimate cause of a system failure before the addition of software significantly increased the variety of component configurations [10]. As a consequence, we undertake the following:

Our Goal: Assurance evidence should be contextualized, and that context should be explicit safety argumentation.

2.2 Assurance argumentation should be hierarchical

A second challenge to system assurance efforts stems from the competing goals of assurance documentation: it should be both easily understood yet completely accurate. System descriptions which are brief and abstract may be easily understood, but lack the technical precision and depth necessary to convey a complete understanding of a system. Complete and precise system descriptions, on the other hand, can take a significant amount of time to understand. This problem can be addressed, to an extent, through standardization: when assurance documentation is packaged in an expected format, familiarity with the standard can aide in understanding. But this leads to institutional inertia; i.e., Rushby's criticism of standards-based approaches as slow-moving [18] or Espinoza et al.'s more pointed criticism that standards can be a barrier to innovation [5].

We note that there is an interesting parallel here to another field where arguments are designed to convince human experts of their correctness: that of mathematical proof. In that domain, Lamport has argued for the utility of hierarchically structured, hypertext-enabled proofs [9]. An initial, high-level argument can be presented at an abstract level, but the reader can expand portions that are unclear as necessary; in this way technology and argument structure can be used to address the competing goals simultaneously. Therefore, we also undertake the following:

Our Goal: Assurance evidence should be initially presented at a high level, but a viewer should be able to expand argumentation as desired.

2.3 Assurance evidence should be modular and composable

A third challenge stems from the discrepancy between the way systems are built (compositionally, as aggregates of components) and how safety argumentation is structured (monolithically). That is, because critical systems are designed for operation in particular environment, safety argumentation rarely composes as easily as software or hardware elements. This mismatch can lead to inconsistency in guidance for assuring seemingly related systems [7] or system updates that are either postponed or avoided altogether to avoid the costs of (re)certification [5]. However, fully compositional safety is an enduring challenge because it requires successfully pursuing one of two very challenging strategies:

(1) *Anticipating the environment and role of a component:* This was the approach taken by ISO 26262's concept of a "Safety Element out of Context" [8], ISO 14971's concept of "Intended Use" [1]), and SAFE's concept of a component's role [15].
(2) *Completely describing all aspects of the component:* This description would have to analyze all possible uses in all possible contexts.

Recognizing that this goal may not be achievable, we nonetheless advocate its pursuit because progress towards it will still reduce the burden of creating assurance argumentation and speed the development of critical systems. Thus, our final objective is:

Our Goal: Assurance argumentation should be compositional.

3 TECHNICAL BACKGROUND

This section introduces the background of the modeling technologies, example, and safety methodologies used in the paper.

3.1 AADL and OSATE

The *Architecture Analysis and Design Language* (AADL) is an internationally standardized architecture description language [19]. AADL, which has both a textual and graphical syntax, is supported by the *Open Source Architecture Tool Environment*[1] (OSATE), which is a development environment based on the Eclipse IDE. System designers can use OSATE and AADL to model their system's software elements (e.g., `thread`, `process`, `subprogram`), hardware elements (e.g., `processor`, `memory`, `bus`), and the connections and bindings between them [6]. Annexes extend the core language to address different, non-architectural aspects of system design, such as behavior or data modeling.

One such extension that this work relies heavily upon is the error modeling annex [20], which includes mechanisms for specifying *error types*, which are errors that can be instantiated and propagated as tokens between components (similar to Wallace's Fault Propagation and Transformation Calculus [26]). For example, a sensor that produces a reading that may be higher than the actual value would be modeled as being a `source propagation` for tokens of the `Value High` error type. *Error propagations* represent the broadcast or reception of error tokens from/into components, typically via ports. This is used to represent a component producing erroneous output or receiving erroneous input, and can be used to trace the path that errors take through a system, e.g., a flawed sensor value is transformed by a software controller into an inappropriate command, which is transformed into a potentially unsafe actuation by a servo. The error modeling annex comes with a user-extensible library of error types, which is organized hierarchically into broad categories of error [16]. The type system is quite flexible, and can be used by a modeler to represent arbitrary error conditions, potentially including, e.g., the state of the system's environment.

3.2 An Example System: PulseOx Forwarding

To illustrate the features of ASAP, we use an illustrative, open-source[2] example from the medical domain. In addition to a control loop with multiple inputs and outputs, it has a single safety concern, which is the failure to issue a necessary alert. It consists of the following elements:

- **Hardware Devices** – Represented as AADL `devices`
  - *pulseOx:* A pulse oximeter device, which measures the blood oxygen saturation (SpO_2) of a patient via a non-invasive fingerclip.

```
1 package PulseOx_Forwarding_Logic
2 public
3   -- Import statements elided for space
4
5   process PulseOx_Logic_Process
6   features
7     LogicSpO2 : in data port
        ↪ PulseOx_Forwarding_Types::SpO2;
8     LogicDerivedAlarm : out event port
9     {MAP_Properties::Output_Rate => 200 ms .. 400
        ↪ ms;};
10  properties
11    MAP_Properties::Process_Type => logic;
12    MAP_Properties::Component_Type => controller;
13  annex EMV2 {**
14      use types PulseOx_Forwarding_Errors;
15      error propagations
16        LogicSpO2: in propagation {SpO2ValueHigh,
            ↪ SpO2ValueLow, EarlySpO2, LateSpO2,
            ↪ NoSpO2, ErraticSpO2};
17        LogicDerivedAlarm: out propagation
            ↪ {MissedAlarm, BogusAlarm};
18      end propagations;
19    **};
20  end PulseOx_Logic_Process;
21  process implementation PulseOx_Logic_Process.imp
22  subcomponents
23    CheckSpO2Thread : thread CheckSpO2Thread.imp;
24    SpO2Val : data PulseOx_Forwarding_Types::SpO2
25    {MAP_Error_Properties::Process_Variable => true;};
26  connections
27    outgoing_alarm : port CheckSpO2Thread.Alarm ->
        ↪ LogicDerivedAlarm;
28  end PulseOx_Logic_Process.imp;
29
30  thread CheckSpO2Thread
31  features
32    Alarm : out event port;
33  properties
34    Thread_Properties::Dispatch_Protocol => Periodic;
35  end CheckSpO2Thread;
36  thread implementation CheckSpO2Thread.imp
37  end CheckSpO2Thread.imp;
38
39 end PulseOx_Forwarding_Logic;
```

Listing 1: An example of AADL's textual syntax, showing the specification of a software controller.

[1] https://osate.org/

[2] https://github.com/osate/osate2-asap/tree/main/org.osate.asap.examples

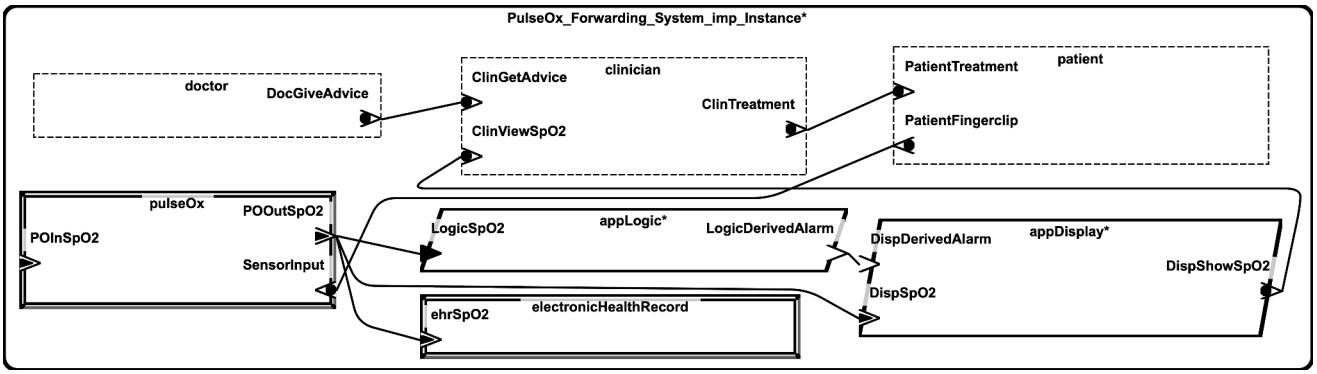


Figure 1: The PulseOx Forwarding System, in AADL's graphical syntax

  - *electronicHealthRecord*[3]*:* An adapter for the electronic health record, which records the patient's SpO_2.
- **Software Processes** – Represented as AADL `processes`
  - *appLogic:* Simple application logic that triggers an alert if the patient's SpO_2 is too low.
  - *appDisplay:* Simple logic to display both the alert (if present) and the patient's SpO_2.
- **Humans** – Represented as AADL `abstracts`
  - *clinician:* A clinician who monitors the display and treats the patient.
  - *doctor:* A doctor who advises the clinician but does not directly treat the patient.
  - *patient:* The patient who provides SpO_2 readings (via the PulseOx) and receives treatment.

Figure 1 shows the overall system in AADL's graphical syntax. This system is not complex, but has two aspects which make it ideal for demonstrating ASAP's features: it shows multiple *control loops*, i.e., circular paths through the system which involve both sensing and actuating; and some components (i.e., the `processes`) decompose cleanly into subcomponents (in this case a collection of communicating `threads`). Some of these components would be already be modeled as part of a normal system engineering process, while others (i.e., the humans) would typically need abstractions created specifically for safety analysis, e.g., to represent particular execution or interaction scenarios. Exactly which ones would need to be created for ASAP is impossible to specify without knowing what other analyses are being run on the system: modelers are typically encouraged to add only as much detail as required by the analyses they want to perform.

[3]We recognize that this is not how electronic health records are typically used, here we use the term for a more generic data store.

3.3 STPA and SAFE

The *System Theoretic Process Analysis* (STPA) is a hazard analysis that is designed to address many of the criticisms found with more reliability-oriented, hardware-focused analysis techniques [11, 13]. It has been adapted to work with AADL models without significant modification to the process [17].

The *Systematic Analysis of Faults and Errors* (SAFE) is a hazard analysis technique that is heavily derivative of STPA, but contains a number of modifications, including a formal definition of hazard [15]. This, along with additional specificity available with low-level architectural specifications such as AADL models, enables the new automation discussed in below. We discuss it, STPA, and SAFE in more depth as we explore ASAP in Section 4.

4 THE ARCHITECTURE-SUPPORTED AUDIT PROCESSOR

The Architecture-Supported Audit Processor (ASAP) is a plugin to OSATE that enables three "viewpoints" of a system. These viewpoints are diagrams and tables which are dynamically generated (i.e., in response to user input) and designed to support activities performed by system safety auditors. ASAP's viewpoints are generated from the system architecture as modeled in OSATE and supplementary safety information, entered by the system designer or analyst.

The first viewpoint is somewhat abstract and presents high-level *fundamental* aspects of the system (or component's) safety. The second presents elements in their immediate context, and the third focuses narrowly on the causes (and potential compensations) of errors within a component. This progression from abstract to specific is typical of both model-based system design in AADL and hazard analysis techniques such as SAFE and STPA. This progression means that more shallow analyses can be performed using less time: a simpler model and less ASAP-specific annotations will produce less rich diagrams and information. Alternatively,

different portions of the system can be modeled to different depths: a subcomponent whose behavior should be more deeply analyzed can be richly annotated while others are left essentially unspecified as "black boxes." This lets designers focus on the portions of the system that are relevant to particular stakeholders without getting bogged down in creating sophisticated models solely for the purpose of enabling tool functionality.

4.1 Viewpoint 1: Fundamentals

In order to orient the analysis towards particular safety issues, STPA and derivative analyses such as SAFE make explicit the links between low-level faults and errors and high-level safety concerns such as death or injury to a human. This is done by creating a *fundamentals hierarchy*, a structure that relates safety problems, and their solutions, to specific and general notions of accidents / losses. At the top of the hierarchy are *accident levels*, which are broad categories of harm that can be prioritized. A typical system might have death or injury to a human as the highest-ranked accident level, followed by damage to or destruction of mission equipment. *Accidents* are losses that can be caused by the system; any number of accidents can be linked to a single accident level. That is, there might be multiple specific accidents that would each result in harm to a human. A diagrammatic view of a fundamentals hierarchy is shown in Figure 2(a).

Linking concrete losses resulting from system failure (accidents) to the specific ways they occur (hazards) is a significant open challenge in designing architecturally-integrated safety analysis techniques. The difficulty comes in concisely specifying which system elements could be involved in causing a particular accident, and how the failure of those elements would cause the loss. Specifying the links between accidents and the system elements involved in their causation is necessary as these links form the context required to both understand the impact of system design choices and to construct coherent argumentation.

Hazards have a two-part definition in SAFE (which formalizes STPA's definition[4]) as a combination of one system and one environment state that will cause an accident; note that multiple hazards can lead to a single accident. Intuitively, this two-part definition results from the notion that certain system behaviors are rarely always unsafe, but rather only unsafe given a particular state of the environment. Leveson uses the example of a train: it is only unsafe for train's doors to be open while the train is moving [11]. In ASAP, hazards are modeled using the condition that causes them[5]; i.e., as the arrival of an error type at a component via its interface (e.g., a `port`), see Figure 2(b).

Figure 3 shows the *fundamentals* viewpoint in ASAP which supports STPA's first step. As shown by the lower portion, hazards contain a number of references to model elements including: a) *Accident:* The accident the hazard's occurrence would cause, b) *Environment Element:* The component whose state is the environmental "half" of the hazard, c) *System Element:* The component whose state is the system half of the hazard, d) *Error Type:* The AADL error type representing the system element's deviation from intended or acceptable behavior, and e) *Hazardous Factor* A human-readable name of what is being transmitted from the system element to the environment element[6]. Note that five of the nine elements linked to from the hazard (i.e., all those with icons other than) are semantic objects in either the AADL or ASAP model, as opposed to plain text. By semantic objects, we mean that these refer to actual elements in the model, represented in OSATE as rich data structures with links to other elements. This helps keep the documentation synchronized with the model, and enables the query-driven behavior of the other viewpoints.

4.2 Viewpoint 2: Connected Neighbors

In hierarchically-organized system models of any useful size, it can be difficult to understand how a particular component fits into the larger system. Even the relatively simple PulseOx Forwarding system can be difficult to quickly understand: there are multiple cyclic control flow paths as well as multiple levels of abstraction; we use model slicing to reduce the complexity [24]. AADL is purpose-built for hierarchically specifying system details.

STPA uses scoped control flow diagrams to present components in context in its second step, however, so we developed the *Connected Neighbors* viewpoint to show a given component, its immediate neighbors (i.e., those components that either produce input for the component or use its output), their neighbors, and any connections between the displayed components. AADL models contain all of this information already, the ASAP tooling extracts it automatically rather than requiring the diagrams to be constructed manually. Figure 4 shows an example of this viewpoint centering on the appLogic component of the PulseOx Forwarding system. Note that some elements, such as the `electronicHealthRecord` or `doctor`, are too distant[7] from the focused element, and thus are not displayed.

[4]STPA's full definition is "A system state or set of conditions that, together with a particular set of worst-case environmental conditions, will lead to an accident (loss)." [11, pg. 184]

[5]Note that in STPA, hazards can be either states or conditions (Leveson has discussed their equivalence [11]) but in ASAP they must be conditions.

[6]see, e.g., Ericson's text for the role of hazardous factors in accidents [4].

[7]Distance here is the number of "hops" from a given component, i.e., those which intransitively interfere according to van der Meyden's definition [25].

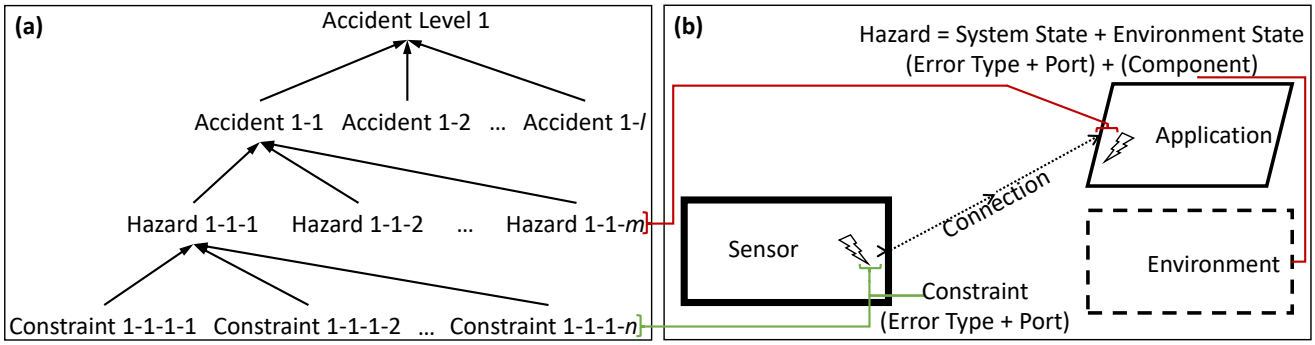


Figure 2: Part (a): A diagrammatic view of the fundamentals hierarchy. Part (b): How hazards and constraints are modeled in AADL.

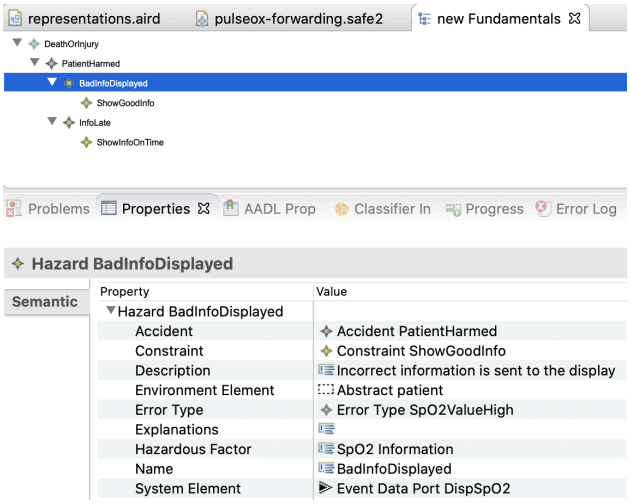


Figure 3: A fundamentals hierarchy (i.e., an instantiation of Figure 2a) in the ASAP tool.

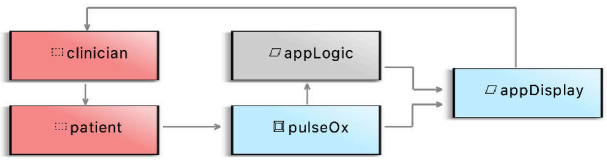


Figure 4: The Connected Neighbors view of the appLogic component in the ASAP tool. The primary element is shown in grey, immediate neighbors are blue, and the immediate neighbors of the neighbors are red. Connections represent the flow of data or commands. Note that this is a subgraph of Figure 1

Our goal with this view is not to replace system-level views like what AADL presents, but rather to support analyst intuition and rapid understanding of a particular component's "point of view" of a system. For example, any input from the doctor that affects the application logic would first have to be understood by the clinician, which would affect the treatment administered to a patient, which would be detected by the PulseOx, at which point it would be received by the appLogic. Put another way, the doctor's impact on the appLogic is mediated by three different system elements and thus may not be immediately relevant for gaining quick understanding, so the doctor does not appear as a neighbor of the appLogic in Figure 4.

4.3 Viewpoint 3: Unsafe Control Actions

ASAP's third viewpoint displays information on how system hazards might come to occur, and how they could be prevented.

4.3.1 Preliminaries: Causes, Compensations, and Guidewords. In any non-trivial system, there are a large number of ways that things can go wrong; in a safety-critical system these are documented as violations of the system's safety constraints. We refer to these violations as *causes*, i.e., ways that hazards (and associated losses) are caused. When thinking about a cause, the solution may or may not be apparent. If it is, a safety analyst should document it, we refer to these solutions (which may be partial, or conditioned on some other system behavior) as *compensations*. There is again a challenge in organizing and presenting a large amount of highly contextual information: a simple listing of causes and compensations is much harder to use than one organized, for example, around the system architecture or a taxonomy of system error types. These error types are equivalent to *guidewords* in hazard analysis techniques, which can be thought of as generic causes that are designed to prompt (i.e., guide) the thinking of analysts to consider various ways in which system elements might fail.

In addition to the manually-specified causes and compensations, however, a second form of loss scenario specification

emerges naturally from a fully specified EMV2 model of a system. Given a specification of how an error can come about (i.e., an `error source`), be transformed by various components (`error transformations`), and its final result (`error sink`), analysts can gain a fairly clear description of how a failure might occur. Because this form of scenario specification is machine-readable, tooling can also interpret these loss scenarios for various purposes such as building fault trees or calculating failure rates. This view is not present in ASAP, but it is available from other safety analyses in OSATE [2].

4.3.2 A Hierarchical Table. STPA's third step involves identifying control actions that could be unsafe. Typically, this identification is performed as analysts fill in a table where each row is a *control action* and there are four columns, one for each way STPA suggests a control action could be unsafe [13].

ASAP's version of this table is shown in Figure 5, note that there is an X for a connection (row) in the `ItemTimingError` (column). This denotes that documentation exists regarding the cause of a safety constraint violation involving the specified connection and the timing of messages being sent across it. A second table, from the same viewpoint can be generated that displays the same set of communication channels (i.e., rows) but the columns changed to show only the timing family of errors in the given component: here the full cause description (as well as optional compensation description) is shown. Identifying these scenarios where accidents / losses can occur is the fourth and final step of STPA.

The information required for these tables is pulled directly from both the system model (i.e., its error propagations), and the ASAP-specific fundamentals model, completing the deep integration of system model and hazard-analysis data. That is, there are two sources that are queried for each cell in the table: the "Fundamentals," which were created for ASAP's first viewpoint as well as the error propagations specified in the AADL model itself. If analyst-provided cause and / or compensation information is available, it is displayed; if only an error propagation indicates the presence of a problem, the text "Undocumented error propagation" is displayed instead.

Note that while the rows in the second, *refined* Unsafe Control Action table are the same as in the overview table, the columns can be errors from any error family, i.e., any of the abstract guidewords used in the model. While these column headings could be the guidewords from STPA, they could also be from AADL's Error Library (as in Figure 5) or any other set of guidewords / error collection. That is, while ASAP's *Unsafe Control Actions* viewpoint (i.e., the top-level table shown in Figure 5 and the refined version of the table) are usable for STPA's third and fourth steps, it has two enhancements.

First, the rows are not restricted to just control actions, but instead include all connections in the selected system / component. This change was made because some problems can be associated with non-control actions (sensor readings can be incorrect, electrical or hydraulic power can be over- or under-supplied, etc.) Recognizing and documenting these potential problems directly—instead of only when they manifest as unsafe control actions—is more concise. Additionally, distinguishing between a control action and sensor feedback is difficult to do consistently: it depends on the analyst's judgment and the component's role within the system.

Second, as ASAP is not directly tied to any particular hazard analysis process, the columns are generic: they are derived from the top-level error types used in the AADL model, rather than being unchangeable. An analyst could certainly choose to use STPA's set of guidewords, or they could use error types from the AADL Error Library, a custom set, or one derived from / aligned with a particular safety standard required by their company or the domain they are working in. Several such sets of system errors exist; Procter and Feiler have described AADL's Error Library and its relationship to guidewords used in hazard analyses [16].

4.3.3 A Hierarchical Taxonomy. Building a taxonomy of guidewords is rarely the primary goal of a research effort, however; typically guidewords are created as part of developing a new hazard analysis method. It is a challenge, though, to balance the goals of being both sufficiently expressive (so analysts do not miss potential causes) while also not being overly prescriptive (which can make the analysis unwieldy and verbose). In ASAP, we addressed this problem by defining two levels of tables which can be generated for every component, but this relies on a hierarchical organization of guidewords / error types.

The two-level approach taken by ASAP relies on a hierarchical specification of system error, i.e., a generic error type must be refinable into a set of more specific error types. AADL's EMV2 supports just such an approach [20]; using it, system modelers or safety analysts can define custom error types, and then refine those into more specific types. Alternatively, the EMV2 standard comes with a predefined Error Library, which contains a straightforward decomposition of standard system errors [16]. Typically, users combine the two approaches: they begin with the Error Library's set of error types and refine those to align with their domain or system. These custom error types are supported by the Unsafe Control Actions tables, so domain-specific extensions to the model will be fully incorporated. See Figure 6, which shows a graphical view of the EMV2 library's hierarchy of timing-related errors, extended with custom error types specific to the PulseOx Forwarding application described in Section 3.2.

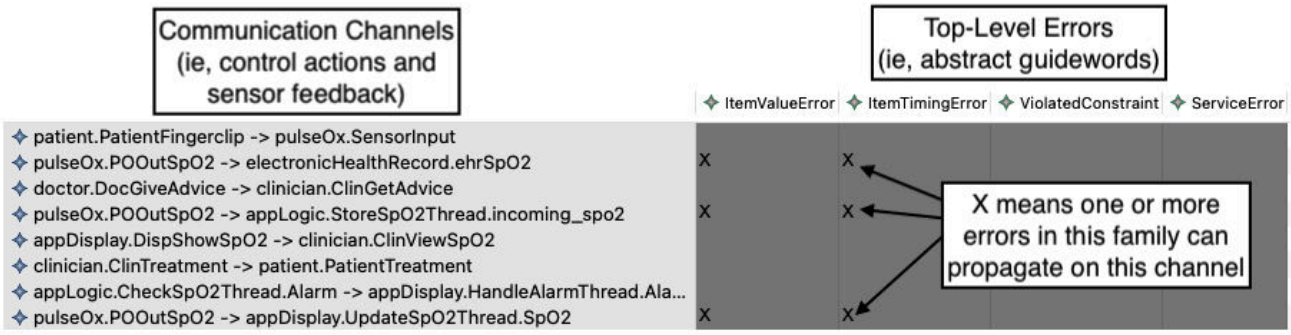


Figure 5: Unsafe Control Actions table generated on the PulseOx Forwarding System.

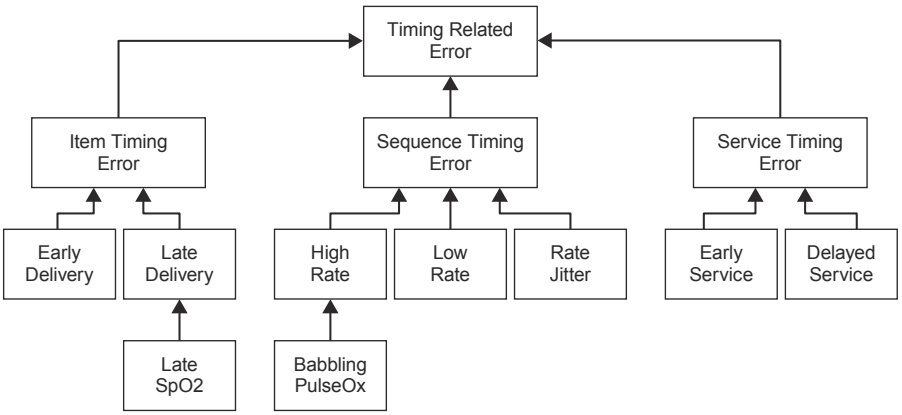


Figure 6: Hierarchy of timing errors, adapted from [16].

4.4 Tying it together: Focus

A common feature in diagrams used for safety analysis is a way to highlight a particular component or fundamental as well as those related to it, i.e., some way to call attention to system elements related to a specific hazard, accident, component, error etc. For fundamentals, this task is straightforward: we simply highlight the higher-level fundamentals which contain the focus target (i.e., move up the tree from Figure 2(a)) as well as all the lower-level fundamentals it contains (i.e., the subtree rooted at the focus target). Handling a focused Hazard or Constraint requires additional effort, though: recall that those fundamentals contain references to the system model, i.e., to error propagations occurring at specific component ports. Thus we include predecessors and successors, which are the components that might cause or be affected by the associated error propagation.

Thus, focusing on an error propagation or system component is significantly more complex than focusing on an Accident or Accident Level. We note that there is a related challenge in static analysis of software: it is often necessary to determine which program statements could have affected, or have been affected by, the program's state at a specific point in the source code. This issue is typically addressed through the use of program *slicing*, which Silva describes as "a technique for decomposing programs by analyzing their data and control flow" [23]. There are a number of program slicers available, we use one that has been built to run on AADL models [24].

A *backward* slice finds system elements (or error propagations) which could potentially affect the focused system element (or cause the focused error propagation). Correspondingly, a *forward* slice finds system elements (or error propagations) which could potentially be affected by the focused system element (or have been caused by the focused error propagation).

4.5 Discussion

A safety analyst who wishes to use ASAP today, i.e., given existing regulatory regimes, will find it most useful to use

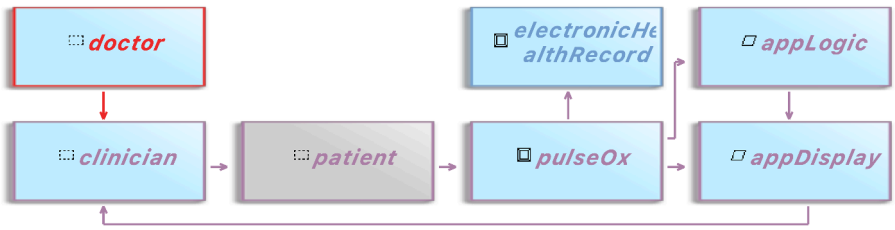


Figure 7: The Connected Neighbors view of the Patient, which has been selected as the current focus. Red elements are reachable from the focused element only via a backwards traversal. Blue elements are reachable only via a forwards traversal. Purple elements are reachable via both.

the tool while developing the argument for certification authorities. They might begin by specifying metadata about her system using Viewpoint 1, check the inputs and outputs of a high-criticality component using Viewpoint 2, and then explore undocumented causes and compensations using AADL's EMV2 and Viewpoint 3's tables. In the longer term, we envision safety processes that support the interactive, query- / view-driven approach explored in this work directly.

5 RELATED WORK

There is a large body of work that considers how the creation of systems and safety documentation should be automated. We differentiate the approaches into two groups, those whose automation is independent of a system's hierarchical decomposition and those who integrate with it more deeply. We discuss these groups with representative technologies.

5.1 Automated Assurance Cases

AdvoCATE [3] is software developed by NASA that brings considerable automation to the creation of *assurance cases*, i.e., structured arguments that follow a defined, logical format. They incorporate both arguments and evidence, but are typically not as deeply integrated into a system's architecture as the arguments in ASAP. The methods of argument traversal are, however, similar to ASAP in that AdvoCATE supports, e.g., hierarchical abstraction and queries / views. However, these are queries and views of the argument itself rather than the system under analysis as was our goal.

5.2 Hierarchical Safety Analysis

Hierarchically Performed Hazard Origin and Propagation Studies (HiP-HOPS) [22] takes into account system hierarchy and supports compositionality. Compared to ASAP, it uses a more traditional (i.e., not system-theoretic) model of safety and accident causality, and it operates on systems in MATLAB rather than AADL / OSATE. HiP-HOPS' primary goal is the generation of hazard analysis reports, rather than deeply integrating safety information and argumentation into a system's architecture.

6 FUTURE WORK

6.1 Autogenerating Causes and Impacts

As discussed in Section 4.3 a fully-specified EMV2 description of a system encodes a machine-readable error path / event chain. In addition to the analyst-supplied narrative now visible in the unsafe control actions table, we are interested in transforming these event chains into something human readable. These causal event chains can be calculated using AWAS's backward slicing functionality [24] (starting from either a constraint violation or an arbitrary error occurrence), though how these chains are best displayed to the user remains an open question.

AWAS's forward slice calculates the errors and failures resulting from an error's occurrence. These impacts could also be useful, though work would need to be done to align them with existing safety notions from, e.g., academic literature and / or safety standards. Once aligned, the best format for their presentation to the user would also need to be determined.

6.2 Alignment with Requirement Specifications

The goals and activities involved in safety engineering overlap to some extent with those involved in specifying a system's requirements. We expect requirement specification may evolve, somewhat, with the use of ASAP and are interested in exploring ways of supporting and automating more rigorous requirement specifications. However, we recognize that specifying functional and safety requirements simultaneously is not a straightforward task. There is research in this area from both system theoretic safety [21] and architecture-centric perspectives [14]; we are interested

in seeing the extent to which ASAP's viewpoints can be extended or supplemented with additional requirement detail or traceability information.

7 CONCLUSION

In this paper we presented the Architecture-Supported Audit Processor, or ASAP, tool as well as its motivation and underlying theory. We applied it to a small example, demonstrated how it aligns with and improves upon a popular system-theoretic hazard analysis, and discussed possible avenues for future improvements.

ACKNOWLEDGEMENTS

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

REFERENCES

[1] Association for the Advancement of Medical Instrumentation. 2000. *ANSI/AAMI/ISO 14971: Medical devices—Application of risk management to medical devices.* Technical Report. ANSI/AAMI/ISO.

[2] Julien Delange and Peter Feiler. 2014. Architecture Fault Modeling with the AADL Error-Model Annex. In *2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications*. IEEE, Verona, Italy, 361–368.

[3] Ewen Denney and Ganesh Pai. 2018. Tool support for assurance case development. *Automated Software Engineering* 25, 3 (2018), 435–499.

[4] Clifton A. Ericson II. 2016. *Hazard Analysis Techniques for System Safety* (second ed.). John Wiley & Sons, Inc., Fredericksburg, Virginia, United States of America. 1–640 pages.

[5] Huáscar Espinoza, Alejandra Ruiz, Mehrdad Sabetzadeh, and Paolo Panaroni. 2011. Challenges for an open and evolutionary approach to safety assurance and certification of safety-critical systems. In *Proceedings - WoSoCER 2011 - In Conjunction with ISSRE 2011*. IEEE, Hiroshima, Japan, 1–6.

[6] Peter Feiler and David Gluch. 2012. *Model-Based Engineering with AADL* (1st ed.). Addison-Wesley Professional, Upper Saddle River, NJ. i–468 pages.

[7] Brendan Hall, Jan Fiedor, and Yogananda Jeppu. 2020. Model Integrated Decomposition and Assisted Specification (MIDAS). *INCOSE International Symposium* 30, 1 (jul 2020), 821–841.

[8] International Organization for Standardization. 2018. *ISO 26262-10: Road vehicles—Functional safety—Part 10: Guidelines on ISO 26262.* Technical Report.

[9] Leslie Lamport. 2012. How to write a 21 st century proof. *Journal of Fixed Point Theory and Applications* 11, 1 (mar 2012), 43–63.

[10] Nancy Leveson. 1995. *Safeware: System Safety and Computers.* Addison-Wesley.

[11] Nancy Leveson. 2011. *Engineering a Safer World: Systems Thinking Applied to Safety.* MIT Press.

[12] Nancy Leveson. 2020. Are you sure your software will not kill anyone? *Commun. ACM* 63, 2 (jan 2020), 25–28.

[13] Nancy Leveson and John Thomas. 2018. *STPA Handbook.* Technical Report. 1–188 pages.

[14] Anitha Murugesan, Sanjai Rayadurgam, and Mats Heimdahl. 2019. Requirements reference models revisited: Accommodating hierarchy in system design. In *Proceedings of the IEEE International Conference on Requirements Engineering*. 177–186.

[15] Sam Procter. 2016. *A Development and Assurance Process for Medical Application Platform Apps.* Ph.D. Dissertation. Kansas State University.

[16] Sam Procter and Peter Feiler. 2018. The AADL Error Library : An Operationalized Taxonomy of System Errors. In *HILT 2018*. Boston, MA.

[17] S. Procter and J. Hatcliff. 2014. An architecturally-integrated, systems-based hazard analysis for medical applications. In *MEMOCODE 2014*.

[18] John Rushby. 2010. Formalism in Safety Cases. In *Making Systems Safer: Proceedings of the Eighteenth Safety-Critical Systems Symposium*. Springer, London, Bristol, UK, 3–17.

[19] SAE AS-2C Architecture Analysis and Design Language Issuing Committee. 2017. *Architecture Analysis & Design Language (AADL).*

[20] SAE AS-2C Architecture Description Language Subcommittee. 2015. *SAE Architecture Analysis and Design Language (AADL) Annex Volume 1: Annex E: Error Model Language.* Technical Report. SAE Aerospace.

[21] Andrea Scarinci, Amanda Quilici, Danilo Ribeiro, Felipe Oliveira, Daniel Patrick, and Nancy Leveson. 2019. Requirement Generation for Highly Integrated Aircraft Systems Through STPA: An Application. *Journal of Aerospace Information Systems* 16, 1 (jan 2019), 9–21.

[22] Septavera Sharvia and Yiannis Papadopoulos. 2015. Integrating model checking with HiP-HOPS in model-based safety analysis. *Reliability Engineering & System Safety* 135 (2015), 64–80.

[23] Josep Silva. 2012. A Vocabulary of Program Slicing-Based Techniques. *Comput. Surveys* 44, 3, Article 12 (June 2012), 41 pages.

[24] Hariharan Thiagarajan, John Hatcliff, and Robby. 2020. Awas: AADL Information Flow and Error Propagation Analysis Framework. In *European Conference on Software Architecture (ECSA20)*. Springer, Cham, L'Aquila, Italy, 294–310.

[25] Ron Van Der Meyden. 2007. What, Indeed, Is Intransitive Noninterference?. In *Proceedings of ESORICS 2007: 12th European Symposium On Research In Computer Security*. Springer Berlin Heidelberg, Dresden, Germany, 235—-250.

[26] Malcolm Wallace. 2005. Modular Architectural Representation and Analysis of Fault Propagation and Transformation. In *FESCA 2005*, Vol. 141. 53–71.

Automated Generation of Requirements for the Highly Fault-Tolerant System Behaviour of a Distributed and Integrated Avionics Platform

Robert Wipperfürth*†, Thorben Hoffmann*, Christoph Kurz*, Tim Belschner* and Reinhard Reichel*
* *Institute of Aircraft Systems*
University of Stuttgart
Stuttgart, Germany
† robert.wipperfuerth@ils.uni-stuttgart.de

***Abstract*—Fully autonomous Unmanned Aerial Vehicles, Remotely Piloted Aircraft, Air Taxis, as well as advanced CS-23 aircraft require numerous complex and safety-critical system functions, such as vehicle management and utility functions, automatic take-off and landing or flight control. The development and qualification of the related avionics systems are characterised by a very high effort. The Institute of Aircraft Systems at the University of Stuttgart, in close cooperation with Aviotech GmbH, aims at a highly automated development and verification process for such fault-tolerant avionics systems to significantly reduce development effort, time, and risk and thus costs. For this reason, the Flexible Avionics Platform was developed. It enables the implementation of integrated fly-by-wire platform instances and is characterised by the following key aspects. (1) A platform-based development approach featuring an integrated, distributed, and highly redundant avionics architecture. (2) The platform management, a high-level abstraction layer providing a full abstraction towards integrated applications regarding the distribution, fault-tolerance, and redundancy of a fly-by-wire platform instance including redundant peripherals. (3) The AAA process, a comprehensive automation process for the highly automated generation of development and qualification artefacts, such as an instance of the Platform Management, the corresponding specification at the system and software level, and related test cases and test scripts. This paper presents the basics for the automated requirements generation at the system level with a focus on the specification of the highly fault-tolerant system behaviour of fly-by-wire platform instances based on the Flexible Avionics Platform.**

***Index Terms*—Flexible Avionics Platform, AAA process, platform-based development, model-driven development, automatic requirements instantiation, requirements reuse**

I. Introduction

A. Motivation

Future aircraft require numerous complex and safety-critical system control functions. This is especially true for fully autonomous Unmanned Aerial Vehicles (UAVs) and Remotely Piloted Aircrafts (RPAs) that are certified and operated in non-segregated airspace. Such aircraft must be able to react in a semi- or fully-automated manner to all critical situations as, for instance, the loss of the command and control data link or the loss of an engine. Furthermore, standard operational functions like take-off and landing or the entire vehicle management and utility functions must be automated as well.

Accordingly, the implementation of the related avionics systems is characterised by a very high effort. An implementation based on avionics platforms with highly integrated architectures is a first step to reduce system costs. While this approach enables the integration of a high number of systems within a single instance of such an avionics platform, adequate safety, up to failure rates of $10^{-9}/h$, has to be ensured. To enable an affordable usage of such integrated avionics platforms, especially within the CS-23 domain, a new approach is required that reduces development and qualification effort significantly while maintaining a high safety level.

B. State of the Art

In the avionics domain, Integrated Modular Avionics (IMA) represents the state of the art regarding platform-based development approaches as well as integrated and distributed architectures [1], [2], [3]. The hardware is based on standardised modules connected via an Avionics Data Communication Network (AFDX®). Concerning the software architecture, the hardware and communication are abstracted towards integrated applications (see Fig. 1). This includes inter-partition communication, even if the partitions are allocated to different hardware modules. There are comprehensive automation approaches for the automated instantiation of these IMA abstraction layers [4], [5], [6], [7].

Due to these features, IMA represents a significant advance in the realisation of avionics systems, their development, and their qualification. However, despite all process automation approaches, the development and qualification of avionics systems based on IMA still require a high effort. The reason for this is the low abstraction level provided by IMA. IMA does not provide an abstraction of system management aspects such as the management of redundant sensors, redundant actuators, and the failure and redundancy management of distributed modules ensuring consistency in a distributed architecture. Thus, this extremely complex part of the system management, especially with highly redundant avionics systems, has to be

This research was funded by the German Federal Ministry for Economic Affairs and Energy (BMWi) within the LuFo V-2 and LuFo V-3 program.

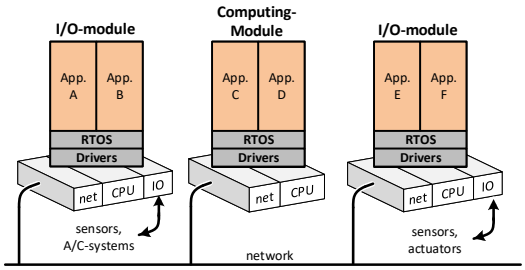


Fig. 1. Integrated Modular Avionics architecture.

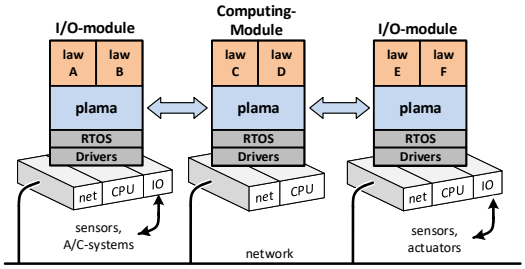


Fig. 2. Flexible Avionics Platform architecture.

implemented as part of the integrated applications. Consequently, the most challenging tasks of the system management are not part of the IMA development automation. This is the key difference to our approach of the Flexible Avionics Platform with its associated process automation.

C. The Flexible Avionics Platform [8]

The Flexible Avionics Platform (apf) is a platform-based development approach in the sense of Di Natale et al. [9] and features an integrated, distributed, and highly redundant architecture. It consists of standardised hardware components as well as a library of generic software components, i.e. a Real Time Operating System (RTOS), drivers, and small software bricks called basic services. Each specific avionics system is generated as an instance of this Flexible Avionics Platform.

The apf is characterised by a high-level abstraction layer realised as a middleware, the so-called Platform Management (plama) (see Fig. 2). It covers the entire system management for all distributed and redundant modules (internal and external) as well as redundant peripherals, for example, redundant sensors and redundant actuators. Thus, it manages tasks such as fault-tolerant intra-module and inter-module communication, communication to other systems, the redundancy of all modules and peripherals, and reconfigurations in the event of failure. Furthermore, it controls the system-wide operation phases such as *normal operation*, *pre-flight built-in test*, and *interactive system operation* during maintenance. These system management tasks are handled by plama in a distributed manner and are executed on all modules of a Flexible Avionics Platform Instance (apfi). Each plama instance is implemented as a composition of specialised basic services.

Due to the high degree of abstraction provided by plama, integrated applications are executed in a failure-free virtual simplex environment and are not required to perform any system management tasks. Hence, applications are reduced to their cybernetic control law.

D. The AAA Process

The apf is complemented by a comprehensive automation process for the development and qualification of an apfi, the AAA process. It consists of the following subprocesses:

- Axx subprocess – Automated design and parameter instantiation [10]: Based on a high-level system specification a high-level system design model is implemented manually. It is expressed with a domain-specific modelling language developed by the Institute of Aircraft Systems (ILS). This model describes the basic hardware structure, connected sensors and actuators, interfaces to other systems, a placeholder for the applications, the degree of redundancy, the basic reconfiguration strategy, and the scheduling of the apfi-wide operation phases.
  In a first automation step, the implemented system design model is refined by synthesis rules into a software architecture model containing the key software components and their coupling. In a subsequent automation step, the software architecture model is further refined into a model of the software components' Parameter Data Items (PDIs). The PDI model defines the selection of suitable basic service, their functional specialisation, and their data and control coupling. In addition, it defines the configuration data for all drivers and the RTOS. As a last step, this model is automatically transformed into source code. This PDIs source code together with the source code of the selected basic services, drivers, and the RTOS as well as the hardware modules form an apfi.
- xAx subprocess – Automated document generation [11]: The approach of the automatic generation of requirements is based on requirement classes describing characteristic behaviour that can be realised with the apf. Requirement classes are modelled once in terms of characteristic patterns, defining aspects such as their instantiation condition, representation, and the structure of possible instances. The corresponding models are automatically transferred into synthesis rules. Executing these synthesis rules, the xAx tool suite automatically analyses the apfi design models, generated in the Axx subprocess, with regard to the characteristic patterns of each requirement class. The match of a single or a group of patterns leads to the automatic generation of a requirement instance, i.e. the specialisation of the related requirement class with the apfi specific information of this match.
  In this way, all apfi requirements are generated automatically for system-level, software high-level and software low-level. In general, a requirement instance is expressed in the form of a natural language representation and a formal representation. The natural language representations are human-readable descriptions of the requirements and include information on traceability and versioning.

They conform to the relevant aeronautical standards ARP4754A and DO-178C and represent a large part of the entire apfi specification documents that are used for an apfi certification process. The formal representations feature corresponding and consistent models of the apfi behaviour and are further processed by the xxA subprocess.

- xxA subprocess – Automated generation of verification artefacts [12], [13]: Based on the formal representations the xxA tool suite automatically generates associated test cases and test procedures respectively test scripts using a dedicated test oracle. During the integration and verification processes for an apfi, these test scripts can be automatically executed on an apfi-in-the-loop testrig.

E. Automated Requirements Generation for Plama's High-Level Management Layer

A first approach for the automated generation of requirements focused on plama's communication layer (low-level management) which manages the fault-tolerant intra-module and inter-module communication as well as the monitoring and data fusion [11].

This paper presents the enhancement of the approach for the system specification of the high-level management layer of plama, i.e. the layer managing reconfigurations and the operation moding of an apfi. The strategy for an automated generation of requirements is based on the following considerations. The automatically generated instances of the requirement classes have to enable a meaningful apfi specification that can be used for validation processes and that is comparable to a manual specification. In addition, the set of existing requirement classes has to cover the entire usage domain of the apf. Especially at the system level, these requirement classes should describe the main characteristics. Furthermore, the effort considering the manual implementation of requirement classes must be justifiable.

For this reason, the behaviour of an apfi at the system level must be characterised by a limited number of generic requirement classes valid for all apfis. The central question resulting from this is how these requirement classes are to be defined. This is the focus of our article.

F. Paper Structure

First, we set the context by showing central architectural and operational aspects of an apfi in section II. While section III analyses what must be specified for an apfi at the system level, section IV elaborates on how the system-level behaviour is specified using requirement classes. Finally, section V summarises the key results and provides an outlook on our ongoing and future research.

II. Flexible Avionics Platform

The section is intended to clarify the framework of the apf. For this purpose, the architecture of the apf is presented and important operation principles are explained.

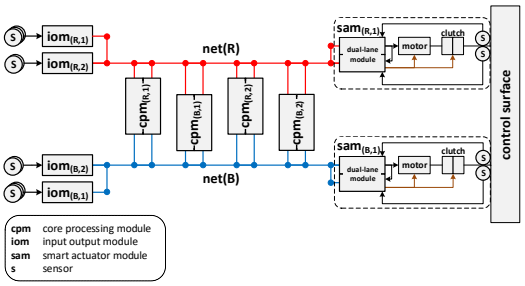


Fig. 3. Hardware structure of an exemplary instance of the apf.

A. Architecture

Figure 3 shows the hardware structure of an exemplary instance of the apf consisting of input output modules (ioms), core processing modules (cpms), smart actuator modules (sams), as well as the networks (net). An iom provides necessary input/output interfaces to connect system-specific peripheral units such as sensors, actuators, human-machine interfaces, or other systems. Measured Signals are preprocessed and transmitted to the cpms. A cpm is a dual-lane module. Each lane operates cyclically and all tasks are executed in parallel and frame synchronously with the other lane. A cpm features a fail/passive behaviour. This is realised by the cross-comparison mechanisms of plama, which ensure the required consensus [14] between the lanes. Cpms have two main tasks. First, they plan and execute the central management decisions within an apfi. Second, they execute the laws of integrated applications, i.e. integrated system function laws. Control commands for the operation of the control surfaces of an aircraft are transmitted to the sams. A sam operates and monitors a control surface. It features a dual-lane architecture including the power electronics for an electric actuator with an optional clutch. All modules communicate via two independent networks, which can provide further internal redundancy. As shown, each module is logically assigned to one network side red $S_{(R)}$ or blue $S_{(B)}$. For example, $cpm_{(R,1)}$ with module ID 1 is assigned to network side $S_{(R)}$.

B. Apfi Operation

The apfi including connected peripherals and all integrated system functions must operate correctly in numerous operation phases as well as numerous apfi configurations. Here, an apfi configuration describes the state of an apfi regarding the dynamic configuration of the resources taking part in the apfi operation. It is not to be confused with the PDI generated by the Axx tool suite. If the apfi operation is affected by a failure, apfi reconfiguration measures are required. Likewise, the operation phase must be adjusted if the apfi operation conditions change. The decision making regarding such actions is designed strictly hierarchically [15], [16]. The corresponding decision hierarchy is as follows:

1) Determination of the membership (membership determination).
2) Allocation of multi-application (mapp allocation).

3) Allocation of the master-slave engagement (master-slave allocation).
4) Scheduling of the operation phases (operation moding).

In general all apfi modules, as well as the connected peripheral units, are considered within this hierarchy. However, the following sections focus on the apfi core which comprises all cpms of the distributed apfi.

1) Membership Determination: The basis for consistent distributed decisions of the cpms is the so-called membership. It ensures that only non-faulty cpms participate in the apfi operation.

The membership determination is a two-step process. First, each cpm determines the membership of its own module locally (module membership). Based on this, each cpm determines the membership of the other cpms (inter-cpm membership). For the inter-cpm membership consensus between all correct cpms is required.

2) Mapp Allocation: The apfi features a dynamic in-flight reallocation of integrated system function laws (sfls) between cpms thus allowing for an efficient use of the cpm hardware. For this, several sfl of the same criticality level are grouped to a multi-application (mapp). Mapps are prioritised according to their criticality. For example, $mapp_{(1)}$ contains the absolute safety-critical system functions *flight control*, *braking*, and *steering* while $mapp_{(2)}$, with the lower priority, comprises functions such as *flight guidance* or *flight management*. Although each cpm has loaded the software for all mapps, it executes only a single mapp at a time.

The assignment of which cpm executes which mapp is called mapp allocation. Mapps are allocated only to such cpms for which membership is given. The mapp allocation is based on apf generic mapp allocation rules. If the rules are met, the mapp allocation is correct. If the rules are violated, the corresponding incorrect mapp allocation is changed to a new, correct mapp allocation. This mapp reconfiguration may take place in flight. For the mapp allocation, consensus is required between all cpms with a given membership.

3) Master-Slave Allocation: If the same mapp is allocated to multiple cpms, i.e. if there are multiple replicas of a mapp, an active-hot-standby replication strategy is applied. Only data provided by the active mapp-replica, i.e. the mapp engaged as master, is processed by other mapps, ioms, or sams. The data provided by the hot-standby mapp-replica, i.e. the mapp engaged as slave, is transmitted via the network but is ignored by all recipients[1]. If there are further mapp-replicas, these are operated as cold-standby replicas. In our article, this so-called shadow engagement is not considered any further.

The selection of which mapp-replica is to be operated as master or slave is called master-slave allocation. As with the mapp allocation, the master-slave allocation is also based on an apf generic ruleset. These rules ensure that the mapp-replica with the best performance execution level[2], regarding the implemented sfls of a mapp, is selected as the master. For the master-slave allocation, consensus between all replicas of a mapp is required.

4) Operation Moding: During a mission, the system of integrated systems, i.e. the apfi with all its integrated system functions including connected peripheral units, undergoes various operation phases. For each operation phase, the required behaviour can change. Examples are the normal operation during flight, the execution of all test steps of a pre-flight built-in test before dispatch, or the interactive activation of built-in tests during maintenance.

Each operation phase is associated with an apfi specific set of hierarchically structured operation modes that can be specified in the high-level system design model for the Axx subprocess. These operation modes define the behaviour of the entire system of integrated systems. The scheduling of the operation modes, which mainly depends on the apfi operation conditions, is performed centrally by the master replica of $mapp_{(1)}$ ($mapp_{(1,ma)}$). Other replicas of $mapp_{(1)}$ simply adopt the current operation modes determined by $mapp_{(1,ma)}$ which ensures consensus.

5) Examplary Apfi Reconfiguration: To provide an insight into the apfi operation, a failure induced mapp and master-slave reconfiguration is presented exemplarily. The initial state is shown in Fig. 4(a). The module membership of each cpm, as well as the inter-cpm membership on all other cpms, is set to ON. Thus, membership is given for all cpms. The shown mapp and master-slave allocation is correct.

Starting from this state, there is a failure in one lane of $cpm_{(R,1)}$. Due to the dual-lane architecture, this failure is detected by the other correct lane of $cpm_{(R,1)}$, which then passivates the cpm. To do this, the correct lane changes its module membership to OFF and stops the transmission of messages over the network.

The passivation of $cpm_{(R,1)}$ which executed $mapp_{(1,ma)}$ affects the other cpms of the apfi. They cannot communicate with $cpm_{(R,1)}$ any more. Hence, the remaining correct cpms consistently set the inter-cpm membership about $cpm_{(R,1)}$ to OFF. However, the mapp and master-slave allocation of the cpms with given membership violate the allocation rules:

1) Although the priority of $mapp_{(1)}$ is higher than the priority of $mapp_{(2)}$, it is allocated to only one cpm.
2) There is no master replica of $mapp_{(1)}$.

Consequently, the mapp allocation and the master-slave allocation have to be reconfigured. In a first step, $cpm_{(B,1)}$ reconfigures from $mapp_{(1,sl)}$ to $mapp_{(1,ma)}$. In a second step, $cpm_{(R,2)}$ reconfigures from $mapp_{(2,sl)}$ to $mapp_{(1,sl)}$. The state illustrated in Fig. 4(b) represents the resulting, correct allocation. Since the apfi operation condition did not change, the new $mapp_{(1,ma)}$ maintains the operation modes adopted from the previous master-replica.

III. Overview of the Specification of an Apfi at System Level

This section aims to illustrate what must be specified at the system level. For this purpose, the classification of the

[1]If the master-slave allocation changes, this enables a transient-free usage of data from the new master by the recipients.

[2]The performance level accounts for the state of sensors and actuators being part of the execution of the system functions.

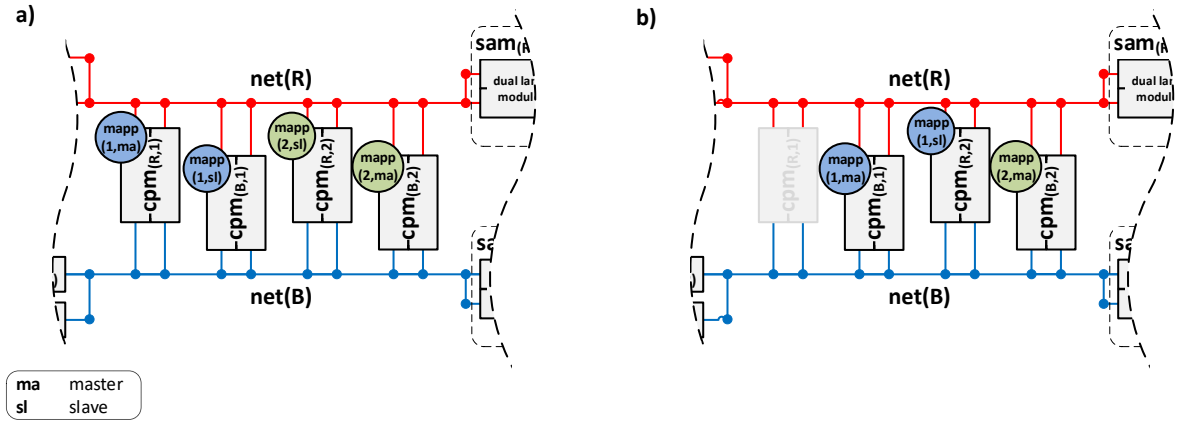


Fig. 4. Mapp and master-slave reconfiguration due to a failure in $cpm_{(R,1)}$.

specification is explained and details on each specification domain are provided.

A. Basics

Most of the time, the apfi operates in a steady-state. A steady-state is characterised by a steady apfi configuration and a steady operation phase. Hence, all allocation rules are met and the required behaviour of the system of integrated systems is stationary. However, failures or changes in the aircraft operation condition can lead to the transition of one steady-state to another. Based on this, the apfi specification at the system level is divided into the following specification domains:

1) The specification of the steady-state operation.
2) The specification of transitions between steady-states.

B. Specification of the Steady-State Operation

The requirements of this domain specify the correct operation of an apfi for all steady-states. A correct operation of a steady-state is characterised by

- the data flow,
- the operation features.

Considering the data flow, data required for the execution of a system function must be correctly transferred from the input of the apfi (e.g. sensors) to the application programming interface of the sfls integrated on each cpm. Furthermore, data must be transferred between sfls and from an sfl to the output of the apfi (e.g. actuators).

Regarding the operation, key features such as the cyclic operation, synchronicity, and consensus, have to be ensured.

C. Specification of Transitions between Steady-States

This specification domain defines the system-level transitions from one steady-state to a new steady-state, due to failures or changes in the aircraft operation condition. Thus, the requirements of this domain specify the reconfigurations and operation moding of an apfi at the system level. Based on the decision hierarchy presented in section II-B, the specification domain is further divided into subdomains.

- The intra-module subdomain specifies the reconfiguration behaviour of single modules.
- The inter-module subdomain specifies the reconfiguration of sets of modules. This includes reconfigurations of the inter-cpm membership, the mapp allocation, or the master-slave allocation.
- The peripheral subdomain specifies reconfigurations affecting the operation of sensors and actuators as part of the execution of system functions.
- The operation moding subdomain specifies the scheduling of operation modes for the system of integrated systems.

For each of these specification subdomains, there is a dedicated set of requirement classes, which together cover all possible aspects of that subdomain. This facilitates a focused and expressive specification of the related behaviour. The sum of all requirement classes covers the overall reconfiguration and operation moding behaviour for all possible apfis. In the following, exemplary requirement classes for an apfi reconfiguration are presented.

IV. Requirement Classes at System Level

This section exemplarily describes how the requirement classes of apfi reconfigurations at the system level are defined. For this, the basic reconfiguration principle of the apf is shown. This is followed by exemplary requirement classes covering different aspects of a reconfiguration.

A. Basic Reconfiguration Principle

Figure 5 illustrates the basic reconfiguration principle valid for each apfi. An apfi features a large number of possible failure events such as a power interruption, CPU failures, or sensor failures. If a failure event ev_i is monitored by a dedicated failure detection mechanism, the corresponding particular indication $z_{I,i}$ is set. Particular indications are then fused to so-called categorical indications $z_{CI,x}$ which are confirmed in time. While the events and the particular indications are mostly apfi specific, the set of possible categorical indications is generic for the apf. Because of the large usage

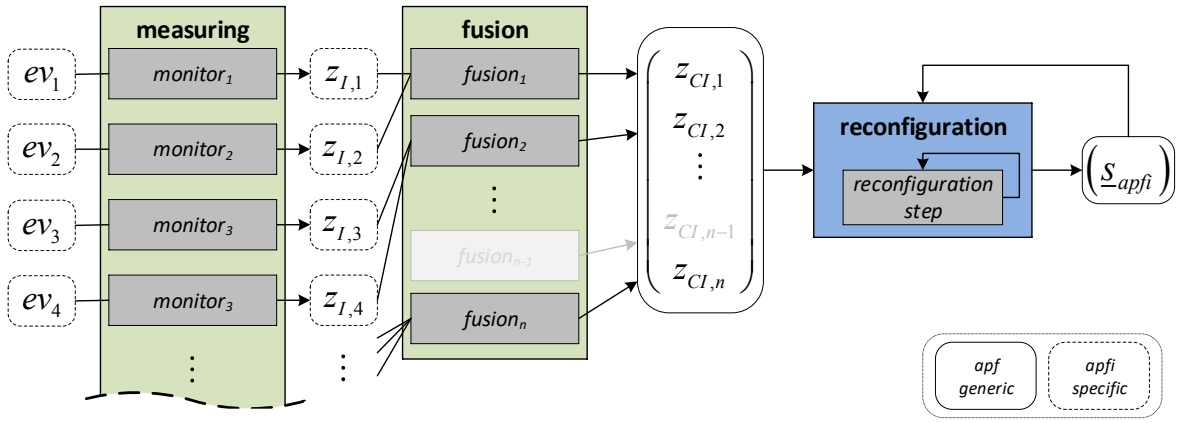


Fig. 5. Illustration of the basic reconfiguration principle of the apf.

domain of the apf, in general, only a subset of all possible categorical indications is needed for a specific apfi[3].

Necessary reconfiguration steps for the current apfi configuration $\underline{s}_{apfi}$ are planned exclusively based on these categorical indications. The steps are executed over a short period until the overall reconfiguration is completed, i.e. until a new steady-state of the apfi configuration is re-established. As each apfi reconfiguration is associated with a categorical indication, the set of all possible reconfigurations is generic for the apf as well. In the following, the requirement class for the fusion of particular indications is introduced, which enables the specification of the categorical indications as the basis for each apfi reconfiguration[4]. Then, the requirement class for the mapp reconfiguration is presented.

B. Requirement Class for the Fusion of Particular Indications

The requirement class related to the fusion of particular indications describes transformations of the type

$$f_{fusion} : \{z_{I,i}\} \rightarrow \{z_{CI,x}\} \tag{1}$$

within a single module. Using an if-then-else construct, the requirement class can be expressed as follows[5]. Values that need to be specialised during instantiation are coloured and marked with #.

REQ - fusion of particular indications

Let $\mathbf{Z}_{I,\#nameCI\#,(\#M(x)\#)} := \{z_{I,\#nameI\#}\}_{(\#M(x)\#)}$.

If there is a $z_{I,i,(\#M(x)\#)} \in \mathbf{Z}_{I,\#nameCI\#,(\#M(x)\#)}$ with $z_{I,i,(\#M(x)\#)} = true$ for #confirmTime# ms,
then $z_{CI,\#nameCI\#,(\#M(x)\#)} = true$.
Else, $z_{CI,\#nameCI\#,(\#M(x)\#)} = false$.

EndOfREQ

Here, $\mathbf{Z}_{I,\#nameCI\#,(\#M(x)\#)}$ is the set of all particular indications $z_{I,\#nameI\#,(\#M(x)\#)}$ that are mapped to the categorical indication $z_{CI,\#nameCI\#,(\#M(x)\#)}$ within a specific module $\#M_{(x)}\#$.

The basic pattern for the instantiation condition related to the requirement class is illustrated in Fig. 6(a) using an UML syntax[6]. The pattern corresponds to the generic structure of f_{fusion} extended by the confirm time.

Specific instances of this pattern are searched for in the generated apfi design models, which result from the Axx subprocess. These models comprise the information about every existing instance of f_{fusion} for a specific apfi. Figure 6(b) exemplifies the relevant part of the design model for $cpm_{(R,1)}$. The matches of the instantiation pattern are outlined with a dashed line.

All matches with the same categorical indication are grouped at which each group triggers the instantiation of a corresponding requirement. With regard to the example in Fig. 6(b), two requirement instances are generated. The requirement instance or its textual representation[7] for the categorical indication $z_{CI,BIT}$[8] is as follows:

REQ - fusion of particular indications

Let $\mathbf{Z}_{I,BIT,(cpm(R,1))} := \{z_{I,Partition(1)Fail}, z_{I,CrossCompareFail}\}_{(cpm(R,1))}$.

If there is a $z_{I,i,(cpm(R,1))} \in \mathbf{Z}_{I,BIT,(cpm(R,1))}$ with $z_{I,i,(cpm(R,1))} = true$ for 20 ms,
then $z_{CI,BIT,(cpm(R,1))} = true$.
Else, $z_{CI,BIT,(cpm(R,1))} = false$.

EndOfREQ

[3]The apfi specific subset is defined in the high-level system design model for the Axx subprocess.

[4]Although, strictly speaking, this requirement class does not describe a reconfiguration, it serves to clarify basic mechanisms.

[5]In addition to the actual requirement text, a requirement class comprises further attributes, such as a rationale, assumptions or the allocation to apfi functions assigned to a specific Design Assurance Level (DAL).

[6]For the implementation into the xAx tool suite, the patterns are modelled using a dedicated domain-specific language. Details can be found in [11].

[7]The formal representation of the requirement classes is not considered in this paper but will be part of future publications.

[8]$z_{CI,BIT}$ leads to the execution of a module built-in test (BIT). If a fatal failure is found during this BIT, the module is passivated for the rest of the mission.

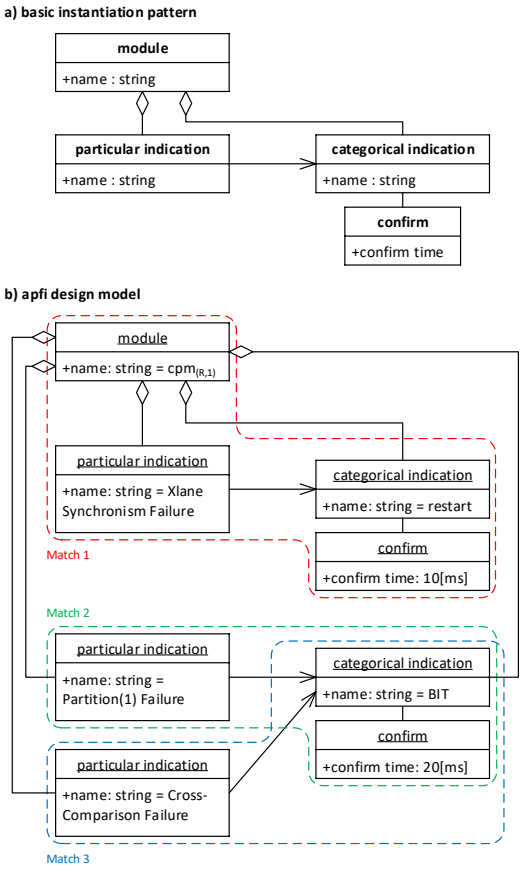


Fig. 6. (a) Basic instantiation pattern for the requirement class of the fusion of particular indications as well as (b) the relevant part of an exemplary apfi design model.

C. Basics of the Reconfiguration Requirement Classes

Compared to the requirement class of the fusion of particular indications, the following has to be considered for the reconfiguration requirement classes. While all reconfigurations are based exclusively on categorical indications, these categorical indications are only software-based quantities. However, states such as a power-off of all lanes of a module cannot be detected by the affected module and thus cannot be expressed by its categorical indications. For this reason, the physical health state z_s of each apfi hardware resource is introduced. Within the scope of the specification, the categorical indications are extended with these physical health states to so-called extended categorical indications. The vector of the extended categorical indications is defined as $\underline{z}_{ECI} := (\underline{z}_s, \underline{z}_{CI})^T_{ECI}$.

Based on the presented reconfiguration principle and the extended categorical indications, all possible system-level reconfigurations of each apfi can be generically described by the transformation:

$$f_{reconfig} : \left\{ \begin{matrix} \underline{s}_{apfi} \\ \begin{pmatrix} \underline{z}_s \\ \underline{z}_{CI} \end{pmatrix}_{ECI} \end{matrix} \right\} \rightarrow \{\underline{s}_{apfi}\}. \qquad (2)$$

Consequently, $f_{reconfig}$ is the basis for all reconfiguration requirement classes at the system level. As an example, the requirement class related to the mapp reconfigurations of apfis with two mapps and up to four cpms is presented in the following section.

D. Requirement Class for the Mapp Reconfiguration

The requirement class in Fig. 7 describes mapp reconfigurations due to the passivation or activation of cpms[9]. Therefore, it enables the specification of each transition from a mapp allocation for q correct cpms to the mapp allocation for r correct cpms. A correct cpm implies that the membership is given for this cpm. For a passivation transition, there is $q \in \{2,3,4\}$ and $r = q-1$. In contrast to this, for an activation transition $q \in \{1,2,3\}$ and $r = q+1$.

To express the requirement class based on $f_{reconfig}$, the vectors $\underline{s}_{xcpm,mapp,q}$ and $\underline{s}_{xcpm,mapp,r}$ as well as the vector $\underline{z}_{ECI,xcpm,mapp,q\mapsto r}$ are introduced.

- $\underline{s}_{xcpm,mapp,q}$ describes the initial steady-state apfi configuration, i.e. the apfi-wide mapp allocation for q correct cpms.
- Starting from q correct cpm, $\underline{z}_{ECI,xcpm,mapp,q\mapsto r}$ describes the apfi wide view of the passivation or activation of an arbitrary cpm to r correct cpms.
- $\underline{s}_{xcpm,mapp,r}$ describes the resulting steady-state mapp allocation for r correct cpms.

Using these vectors, the mapp reconfiguration requirement class is illustrated in Fig. 7. The used variables can be described as follows:

- $s_{mapp,(cpm(i))}$ describes the mapp currently allocated to an arbitrary $cpm_{(i)}$, with $s_{mapp,(cpm(i))} \in \{mapp_{(1)}, mapp_{(2)}, nil\}$. Nil is a state in which $cpm_{(i)}$ failed passive and thus no mapp is allocated.
- $z_{s,(cpm(j))}$ is the physical health state of an arbitrary $cpm_{(j)}$. It applies that $z_{s,(cpm(j))} \in \{c, f_p\}$, where c is a correct state of $cpm_{(j)}$ and f_p is a state in which $cpm_{(j)}$ failed-passive. Accordingly, $z_{s,(cpm(j))}$ is used to describe the passivation or activation of a $cpm_{(j)}$.
- Such a passivation or activation can be monitored by other, correct $cpm_{(i)}$ using the categorical indication $z_{CI,comm,(cpm(i),cpm(j))}$. It describes the opinion of $cpm_{(i)}$ about $cpm_{(j)}$ regarding their overall communication, with $z_{CI,comm,(cpm(i),cpm(j))} \in \{ok, \neg ok\}$. For example, if a $cpm_{(j)}$ failed-passive no messages are transmitted via any communication network to other $cpm_{(i)}$ which thus set $z_{CI,comm,(cpm(i),cpm(j))} = \neg ok$.

For the indices that define the assignment of these variables to the cpms the following applies.

- Considering $\underline{s}_{xcpm,mapp,q}$, the network side indices $x, y \in \{R, B\}$, with $x \neq y$. For the corresponding module ID indices, there is $i, j, k, l \in \{1, 2\}$, with $i \neq j$ and $k \neq l$.
- Analogously, considering $\underline{z}_{ECI,xcpm,mapp,q\mapsto r}$ and $\underline{s}_{xcpm,mapp,r}$, the network side indices $a, b \in \{R, B\}$,

[9]Note that the temporal aspects of this requirement class are neglected in the context of the paper.

REQ - Mapp reconfiguration due to a change from $\#q\#$ to $\#r\#$ correct cpms

$$\begin{pmatrix} \#\underline{s}_{xcpm,mapp,q}\# \\ \#\underline{z}_{ECI,xcpm,mapp,q\mapsto r}\# \end{pmatrix} \mapsto (\#\underline{s}_{xcpm,mapp,r}\#)$$

$$= \left(\# \begin{pmatrix} s_{mapp,(cpm(S(x),i))} & s_{mapp,(cpm(S(x),j))} \\ s_{mapp,(cpm(S(y),k))} & s_{mapp,(cpm(S(y),l))} \end{pmatrix}_q \# \right.$$

$$\left. \# \left(\begin{pmatrix} z_{s,(cpm(S(a),m))} \\ z_{CI,comm,(cpm(S(a),m),cpm(S(a),n))} \\ z_{CI,comm,(cpm(S(a),m),cpm(S(b),o))} \\ z_{CI,comm,(cpm(S(a),m),cpm(S(b),p))} \\ z_{s,(cpm(S(b),o))} \\ z_{CI,comm,(cpm(S(b),o),cpm(S(a),m))} \\ z_{CI,comm,(cpm(S(b),o),cpm(S(a),n))} \\ z_{CI,comm,(cpm(S(b),o),cpm(S(b),p))} \end{pmatrix} \begin{pmatrix} z_{s,(cpm(S(a),n))} \\ z_{CI,comm,(cpm(S(a),n),cpm(S(a),m))} \\ z_{CI,comm,(cpm(S(a),n),cpm(S(b),o))} \\ z_{CI,comm,(cpm(S(a),n),cpm(S(b),p))} \\ z_{s,(cpm(S(b),p))} \\ z_{CI,comm,(cpm(S(b),p),cpm(S(a),m))} \\ z_{CI,comm,(cpm(S(b),p),cpm(S(a),n))} \\ z_{CI,comm,(cpm(S(b),p),cpm(S(b),o))} \end{pmatrix} \right)_{q\mapsto r} \# \right) \mapsto \left(\# \begin{pmatrix} s_{mapp,(cpm(S(a),m))} & s_{mapp,(cpm(S(a),n))} \\ s_{mapp,(cpm(S(b),o))} & s_{mapp,(cpm(S(b),p))} \end{pmatrix}_r \# \right)$$

EndOfREQ

Fig. 7. Requirement class for the mapp reconfiguration due to the passivation or activation of cpms.

with $a \neq b$. Furthermore, the related module ID indices $m, n, o, p \in \{1, 2\}$, with $m \neq n$ and $o \neq p$.

There is a major difference between the instantiation of the requirement class for the fusion of particular indications and the requirement class for the mapp reconfiguration. Here, the aforementioned mapp allocation rules for apfis with two mapps and up to four cpms are a part of the requirement class. All correct cpms have to meet these rules, which are as follows:

1) The number of replicas of $mapp_{(i)}$ per network side is at most 1: $i \in \{1, 2\}, x \in \{R, B\} : N_{mapp(i),(S(x))} \leq 1$.
2) The number of replicas of $mapp_{(1)}$ is equal to or at most greater than 1 compared to the number of replicas of $mapp_{(2)}$: $N_{mapp(1)} - 1 \leq N_{mapp(2)} \leq N_{mapp(1)}$.

The first rule is based on a design decision related to the communication architecture of the apf. The second rule ensures that each mapp is executed for as long as possible while taking into account the mapp prioritisation.

Using these rules, the requirement class itself generically describes each mapp reconfiguration due to a passivation or activation. Consequently, the number of mapps and cpms is the only characteristic required for the generation of the requirement instances. The associated basic instantiation pattern is shown in Fig. 8(a). If an apfi does show this characteristic, all requirement instances of this class are generated, i.e. one requirement instance for each possible specific activation and passivation transition for the apfi specific number of cpms and mapps. The instantiation is illustrated in the following.

The apfi design model contains the information about the number of mapps and cpms of an apfi. An exemplary model for an apfi with two mapps and four cpms is illustrated in Fig. 8(b). The depicted mapp to cpm relation results in a single match for the class's instantiation pattern. Accordingly, requirement instances for the activation transitions $q \mapsto r \in \{1 \mapsto 2, 2 \mapsto 3, 3 \mapsto 4\}$ have to be generated. Furthermore, requirement instances for the passivation transitions $q \mapsto r \in \{4 \mapsto 3, 3 \mapsto 2, 2 \mapsto 1\}$ have to be generated. In the following, the instantiation steps carried out for each requirement instance are described using the example of a passivation transition from $q = 4$ to $r = 3$ correct cpms.

First, the value of $\underline{s}_{xcpm,mapp,q}$ is determined based on $q = 4$ and the apf generic mapp allocation rules. The resulting

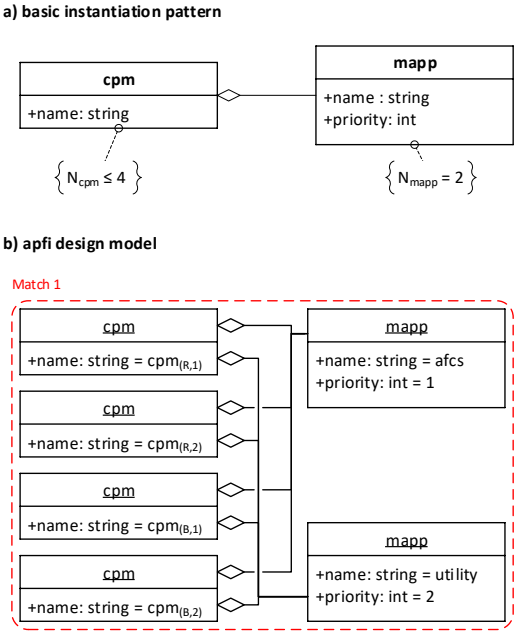


Fig. 8. (a) Basic instantiation pattern for the requirement class of the mapp reconfiguration as well as (b) the relevant part of an exemplary apfi design model.

correct mapp allocation for four correct cpms is

$$\underline{s}_{xcpm,mapp,4} = \begin{pmatrix} mapp_{(1)} & mapp_{(2)} \\ mapp_{(1)} & mapp_{(2)} \end{pmatrix}_4 . \tag{3}$$

Second, the apfi-wide view on the passivation of a cpm is expressed using $\underline{z}_{ECI,xcpm,mapp,q\mapsto r}$. In our example, the passivation of an arbitrary $cpm_{(S(b),p)}$ is represented by

$$\underline{z}_{ECI,xcpm,mapp,4\mapsto 3} = \begin{pmatrix} \begin{pmatrix} c \\ ok \\ ok \\ \neg ok \end{pmatrix} & \begin{pmatrix} c \\ ok \\ ok \\ \neg ok \end{pmatrix} \\ \begin{pmatrix} c \\ ok \\ ok \\ \neg ok \end{pmatrix} & \begin{pmatrix} f_p \\ - \\ - \\ - \end{pmatrix} \end{pmatrix}_{4\mapsto 3} . \tag{4}$$

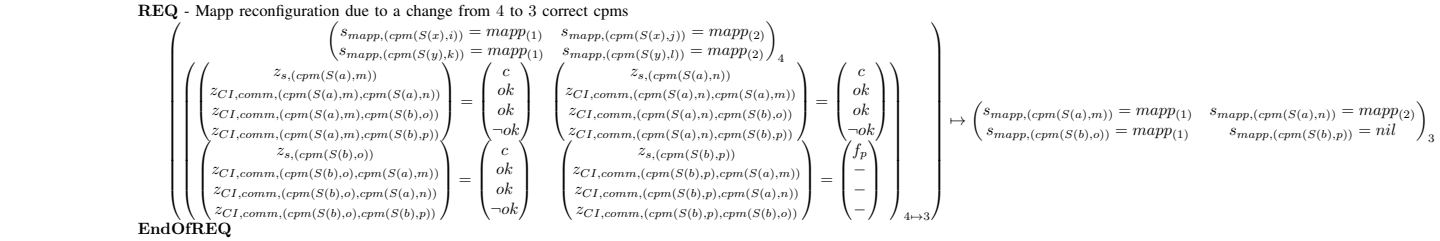


Fig. 9. Requirement instance for the mapp reconfiguration due to the passivation of an arbitrary cpm, from $q = 4$ to $r = 3$ correct cpms.

Third, with $r = q - 1$ the value of $\underline{s}_{xcpm,mapp,r}$ is determined based on the aforementioned mapp allocation rules, i.e.

$$\underline{s}_{xcpm,mapp,3} = \begin{pmatrix} mapp_{(1)} & mapp_{(2)} \\ mapp_{(1)} & nil \end{pmatrix}_3 . \tag{5}$$

The resulting requirement instance for the passivation of an arbitrary cpm from one of $q = 4$ correct cpms is shown in Fig. 9.

Note that a single requirement instance specifies the passivation transition from $q = 4$ to $r = 3$ correct cpms for all possible permutations. Here, a permutation is a specific configuration of how the mapps are allocated to the cpms of an apfi as well as of the cpm that failed-passive. Thus, a permutation is a specific selection of the indices that define the assignment to the cpms[10].

Regarding the verification of such a requirement instance, the following has to be considered. First, the transition must be verified for all specific permutations for a complete verification of the required behaviour. Second, to our understanding, performing the related tests based on the stimulation of the categorical indications is not sufficient for a verification at the system level. In order to verify the entire end-to-end system behaviour, the composed transformation

$$f_{moni} \circ f_{fusion} \circ f_{reconfig} \tag{6}$$

is considered, with $f_{moni} : \{ev_i\} \rightarrow \{z_{I,i}\}$. This means that to verify a reconfiguration, not the categorical indication is stimulated but an actual exemplary failure event associated with the categorical indication. In the given example, such a failure event could be the power interruption of a cpm.

V. Conclusion

For a significant reduction of the costs related to the development of fault-tolerant real-time avionics systems, the Flexible Avionics Platform (apf) and an associated comprehensive automation process, the AAA process, were developed. The apf is characterised by a high-level abstraction layer, the Platform Management, providing a full abstraction of distribution, fault-tolerance, and redundancy towards integrated system functions. The associated AAA process combines the high flexibility of this platform-based approach with a high degree of automation. Therefore, the generation of the development and qualification artefacts of each Flexible Avionics Platform Instance (apfi), such as the system-level specification and the verification artefacts, is largely automated.

This paper focused on the automatic generation of the system-level specification for the high-level Platform Management. The presented approach with its specification subdomains is made possible by the systematic and clear structuring of the apf [15]. Each of the named subdomains is dedicated to specific apfi behaviour. These are the intra-module reconfigurations, inter-module reconfigurations, peripheral reconfigurations, and an apfi's entire operation moding. Our article describes the definition of the requirement classes for the apfi reconfigurations which can be expressed based on an apf generic set of categorical indications that classify all possible failure events of an apfi. On this basis, a limited number of requirement classes were defined. They are sufficiently generic to cover the apf's entire usage domain while still being expressive and comprehensible. This facilitates a manual validation as part of a certification process. In addition to the presented actual requirement text of a requirement class, other attributes can of course be added according to the applied requirements standard. The developed requirement classes for the high-level Platform Management were integrated into the AAA tool suite and thus enable the automatic instantiation of reconfiguration and operation moding requirements. This further completes our overall approach for a cost-efficient development and qualification of fault-tolerant, distributed and integrated avionics systems such as fly-by-wire systems.

The apf approach was successfully demonstrated within multiple projects up to an in-flight demonstrator featuring automatic take-off and landing [17], [18], [19]. Our ongoing research in the LuFo V-3 research project Secured System for Manned Multicopter (SESYMM) focuses on optimising the generated artefacts with regard to DAL A conformity. Moreover, the AAA tool suite is used for the development of a fly-by-wire platform for a remotely piloted aircraft system based on a CS-23 aircraft for operation in the non-segregated airspace without an onboard safety pilot.

Glossary

apf The Flexible Avionics Platform is a platform-based development approach featuring an integrated, distributed, and highly redundant architecture. It is characterised by a high-level abstraction layer, the

[10]The passivation transition illustrated in Fig. 4 is a specific permutation of this requirement instance, with $x = R$, $y = B$, $i = k = 1$, $j = l = 2$, $a = B$, $b = R$, $m = p = 1$, and $n = o = 2$.

Platform Management, enabling integrated applications to be executed in a failure-free virtual simplex environment. Thus, applications can be reduced to their cybernetic control law.

apfi An instance of the Flexible Avionics Platform.

cpm Core processing modules plan and execute the central management decision within an apfi. In addition, they execute the applications for the integrated system function laws.

mapp A multi-application comprises several integrated system function laws of the same criticality level. A dynamic in-flight mapp reallocation between cpms enables an efficient use of the cpm hardware.

REFERENCES

[1] J.-B. Itier, "A380 integrated modular avionics—the history, objectives and challenges of the deployment of ima on a380," in *Proceedings of the ARTIST2 Meeting on Integrated Modular Avionics, Roma, Italy*, 2007, pp. 12–13.

[2] B. Annighöfer and E. Kleemann, "Large-scale model-based avionics architecture optimization methods and case study," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 55, no. 6, pp. 3424–3441, 2019.

[3] T. Gaska, C. Watkin, and Y. Chen, "Integrated modular avionics-past, present, and future," *IEEE Aerospace and Electronic Systems Magazine*, vol. 30, no. 9, pp. 12–23, 2015.

[4] B. Kornek-Percin, B. Petersen, M. Reichle, and J. Bader, "New ima architecture approach based on ima resources," in *2015 IEEE/AIAA 34th Digital Avionics Systems Conference (DASC)*. IEEE, 2015, pp. 6A2–1.

[5] M. Halle and F. Thielecke, "Evaluation of the ashley seamless tool-chain on a real-world avionics demonstrator," in *2017 IEEE/AIAA 36th Digital Avionics Systems Conference (DASC)*. IEEE, 2017, pp. 1–9.

[6] B. Annighoefer, M. Brunner, J. Schoepf, B. Luettig, M. Merckling, and P. Mueller, "Holistic ima platform configuration using web-technologies and a domain-specific model query language," in *2020 AIAA/IEEE 39th Digital Avionics Systems Conference (DASC)*. IEEE, 2020, pp. 1–10.

[7] J. Yin, B. Lawler, and H. Jin, "Application of model based system engineering to ima development activities," in *2017 IEEE/AIAA 36th Digital Avionics Systems Conference (DASC)*. IEEE, 2017, pp. 1–7.

[8] S. Görke, *Eine flexible Plattform für Fly-by-Wire-Systeme-Spezialisierbarkeit als Schlüssel zur effizienten Entwicklung sicherheitskritischer Avionik*. Verlag Dr. Hut, 2013.

[9] M. Di Natale and A. L. Sangiovanni-Vincentelli, "Moving from federated to integrated architectures in automotive: The role of standards, methods and tools," *Proceedings of the IEEE*, vol. 98, no. 4, pp. 603–620, 2010.

[10] F. Kraus, *Verfahren zur weitgehend automatisierten Erzeugung der Middleware für hoch ausfallsichere, integrierte Avioniksysteme mittels Model-Integrated Computing*. Verlag Dr. Hut, 2018.

[11] T. Belschner, *A Method for the Automated Generation of Requirements and Traceability for a Distributed Avionics Platform*. Verlag Dr. Hut, 2020.

[12] P. Müller, *Automated Test Artifact Generation for a Safety-Critical Integrated Avionics Platform*. Verlag Dr. Hut, 2021.

[13] C. Block, S. Dikmen, and R. Reichel, "Automated test case generation for the verification of system and high-level software requirements for fly-by-wire platforms," in *AIAA SCITECH 2022 Forum*, 2022, p. 0254.

[14] V. Hadzilacos and S. Toueg, "A modular approach to fault-tolerant broadcasts and related problems," Cornell University, Tech. Rep., 1994.

[15] T. Hoffmann, R. Wipperfürth, and R. Reichel, "Enabling the automated generation of the failure and redundancy management for distributed and integrated fly-by-wire avionics," in *2021 IEEE/AIAA 40th Digital Avionics Systems Conference (DASC)*. IEEE, 2021, pp. 1–10.

[16] F. Cake, *Supervisor für eine komplexe verteilte Avionikplattform*. Verlag Dr. Hut, 2016.

[17] L. Dalldorff, R. Luckner, and R. Reichel, "A full-authority automatic flight control system for the civil airborne utility platform s15-lapaz," *Euro GNC*, 2013.

[18] R. Kueke, P. Mueller, S. Polenz, R. Reichel, F. Pinchetti, J. Stephan, A. Joos, and W. Fichter, "Fly-by-wire for cs23 aircraft-core technology for general aviation and rpas," in *Aviation in Europe Innovation for Growth-7th European Aeronautics Days*, 2015.

[19] S. Görke, R. Riebeling, F. Kraus, and R. Reichel, "Flexible platform approach for fly-by-wire systems," in *2013 IEEE/AIAA 32nd Digital Avionics Systems Conference (DASC)*. IEEE, 2013, pp. 2C5–1.

Session Th.2.C

Space applications

Thursday 2nd June

11:30

–

Room Pastel

Digital transformation in the European Space Industry

- Category: Regular Paper

- Author: Jean-Loup Terraillon, Software System Lead Engineer, European Space Agency, Software System Department, Noordwijk, The Netherlands

- Keywords: Model-based system engineering, digitalisation, space, spacecraft

Abstract

Digitalisation is a trend in most industrial domains. The Space domain has embarked on it a couple of years ago. Starting from a Model Based for System Engineering initiative, the discussion between Space Agencies and Industry has intensified with several topical working groups. The scope has been enlarged, from MBSE, to full engineering digitalisation, aiming at producing the enablers that will allow to develop digital twins.

The paper describes and positions the ESA digitalisation project, as a joint Space Community endeavour, proposing a development approach, and giving programmatic elements as well as the initial panorama of MBSE deployment in Space projects.

1. CONTEXT

The **ESA Agenda 2025** [1] is a document prepared by the European Space Agency Director General when he took up duty, and indicating to the Space Community, and in particular the ESA Member's States Delegations, the directions for the next four years. It says in particular:

- "*In Europe, ESA has the unique ability to implement, together with industry, complex and ambitious space missions and programmes on an equal footing with other leading space agencies worldwide. We will ensure that this ESA strength and value is further reinforced.*"
- "*ESA will therefore digitalise its full project management, enabling the development of digital twins, both for engineering by using Model Based System Engineering, and for procurement and finance, achieving full digital continuity with industry.*"

Digitalisation is the process of representing all the artefacts of space systems under a (structured) digital representation, on which computers can reason and elaborate. The most explicit example of such digitalisation process is the Model Based technique, where the information, traditionally contained in the form of documents, is instead expressed in a set of data, structured into a model. Computers can be programmed to navigate and search into the models, and create relations between associated data, allowing to discover more properties, and to derive added value such as traceability, optimisation, technical budgets, trends, and knowledge.

Digitalisation includes also e.g. databases or excel sheets, or any format where data is structured, curated and associated to a semantic that allows to unambiguously understand it with a computer. This transformation relies on common standards and new infrastructures that facilitate the exchange of data but also the collaboration along the whole supply chain.

2. THE DIGITAL SPACECRAFT

2.1. General Concept

Digitalisation is under way in many domains: automotive [2], railways [3], aircraft [4], construction [5], airports (Schiphol twin) [6], medical [7], Eurocontrol [8].

The Digital Spacecraft is a concept derived from these similar initiatives, and introduced in ESA to cover the digital transformation of space, ground, launcher segments development and operation, as a new way to collaborate within the space ecosystem throughout the full project lifecycle. It is based on digitalisation and combines as an umbrella a wide spectrum of topics like MBSE, digital transformation, digital twins and full data integration into a single consistent concept covering all aspects related to a spacecraft.

Digitalisation is already ubiquitous in the engineering domain. It allows expressing the concept of operations, structuring the requirements, and tracking them through design and manufacturing up to testing. It is intended to support the optimisation of the design, to make informed trade-offs and to clarify the interfaces. It allows the systems engineers and the project teams to master the complexity of the space system, relying on an authoritative

source of truth of the system, communicating knowledge and exchanging data in an unambiguous way. It enables quick access to key parameters via dashboards, understanding the impact of changes and managing those, building knowledge from lessons learned for optimisation and/or for future reuse. It also allows maintaining a continuous and homogeneous flow of information between the different disciplines of the system, along its life cycle, and across its supply chain.

Besides engineering, project management and product assurance are also transitioning to digital, supporting the workflows related to schedule, changes, deviations and non-compliances in an automated, traceable and searchable way. Finally, procurement complements the picture with the digitalisation of the steps included in the preparation of the Invitation To Tender (ITT) package, management of the resulting contract, deliverables, contract change process and the contract closure, associated to adequate legally compliant processes.

2.2. Space domain characteristics

Producing spacecraft would not appear a priori much different from producing aircraft or cars. The overall life cycles are similar (specification, design, manufacturing, tests). However:

- the number of items produced is different: most satellites are one-off (science) or a few are produced (earth observation). The Galileo constellation is the largest, with 24 satellites and 6 spares. Europe is not yet involved in large constellations like Starlink.
- The way to test them is different, as you cannot test-drive or flight-test a spacecraft or a launcher. The overall development process is ruled by specific Standards called European Cooperation for Space Standardisation (ECSS) [9].
- Although Newspace may trigger some changes, supported by ESA Director General, the current spacecraft procurement is specific, generally done by the various European Space Agencies, through a Supply Chain, using ESA Member States budgets distributed according to specific rules.
- These budgets would not allow partnerships like the one signed between Airbus aircraft and Dassault Systèmes to co-engineer the future aircraft and its digital development environment. They would not either allow ESA to impose specific digital technology to the Space industry without consideration of their valuable heritage in the domain.

Therefore, the Space domain must deploy digitalisation in a context constrained by contractual rules, processes, organisation and technological heritage. This boils down to an essential need: interoperability of all the tools, across disciplines, along the life cycle, and through the Supply Chain. This need appeared in 2018 during a workshop organized by the French Space Agency (CNES) on MBSE [10], where it appeared clearly that industry had internally matured some digital technology, but was facing issues in the collaboration between the stakeholders of a project. ESA had therefore a harmonisation role to play.

2.3. A Joint European Vision

Digital transformation of space systems is an ESA Member States endeavour.

Digitalisation is a change process. The key of its efficiency is the digital continuity in the three dimensions of space systems development: disciplines, life cycle and supply chain. This means that all space actors are essential for digitalisation.
Space actors are not progressing at the same pace, though: some are more advanced in functional system engineering, other in manufacturing, disciplines are not evenly digitalised, and principally,
The success of the change process will be as strong as its weakest link. Consequently, a widespread, staged development and incremental implementation approach is of strategic importance to bring all industrial actors at the same level, and this is where ESA and its Member States have a role to play.

The most important change lies on the new “ways of working” even more than on evolving the processes and tools. Digitalization is not about doing what we are currently doing better or in a smarter way, it is about changing what we do and the way we do it. The Digital Spacecraft is a way to change drastically the way Agencies, primes and supply-chain work together, switching from a 100% customer/supplier relation to a mix between co-engineering approach and contractual relation. This mind-set change is the key enabler for future European success and digitalization is its enabler. Digital engineering come along with higher transparency, trust, partnership and objectives alignment.

2.4. A multi-level harmonisation forum

Digitalisation involves the space European community through a number of European joint agency-industry working groups led by ESA:

- *the MB4SE Advisory Group [11]*
  This group has been established in 2019 with the objective to deploy Model Based System Engineering in Space projects. It includes five space agencies (ESA, CNES, DLR, ASI, and UKSA) and four Large Systems Integrators (ADS, TAS, OHB, and ArianeGroup). It advises ESA on the technical aspects of MBSE research and development, and mainly on interoperability of tools, through the seamless exchange of the associated data.

  The MB4SE AG serves as Steering Group of the *OSMoSE Governance [12]* group. The "Overall Semantic Modelling for System Engineering" initiative arises from the need of enhancing the way information and knowledge is exchanged among the stakeholders involved, enabling efficient interoperability among model-based infrastructures used. The OSMoSE initiative addresses interoperability at semantic level, merging all stakeholder's concepts into a global conceptual data model resulting in the so-called Space System Ontology (see 3.1).

- *the Digital Spacecraft Think Tank [13]*
  This multidisciplinary group has been established in December 2020, with ESA and Industry (i.e. Airbus Defence and Space, Thales Alenia Space, Ariane Group, RUAG and OHB/MT Aerospace). It was established by ESA's Executive Board to engage in identifying and harmonizing the steps to be undertaken for the digital transformation of the end-to-end space mission development and operation, with the aim to be beneficial for all stakeholders in terms of technical efficiency, schedule and consequently costs. This team was mandated to prepare a detailed plan for the progressive deployment of the Digital Spacecraft approach across the Agency in close cooperation with industry.

  The Think Tank is steered by a *Digital Spacecraft Steering Committee* gathering higher managerial levels of stakeholders in view of endorsing the plans and empowering the teams.

Figure 1 depicts the digitalisation as a rocket, where the stages and boosters are the working groups of industry and agencies, delivering the Digital Spacecraft in orbit… IT platform and Data Management are addressed in 3.3 and 3.4.

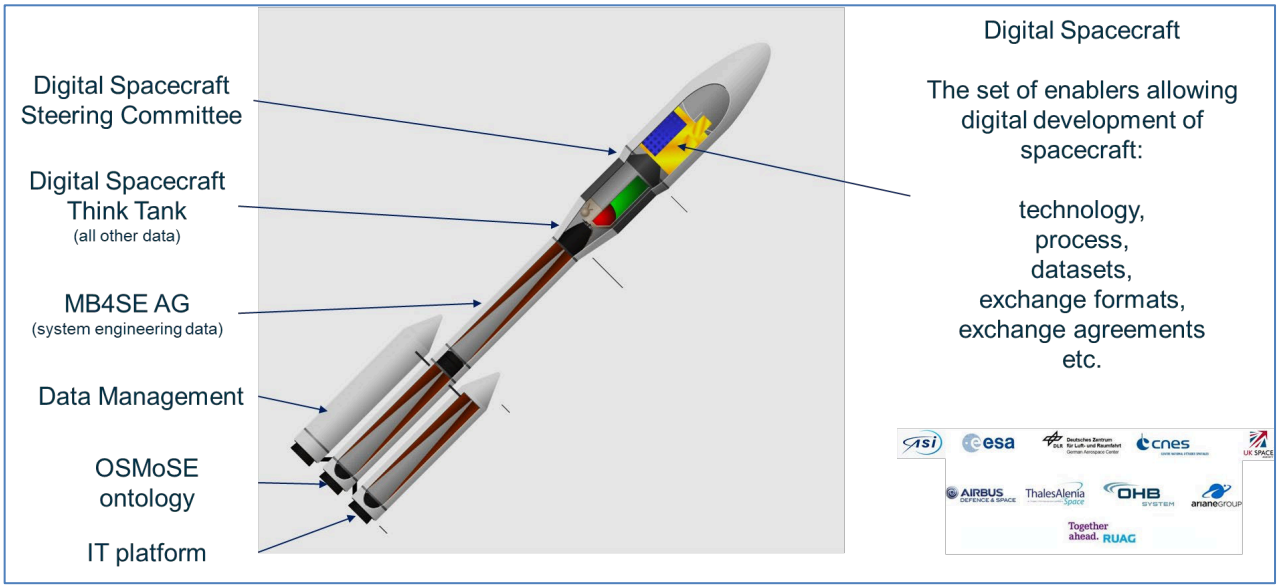


Figure 1 A systematic, multilayer, collaboration

In the Digital Spacecraft Think Tank, several discussions were held on the definition of the Digital Spacecraft and on the support of the “Digital Transformation”, which is a myriad of new thinking and development in the areas of data and lifecycle management, throughout the entire ecosystem, and that can present many unforeseen challenges.
The term “Digital Spacecraft” represents a concept for an end-to-end digital continuous way of working on space missions in the European ecosystem. The Digital Spacecraft is more a set of digitalisation enablers, than a single product, or a particular technology. It includes a number of technologies such as Model Based System Engineering (MBSE), Computer Aided Design (CAD), simulation techniques, Augmented, Virtual and Mixed Reality (AR/VR/MR), Artificial Intelligence (AI) and various communication, collaboration and project control and management technologies at infrastructure level.

Leveraging the new breakthroughs on coupling the simulation results with physical data in continuous synch (digital twin technologies), the Digital Spacecraft allows for feedback and learning (e.g. from operations back to development, and among projects). The Digital Spacecraft therefore enables understanding, learning, reasoning and dynamically recalibrating models, for an improved decision-making.

3. DIGITALISATION IMPLEMENTATION

3.1. Principle

The goal is to identify the digital threads occurring in Space projects, such as to ensure the digital continuity in three dimensions.

- The data shared between disciplines should be defined and understood the same way by everyone, and exchanged continuously between engineers.
- The Spacecraft data state reached at a particular milestone of the life cycle should be propagated in the next phase, in order to ensure traceability and avoid distortion along the development.
- The relationship Customer-Supplier should be based on exchanges of models and data rather than of documents.

To identify the data threads, it is necessary to identify the processes followed by these data, as well as the overall data model. The processes indicate the exchanges between the disciplines or the stakeholder, i.e. the point where we want to be interoperable. Defining process and data at the global level of a spacecraft will fail. Instead, applying the concept of divide and conquer, the identification of processes and data models is first done at discipline level, for which relevant experts can support the local digitalisation of the discipline that they know.

In a second step, putting together the data models shows that some data are shared between disciplines. For example, the system product tree is used by Product Assurance, the centre of gravity of the CAD tool is used by the control system, etc. It is essential that the two disciplines have the same understanding of the shared data: same meaning (dry mass, wet mass), same referential, same units, same structure, same financial reference (for cost data), etc. This means that each data must have a unique semantic for everyone.

Defining the common semantic of data is done at the level of an *ontology*, which is commonly discussed and agreed in the OSMoSE group [12]. Indeed, a set of data can have a physical representation (e.g. XML), a logical representation (e.g. a meta-model), and a conceptual representation (an entity-relation concept, called here ontology). The ontology language selected is ORM [12],[14]. ORM allows to define the agreed entity-relationship of the data set, as well as (thanks to the specific mechanism of “derivation”), any semantically equivalent expression preferred by one or another stakeholder. This semantic equivalence allows generating automatically a physical representation of the data set appropriate for each stakeholder, as well as the tool that translates one format into another one. This allows achieving what is called “*semantic interoperability*” (Figure 2).

It is therefore key that the data sets identified by each disciplines are abstracted into an ontology, all ontologies are harmonised into a single “Space System Ontology (SSO), allowing for semantic interoperability of the tools between the disciplines. This applies in the same way to the exchanges between customer and supplier.

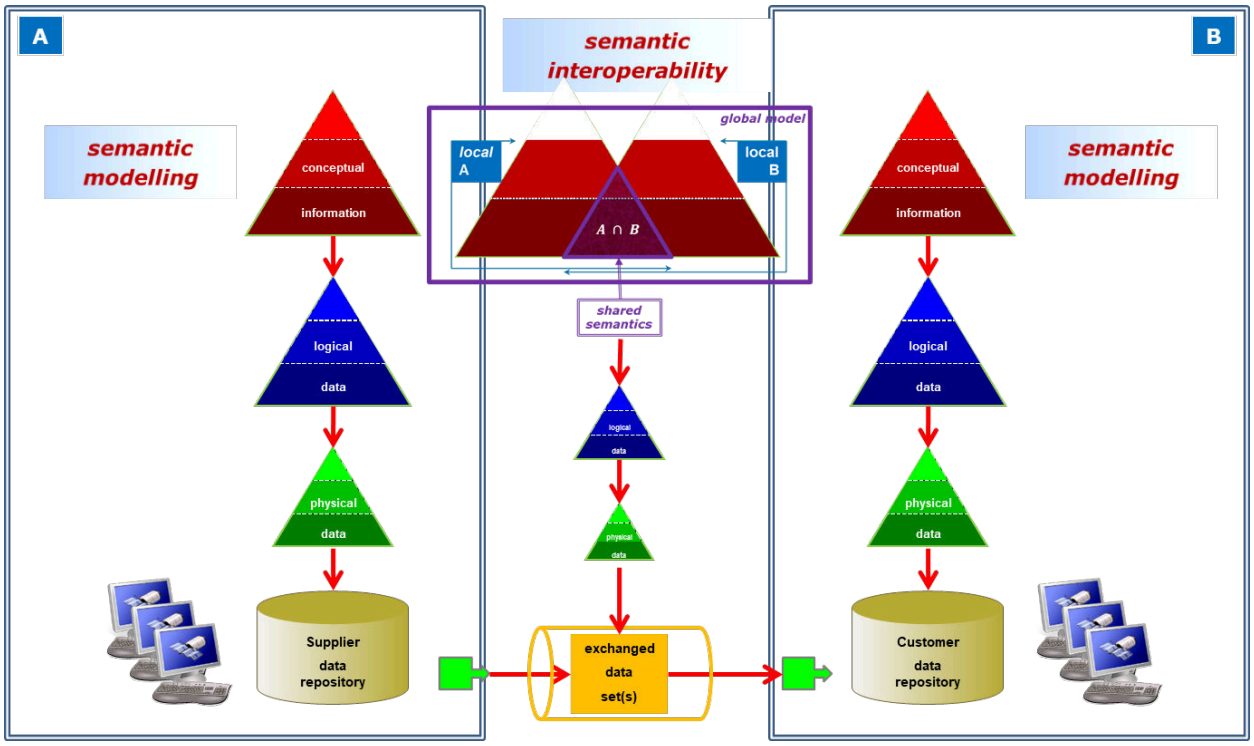


Figure 2 Semantic interoperability

3.2. Application

Figure 3 and Figure 4 illustrate the first steps enabling digital continuity through the development of digitalisation enablers, and in particular

- formalisation of the processes, artefacts and data models of each engineering discipline,
- alignment of the data models with help of the Space System Ontology
- deployment in the Supply Chain to exchanges between each level of the Customer Supplier relationship.

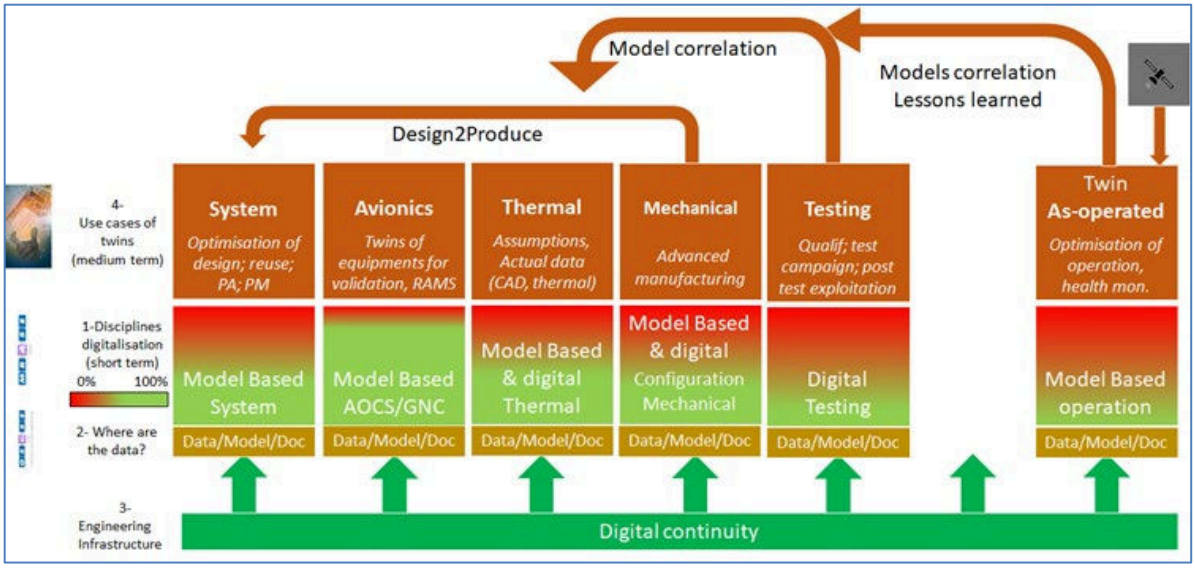


Figure 3 Illustrative example of Digitalisation of each discipline, then digital continuity, then data exploitation

Some disciplines are illustrated here; they need (1) to increase their level of digitalisation, using model-based or other techniques, (2) to identify where are the data in the process, (3) to identify their own set of tools that will later be connected to achieve digital continuity. When these digitalisation enablers are available, data are exploited (4) to optimise and reuse designs, to produce digital twins, to verify test coverage, to make model correlations with the actual data of the physical items. The "as operated" twin is the ultimate achievement allowing managing efficiently the operations of the real spacecraft. Design2Produce is an ESA R&D programme.

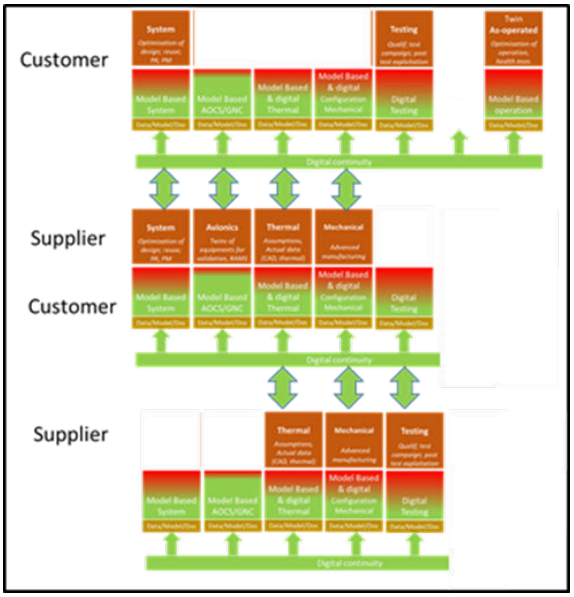


Figure 4 Artist impression of the propagation of digital continuity along the supply chain

The digitalisation enablers are connected through the Supply Chain, generally between the same discipline, allowing experts to work together. Depending on the metier of the Supplier, only some disciplines need to be connected.

3.3. The IT Platform

The next issue is the one of the informatics environment supporting the digitalisation. At ESA, it is called "*IT Platform*". It should ultimately be similar to an (Extended) Enterprise Architecture. This paper will not address the "corporate" part of it dedicated to management, contractual or financial aspects.

Figure 5 is an attempt to understand which tools, presently used in ESA, should be connected to ensure digital continuity. The main interest of the figure is its complexity that illustrates the difficulty to implement such a platform. In this figure, all discipline tools are connected to a central exchange tool called "data hub", making sure that everyone has the right value of the right data at the right time.

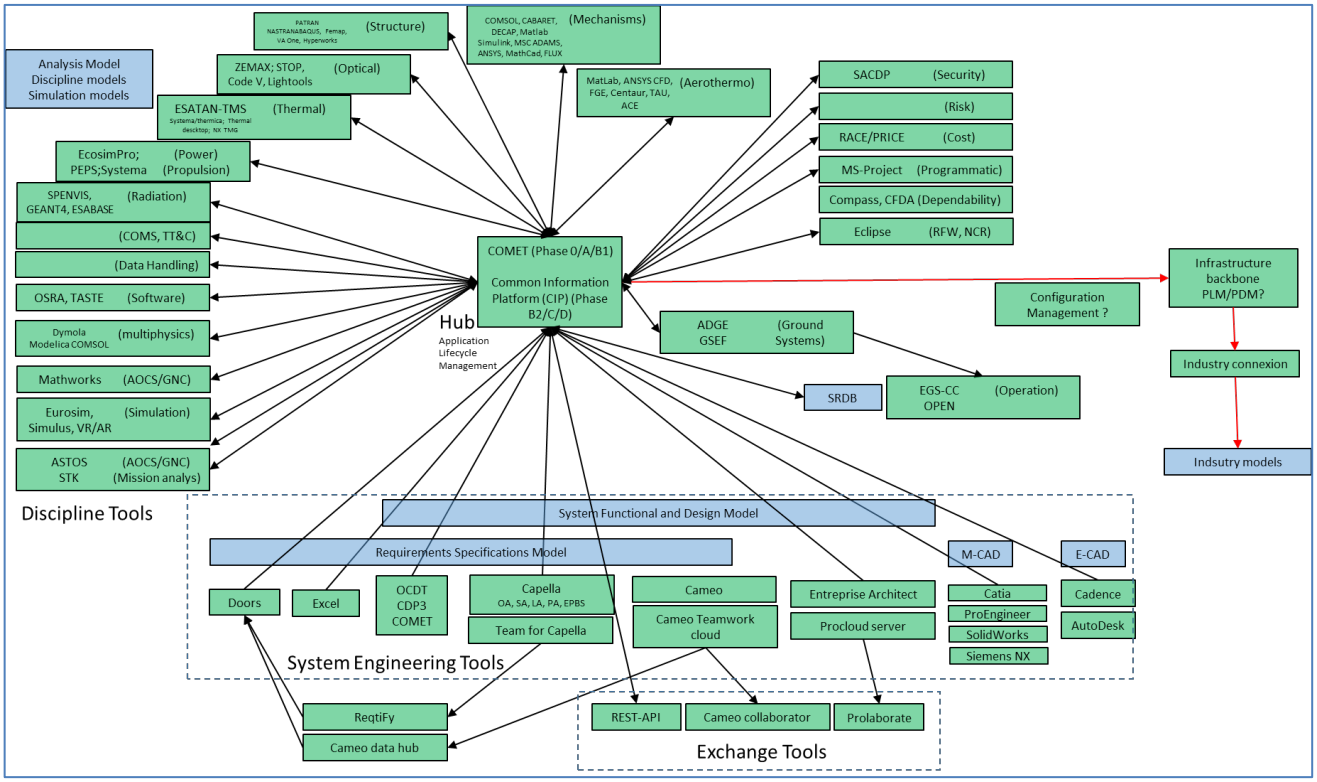


Figure 5 status of spacecraft engineering tools to be included in an IT platform

At the bottom are the system tools: requirement engineering, functional, mechanical, and electrical. The discipline tools come as an umbrella around the central Data Hubs (early phase, later phase, and ground segment specific). The data hub is connected to the IT Platform and the (extended) Entreprise.

This architecture is inspired by the ESA IT platform of the Concurrent Design Facility [15] called COMET [16]

COMET is used in early phases of spacecraft development (called phase 0/A/B1) to centralise all the data of the discipline involved in the concurrent design. Data refreshed by one discipline are immediately pushed to the others. This allows converging quickly during the early mission and spacecraft concept definition. COMET is also used in an activity called Digital Engineering Hub Pathfinder (DEHP) [17], where it intends ultimately to connect all the tools depicted in Figure 6.

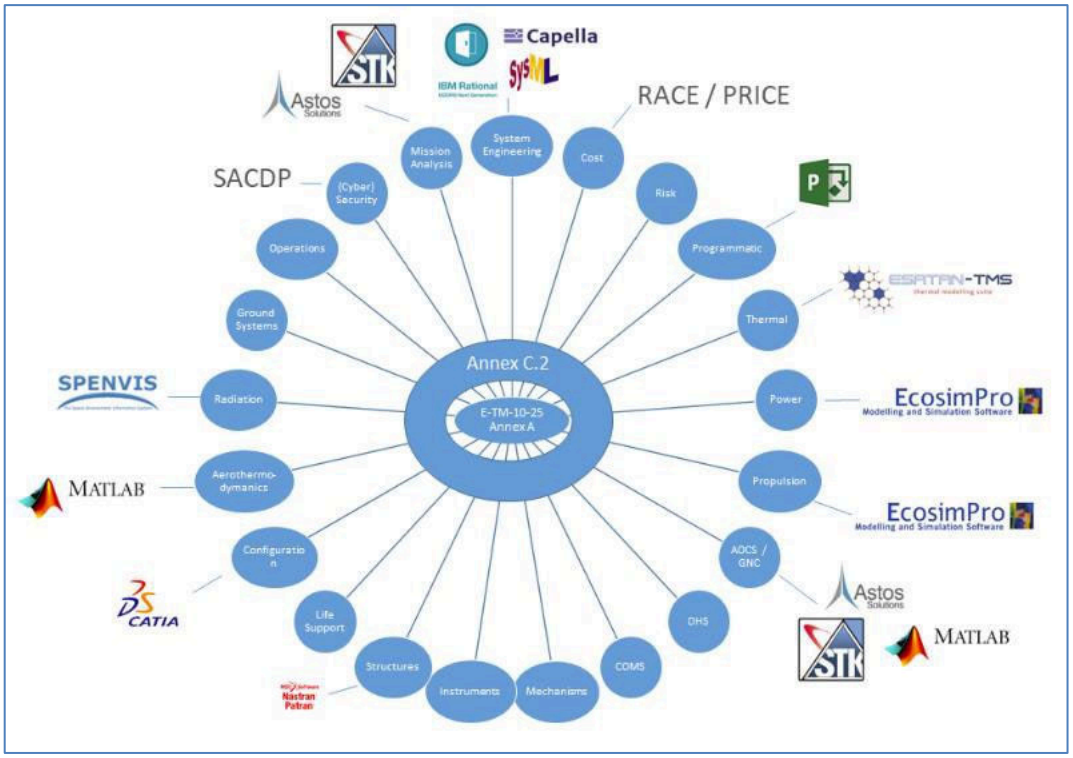


Figure 6 The spoke wheel concept of the Engineering Hub (courtesy Rhea)

This wheel shows that, up to phase B1, the number of tools is “relatively” limited. The complexity arrives in later phases, where the complete workshop of engineering, operations, product assurance and management tools are deployed. ESA is running some activities (not yet completed) to develop Data Hubs for future phases.

3.4. Data management

Digitalisation is not yet fully achieved once data threads are identified and IT Platform is available... Other issues appear such as:

- data ownership and visibility: in the relationship between customer and supplier, the exchange of documents has traditionally allowed the supplier to control the visibility that he wanted to give to his customer on the design. However, “*you cannot hide behind the data*”. Delivering the supplier model can expose a lot of information, including some know-how, to the customer. ESA observes more and more industry reluctance to deliver some models, whereas ESA needs it to perform his tasks. It is important to establish a relation of trust, of course supported in a legal frame, allowing the supplier to know which data are needed by the customer and to do what.
- this applies not only to the verification performed by ESA projects teams on industry design and tests, but also to the ESA reviews. Reviews are milestones in the life cycle where the status of the project is

analysed, is compared to an expected maturity at that stage, and results into mitigation actions to palliate the potential weaknesses. ESA reviews are often done by independent reviewers, working for ESA, but not directly in the project team. This raises the question of visibility that the reviewers should get on the models. It should be enough for them to detect flaws, but less than the one given to the project team. Another issue is the ability of the reviewers to understand models, and the appropriate packaging of these data for reviewers. For example, reviewers could have only a web interface access to the model, supported by documents giving the context of the model, the "user story".

- security: even with the best trust relationship, informatics exchanges are subject to security threats, and it is vital for companies to implement security measures at all levels. This requires a threat analysis, in order to decide where the protections must be placed. The design of the IT platform is generally necessary to understand where the threats are.

These topics need to be addressed at a different level than the pure technical level. They are discussed in the space community in the Data Management working group, a sub-group of the Digital Spacecraft Think Tank (Figure 1).

4. STATUS IN ESA PROJECTS

To date, digitalisation is mainly deployed in ESA projects through the Model Based System Engineering (MBSE) techniques. It addresses the functional part of the system with some links to physical architecture. But the link with the physical world (CAD, testing, etc.) is not yet achieved. Figure 7 shows the ESA projects implemented with MBSE.

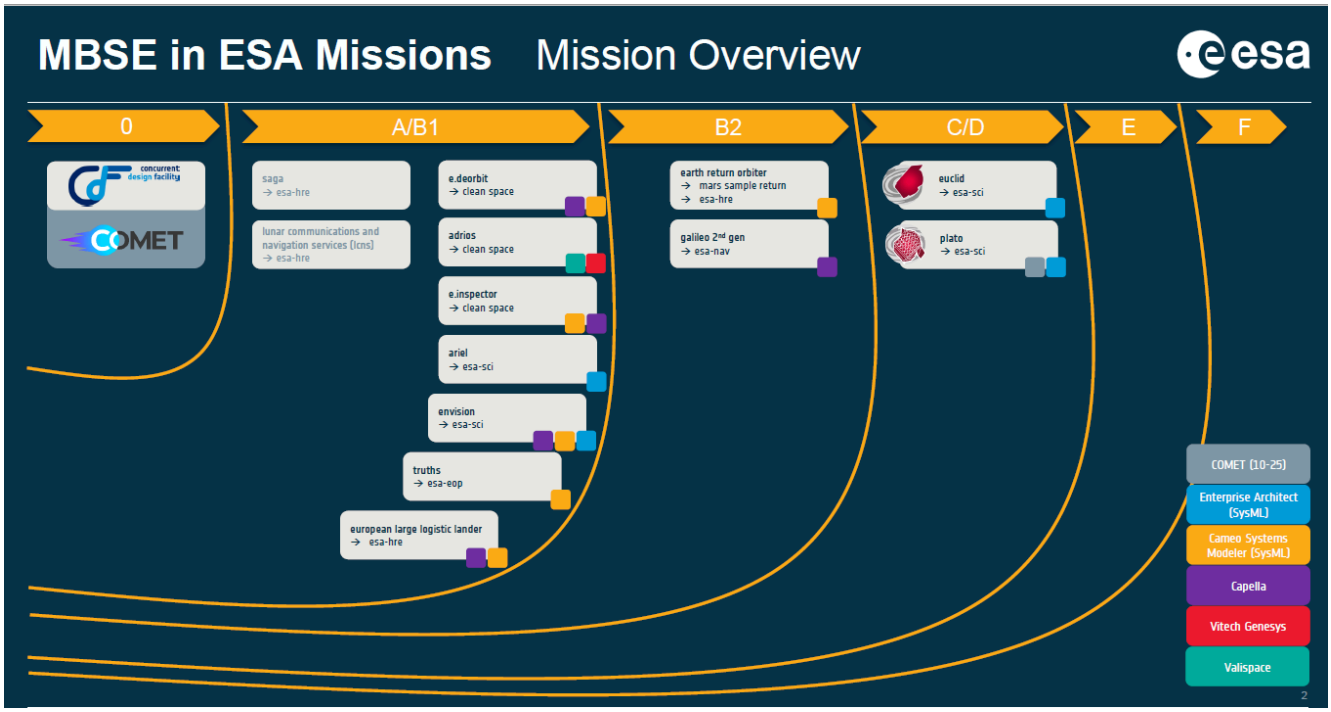


Figure 7 Status of MBSE introduction in ESA projects

MBSE was first introduced in Euclid, thanks to the presence in the team of a SysML guru who needed to solve a particular problem: the payload operation planning. He shared a preliminary model with the scientist community, and they found beneficial to collaborate on it. The model was further developed to include the payload architecture, then the platform/payload interface, etc.

MBSE is now introduced in the beginning of the life cycle, as early as the CDF study in phase 0. Supported by ESA, models are then developed, by ESA or industry, in phase A and B1 and then propagated to phase B2.

ESA is working with Large System Integrators, which uses different MBSE technologies. Tools like Capella, Cameo, Entreprise Architect or Genesys are used for similar purpose of functional description. As said before, it is impossible to impose the same tool to all. For its own purpose, ESA will favour SysML (with a particular profile called “MBSE Space”) and Cameo, although Science projects like Euclid and Plato are based on Entreprise Architect. However, ESA is able to work with any of these tools. A competence centre has been established to provide licences, tool support and training to projects.

In order to prepare the digitalisation of projects, a large R&D plan over 5 years has been established, in a Harmonisation process within Member States, addressing methodology, ontology, tooling, and project deployment. A low Technology Readiness Level call for ideas [18] has been made, resulting in the funding of 22 activities. The difficulty resides in the mandatory coordination of all the R&D activities of ESA and industry towards the same goal of interoperability and digitalisation.

5. TOWARDS DIGITAL TWINS

The availability of digitalisation enablers open the way to the exploitation of data. System engineers have access to dashboards of key parameters offering a synthetic view of the system, allowing design optimisation, traceability to reply to such questions as “What if this equipment is not compliant?” or “what if I introduce this change?”.

Further, the measure of the real values of the data on the physical items can be fed back into the models used for the design (this is called “model correlation”), allowing to sue better models in the next project (thermal models, structural models, environmental models, control models, etc.).

The same applies to simulators, which –when fed with the real data of the physical item measured during operation – becomes dynamically representative of the physical item, and are called digital twins. Twins of avionics equipment may be used to anticipate software verification before the availability of the hardware. With proper mechanical sensors inserted in the structure (e.g. with use of Advanced Manufacturing methods), a twin of the structural model can be produced. Finally, the connexion of the operational simulator to the real telemetry downloaded from the spacecraft allows having a spacecraft twin, representative in real time of the real one, and much closer to the one located at hundreds or millions of kilometres… Failure investigation and operation optimisation and prediction are example of capabilities enabled by the digital twin.

Beside the Spacecraft related twins, ESA (through the Earth Observation directorate), has launched the Digital Twin Earth [19]. It will help visualise, monitor and forecast natural and human activity on the planet. The model will be able to monitor the health of the planet, perform simulations of Earth’s interconnected system with human behaviour, and support the field of sustainable development, therefore, reinforcing Europe’s efforts for a better environment in order to respond to the urgent challenges and targets addressed by the Green Deal.

Similar twins for e.g. scientific data of the universe, or human data of astronauts, are investigated.

6. CONCLUSIONS AND RECOMMENDATIONS

Empowered by the Director General, in line with the strategic technology objectives, ESA has embarked in the change process to digitalisation. ESA is not alone. The change is supported by, and performed together with, the whole space community, national Space Agencies, and Industry.

Aiming at the digital continuity in space projects thanks to the interoperability of tools, a step-wise approach is planned. Many local progresses have been made, but the dots need to be connected. Several discussion forum are organized, allowing Agencies and Large System Integrators to progress together within industrial or programmatic constraints. The next step is to involve better the low end of the Supply Chain.

The main challenges are to maintain the synchronisation of the whole process, to keep awareness by dissemination, to convince gradually all the community without imposing, to ensure the communication of the progress and difficulty between all the actors such that we can solve the problems together, and to survey any opportunity to introduce digitalisation.

Beside the technological challenges, data management and governance is an essential topic to be addressed together, without which the whole initiative could fail.

But the success already observed, and the positive feedback received by project managers and industry, indicates that we are in the right direction. Let us get digital!

7. REFERENCES

[1] ESA AGENDA 2025: Make space for Europe - https://download.esa.int/docs/ESA_Agenda_2025_final.pdf
[2] https://www.futurebridge.com/industry/perspectives-mobility/digitalization-in-automotive-industry/
[3] https://www.thalesgroup.com/en/germany/magazine/digital-transformation-railway-industry
[4] https://www.aerosociety.com/news/aerospace-digital-transformation/
[5] https://www.designingbuildings.co.uk/wiki/Digitalisation_in_Construction
[6] https://www.esri.com/about/newsroom/arcuser/digital-twin-helps-airport-optimize-operations/
[7] https://www.incose.org/incose-member-resources/working-groups/Application/healthcare
[8] https://www.eurocontrol.int/digitalisation-and-information-management
[9] https://www.ecss.nl
[10] https://www.comet-cnes.fr/evenements/journee-mbse
[11] https://mb4se.esa.int
[12] https://mb4se.esa.int/OSMOSE_Main.html
[13] https://mb4se.esa.int/DtSC_Main.html
[14] http://www.orm.net/
[15] https://www.esa.int/Enabling_Support/Space_Engineering_Technology/Concurrent_Design_Facility
[16] http://products.rheagroup.com/comet
[17] Digital Engineering Hub Pathfinder - Extending the ECSS-E-TM-10-25 Tool Ecosystem A. Vorobiev1*, S. Gerené2, N. Smiechowski1 , S. Jahnke3, J. Knippschild3, S. Weikert4, M. Becker4, S. Paquay5, J-P. H. Vogt5, I. Fontaine5 1RHEA Group, Diegem, Belgium; a.vorobiev@rheagroup.com 2RHEA Group, Leiden, The Netherlands 3OHB System AG, Bremen, Germany 4Astos Solutions, Stuttgart, Germany 5Open Engineering, Angleur, Belgium https://indico.esa.int/event/386/contributions/6290/attachments/4286/6498/1645%20-%20digital%20engineering%20hub%20pathfinder.pdf
[18] https://ideas.esa.int/servlet/hype/IMT?documentTableId=45087661363758399&userAction=Browse&templateName=&documentId=a1fecb3f3789ea0eabe29e23c846fbfd
[19] https://www.esa.int/Applications/Observing_the_Earth/Working_towards_a_Digital_Twin_of_Earth

Space organisations landscape		
Space Agencies		
ESA	European Space Agency	https://www.esa.int
CNES	Centre National d'Etudes Spatiales	https://cnes.fr
DLR	Deutsches Zentrum für Luft- und Raumfahrt	https://www.dlr.de
ASI	Agenzia Spaziale Italiana	https://www.asi.it
UKSA	United Kingdom Space Agency	https://www.gov.uk/government/organisations/uk-space-agency
Space Industry		
ADS	Airbus Defence and Space	https://www.airbus.com/en/products-services/space
OHB	Otto Hydraulik Bremen	https://www.ohb.de
RUAG	Rüstungs Unternehmen Aktiengesellschaft	https://www.ruag.com
TAS	Thalès Alenia Space	https://www.thalesgroup.com/fr/espace

Impact of environment on the execution of a real-time Linux process on a multicore platform

Thomas Beck*, Frédéric Boniol†, Jérôme Ermont‡ and Luc Maillet*
*Airbus Defence and Space, Toulouse, France
Email: thomas.t.beck@airbus.com luc.maillet@airbus.com
†ONERA, Toulouse, France
Email: frederic.boniol@onera.fr
‡IRIT-ENSEEIHT, Toulouse, France
Email: jerome.ermont@toulouse-inp.fr

I. Introduction

A. Space context

In the embedded systems industry there are not environments more hostile than space. In fact, a spacecraft is alone in the void without direct human interactions. Its only way to communicate and receive orders is through antennas. In such a context, consequences of software failures can lead to the loss of the spacecraft or even worse. For example, if the AOCS (Attitude and Orbit Control System) software does not work, the solar panels will not always be facing the sun and the battery will not be able to produce enough power for the whole spacecraft. In order to reduce those risks, on-board software applications have a criticality level which represents the consequences of their failures. Therefore, a failure in a high criticality level software will result in worse consequences. In the space context, criticality level for software is defined in the ECSS (European Cooperation for Space Standardization) standards as follows:

- A Catastrophic consequences: loss of human life or environment disaster.
- B Critical consequences: loss of the spacecraft and/or the mission.
- C Major consequences: major mission degradation.
- D Minor consequences: minor mission degradation.

The development of a software with a high level of criticality is more constrained and thus more expensive. Usually, to prevent that a software failure propagates to a software with a higher level of criticality, software applications are executed on different hardware platforms. Most of the time, on-board software applications with the same level of criticality are not made by the same software team, and are separated on different hardware platforms as well. With this approach, software failures are contained by the hardware and software behavior does not affect execution of others software. Clearly this one-software on one-hardware-platform strategy has a non negligible cost considering the size, weight and power of an on-board computer.

B. Problem statement

This article aims at studying the cohabitation of two or more software applications on the same multi-core hardware platform. These two software applications are designed and developed according to the space context described in the previous section, thus each software application has a criticality level and is produced by a different developers team. Once they are executing on the same hardware platform, we want to assure two fundamental properties:

- **Execution interferences:** the perturbation made by one software on to others should not be greater than ϵ. This ϵ can be an execution time or a response time depending on the case. This epsilon is defined for each application and is part of its specification.
- **Failure propagation:** a software failure should not propagate to software with a higher criticality level. The functional failure propagation, being application-specific is not in the scope of this work.

Along with these properties we assume the following hypothesis: all software should be developed and executed on Linux. It means that Linux is used as the embedded operating system of our spacecraft. In order to ensure these two properties, our objective is to use space and time isolation of applications running on a Linux operating system. The use of Linux is motivated by its recent evolution that makes it useful for embedded systems.

C. Related work

In 2000, [1] presented core issues of IMA (integrated modular avionics) which was new at that time. In this technical report, John Rushby laid the groundwork of avionics partitioning showing that a strict isolation is a solution for resolving the IMA problematic. Many years later, solutions have been found to adapt this resolution in space avionics as shown in [2]. Hypervisors such as Xtratum and VxWorks 653 have been developed to respond to the needs of an IMA for space. Isolation proposed by this kind of hypervisor is strict, secure and with a temporal predictability. The disadvantage of those isolation properties is that it makes the whole system less modular. Scheduling is fixed , developers coding libraries are specific and not widely known and the development process is more complex. A performance comparison of these real time solutions is presented in [3].

Linux could be the perfect solution to the problems previously cited and it is used in many space mission as described in [4]. However, Linux is not a hypervisor designed to

strictly isolate and to schedule real-time software applications. Linux and its PREEMPT-RT patch is able to provide real-time features. For hard real-time software applications, a co-kernel solution named Xenomai is available. The performance differences between native Linux and Xenomai are presented in [5]. However Linux is not deterministic and its behavior can introduce latency in the system. A lot of studies measured this latency with the PREEMPT-RT patch ([6] [7] [8]). Moreover, some studies worked on the scheduling model of Linux to reduce these latencies ([9] [10] [11]).

The other aspect of using Linux in a critical real time system is to ensure an isolation of the different applications running on it. This concurrency problem is explained in [12] where they try to reduce the impact by implementing an RCU mechanism. Others studies try to address the problem from the memory point of view and thus work on the hardware memory system ([13] [14] [15]). Containerization is a powerful feature of Linux which helps when an isolation is required. [16] presents a way to modify the Linux scheduler to use container and real-time scheduling at the same time.

Linux is a powerful operating system and is useful in a lot of industries. In the space industry the interest has recently grown with the new space. The last Linux space known project was made by NASA. They sent a helicopter on Mars with an avionic partially based on Linux [17]. As explained in [18] Linux is also an academic subject to study.

Finally, our approach presented in this paper is based on the measurements like in [19]. We want to study Linux with the less customised configurations to take the more advantages of the open-source world offered by Linux.

D. Contribution of the paper

The main objective of this paper is to evaluate the robustness of Linux in a critical real-time context. This paper aims at producing tests results of the Linux behavior and its processes when using an embedded space scenario. We are studying the impact of Linux upon real time processes by varying configurations of the system (cf sections VI-B, VI-C and VI-D). We will also focus on the impacts of running Linux real-time processes upon each other (cf section VI-A).

II. Space software usecase and platform

Software applications we study exchange data between physical sensors, physical memory and the ground station. Those software can be real-time, i.e with a deadline and/or a period, depending on the requirements of the sensor they communicate with. Since all software will be executed on top of a Linux operating system, a software application is represented by a Linux process. We took one space specific use-case software to study its isolation when it is executed with others processes on the same hardware platform.

A. AOCS: attitude and orbit control system

In a satellite, the AOCS software is responsible of controlling the attitude i.e orientation and the orbit of the spacecraft. Inputs of AOCS algorithm are attitude and orbit positions coming from sensors such as Star trackers. The AOCS algorithm generates commands sent to actuators such as gyroscopes and thrusters. These commands aim at correcting the attitude and orbit of the spacecraft. In the scope of this article we study a simple AOCS algorithm doing a flyby i.e approaching a stellar object without entering any orbit. The AOCS software application implementing the algorithm is a periodic Linux process running at a 8Hz frequency. It takes its inputs from a pipe and writes its outputs in another pipe. This software consists of 16000 calculus steps. Each steps doesn't have the same behavior as the others. However, if the inputs stay the same the behavior of the same step in two different execution of the software will have the exact same behavior. The AOCS process has two children processes described below:

- A non periodic Linux process reads inputs from the input file and writes it to the input pipe.
- Another non periodic Linux process reads outputs from the output pipe and compares it to the expected outputs from the output file.

The input and output files are located on the file system. The AOCS process is thus only responsible of computation which is its true behavior in the spacecraft. By forking the AOCS process and thus creating two children processes it creates an isolation between I/Os accesses and calculus code. Moreover, memory accesses made by the I/Os processes and the AOCS process are different. Reading and writing to a file imply loading memory pages from the disk to the main memory. This operation along with every others one related to the virtual memory are handled by the operating system and are slower than accessing the main memory. In another hand, reading and writing to a pipe is faster as it doesn't need memory page management because pipes are located in the operating system area.

B. Hardware platform

Due to the hostility of the space environment, spacecraft embedded computers and electronic devices must be adapted. Those modifications have non negligible costs on satellites production. Recently, the space industry introduced COTS (commercial off-the-shelf) hardware components to be used in the next generation of satellites. Experiments presented in this paper are made on a Zynq Ultrascale+ made by Xilinx which is one of the COTS hardware platforms considered by the space industry for the next generation of satellites.

The SoC has two main processing units (application and real-time) along with a PMU (platform management unit), a CSU (configuration and security unit) and a GPU (graphical processing unit). The Zynq ultrascale+ also contains an FPGA which can be accessed by all I/Os and processing units. This SoC architecture is interesting for the space industry, the application processing unit can run COTS software (such as Linux) while the real-time processing unit handles more critical software or monitors the entire system. Moreover, FPGA is useful for handling communications between I/Os and processing units and is becoming an essential electronic component for satellites.

As described in the previous paragraph, in the context of space systems the real-time processing unit is envisaged to take measures. This unit is made of an ARM Cortex-R5 implementing the ARMv7-R architecture. It is a dual-core

with one L1 cache per core and a L2 shared cache. To ensure reliability it can be used in lock-step, i.e the two cores execute the same instructions and their outputs are compared.

Besides, the application processing unit will run Linux. This processing unit is made of an ARM Cortex-A53 processor with a frequency going up to 1.5 GHz, which implements the ARMv8-A architecture. It is a quad-core, with instruction and data L1 cache along with a shared L2 cache and an MMU (memory management unit). The L2 cache is a 1MB 16 way set-associative cache with ECC shared between the CPUs. It means that it's containing 15 625 lines of 64 bytes. The L2 cache is unified i.e it contains both data and instruction from the L1 memory system. The L1 instruction cache is a 32 KB 2 way set-associative cache with ECC independent for each CPU. It's containing 500 lines of 64 bytes. The L1 data cache is a 32 KB 4 way set-associative cache with ECC independent for each CPU. It can contain 500 lines of 64 bytes. In the scope of this paper, all software applications are executed on the processing unit.

III. Problem formalisation

Our work is focused on analyzing perturbations between software applications on the same hardware platform and respecting the properties described in the introduction of this paper. All software applications will be executed in a Linux process and processes will only contain one thread.

First of all, Linux hasn't been created to be a real-time operating system. As our software applications are connected to complex sensors they sometimes need to be executed periodically and respect a deadline. Linux developers developed real-time mechanisms for Linux. The central question of our work is to ask if real-time constraints of our use-cases can be respected with or without Linux real-time mechanisms. In this paper, we will provide some elements answering this question.

Second of all, the hardware part of our usecase is very important, as execution time can be modified by a memory response delayed by shared caches or busy busses. Interactions between software and hardware is made by two channels: systems calls and drivers. System calls allow software to use special features provided by the operating system. A complex operating system such as Linux proposes many different system calls and some of them might not be suitable for space avionics. In order to measure and then control interference caused by system calls the first step is to list system calls usable in a satellite avionic. Once the list is complete an analysis of the memory impact of each system call will be made. For example, if one software uses shared pipes they can be accessed by others software. These accesses can generate interference, thereby a space isolation of resources accessed through system calls is needed. In the scope of this paper, we consider all systems calls related to the file system, specifically the write and read system calls. In fact, using files is a useful feature for an on-board software.

The main purpose of drivers is to abstract the handling of devices. Drivers code runs in kernel mode like the operating system and can thus access the physical memory. As they are closely related to device characteristics most of the time drivers are written by the company who developed the device. This detail poses a problem in our context, the fact that a non trusted component is running inside the kernel is a clear breach of the isolation sought in this work. The last question of our work is, how to isolate shared software resources such as drivers in a Linux system. This question won't be addressed in this paper.

IV. Linux based solution

Linux is a widely used operating system. Created in the mid 90s Linux is developed by software engineers all around the world. A lot of companies invest time and money in the development of Linux. Nowadays, Linux is used by a lot of people and deployed on many different kind of hardware. It can run on desktop computers, servers, supercomputers but also on IoT and embedded devices. Most of developers learn to use Linux and its development environment which make it a standard in the computer industry. One of most powerful advantages of Linux is its reusability. The open-source philosophy allows to reuse libraries and drivers developed for Linux on different hardware. All these advantages make Linux useful in an embedded devices context. However, it is not conceived for critical applications and the development process of Linux is far from those used in real-time/critical operating system. Critical embedded industries want to be part of the Linux adventure and get all the benefits of using it in their products. Medical, aerospace, automotive, and industrial automation are considering the use of Linux for critical sections. This new approach of critical software asks a lot of questions due to the fact that qualification or certification of open-sourced Linux is certainly impossible. Thereby the embedded operating system verification paradigm has to be changed. In this section, configurable characteristics of Linux will be presented. These characteristics are appropriate for ensuring the properties described in the introduction.

A. Real-time scheduler: SCHED_DEADLINE

For Linux to be used in a critical embedded systems context it need to have real time properties. Thus, process needs to respect their deadline and period cannot be fulfilled with the default Linux scheduler. Nevertheless, Linux has multiple scheduling policies. Each Linux process can have a different scheduling policy and many of them are real-time oriented. In this article we will target the SCHED_DEADLINE policy. It allows a process to have a period and a deadline. Its implementation uses GEDF (Global Earliest Deadline First) in conjunction with CBS (Constant Bandwidth Server). To ensure an optimal scheduling, the scheduler needs to know the process runtime (cf figure 1). This runtime must be greater than its average computation time (or WCET for hard real-time). These 3 properties will be used by the scheduler to ensure the scheduling policy. CBS throttles threads attempting to over-run their runtime to guarantee non-interference between tasks.

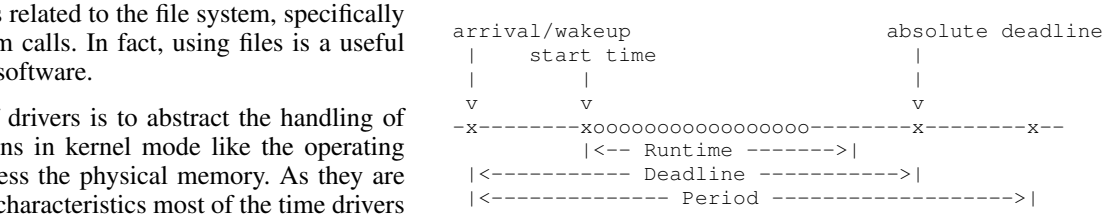


Fig. 1. Parameters of a SCHED_DEADLINE Linux process (man sched(7))

In the Linux system, SCHED_DEADLINE threads have the higher priority. In other words, if one SCHED_DEADLINE thread is runnable it will preempt threads scheduled by another policy.

B. Real-time kernel

Scheduling is not the only characteristic needed for an operating system to be considered as real-time. In Linux the PREEMPT-RT patch has been created to adapt the kernel to a real-time context. This patch is now merged into the upstream Linux stable version. As presented in the previous section, the Linux scheduling can be real-time and implement rt-throttling to ensure that no tasks hang the system. The PREEMPT-RT patch also enables the priority inheritance mechanism in the Linux scheduler. It allows a task with a low priority to take an higher priority if it is blocking a mutually exclusive resource. Thereby, the high priority task waiting for the mutually exclusive resource will be executed as soon as possible.

One important aspect of the Linux PREEMPT-RT patch is the preemption model. In the mainline kernel, plenty of the code is non preemptible which can delay the tasks execution time. To reduce the impact of this, it is possible to make the kernel in a fully preemptible mode. It means that all kernel code is preemptible (except a few critical parts). Large preemption disabled sections are split with locking constructs. Threaded interrupts handlers are forced i.e interrupts handlers run in a threaded context and new mechanisms are implemented such as rt_mutex and spinlocks which allows preemption in mutexes and raw spinlocks.

V. Methodology

A. Protocol

In this section we describe the protocol used to get the results presented in this paper. For each metric we want to retrieve two measurements. One measurement before calling the calculus function and one measurement after. Then, the difference between those two measurements is made to obtain the value of the metric during the execution of the calculus function. For example, to retrieve the number of L2 cache refill made by one execution of the AOCS calculus function, we measure the value of the L2 cache refill before calling the function and after calling the function. Then we can take the difference between those two values to get the number of L2 cache refill made during one execution of the AOCS calculus function. Note that the measurements are only taken on the AOCS process and not on the I/Os processes. It means that writing and reading from the pipes is not considered in the measurements. Also all measurements are taken on one process only which is considered the victim of the experiment. All others processes (users and kernel ones) and all kernel activities are considered the attackers of the system. In the ARM Cortex A53 used in these experiments performance counters registers are 32 bits wide. As we are running an AOCS process running at a frequency of 8Hz for 16 000 steps, the entire execution takes around 33 minutes. During such a long time, the 32 bits registers can overflowed, in this case the counts restart at zero. In most cases this is not a problem as we are taking the difference between two values. However, when the overflow happens during the execution of the calculus function, the value of the difference is not correct. To repair those values we add 2^{32} to each non valid value. The only measure taken before reading the input pipe is from the clock because one important metric to analyze is the wake up date of the process. The code of the measurements is shown in the figure 2.

```
for (i = 0; i < nbStep; i++) {
    clock_gettime(CLOCK_REALTIME, ...);
    read(pipe, ...);
    ...

    read_counter(i);
    clock_gettime(CLOCK_PROCESS_CPUTIME_ID, ...);
    Aocs_step();

    read_counter(i);
    clock_gettime(CLOCK_PROCESS_CPUTIME_ID, ...);
    clock_gettime(CLOCK_REALTIME, ...);
    ...
}
```

Fig. 2. C code of the measure point inside the AOCS process

We collected a lot of data from the performance counters, mostly on hardware memory events (bus accesses, cache accesses, cache refills, ...). In the scope of this paper, data such as context switches or core migration during the execution of an AOCS process could be interesting. Unfortunately, we weren't able to retrieve those data precisely enough. This is due to the fact that there is no way to know when a process is migrated or preempted. It is possible to get the number of the core the process is executed on, but nothing assures you that this won't change in the next moments. Thus, we aren't enable to get information about context switches or core migrations by retrieving hardware data.

In all these experiments we consider a "light" Linux distribution made with Yocto. The Linux kernel used in this paper is a 5.4 version of the Xilinx Linux kernel found in the official Xilinx Github [20]. According to Xilinx recommendations ([21]) we used the Yocto zeus version from the official Yocto repository ([22]). Using Yocto allows a complete theoretical control of what's inside the Linux operating system. In the scope of this article, each application is modelled by a Linux process and each Linux process have exactly one thread i.e threads = processes.

B. Measures impacts

Measure methods affect the experiment and therefore modify the results. Removing measure impacts is difficult and most of the time impossible. Therefore, the question is are those impacts negligible in these experiments ?

Every measures described in the scope of these experiments are taken inside a period of execution. It means that every code outside of the periodic loop isn't considered in the measures results. In particular, the loading and initialization phases of the application are not in the measures scope.

Measures presented in this article are coming from two different sources. The first one is clocks, managed by the operating system and used to measures execution time, relative

wake up time and processing time i.e time passed on the CPU between the beginning and the end of a cycle. Retrieving the value from those clocks takes approximately the same amount of time each time. The time value is encoded with two 64 bits integers, one for the nanoseconds and the other one for the seconds. Impacts of retrieving a timing value is then constant. The second source for the measures is performance counters. The activation of those counters doesn't modify the execution of the code. Retrieving the value of those counters imply reading CPU registers which aren't in the main memory. As for the clocks, the impacts of retrieving performance counters values are constant through the time. For both measures sources impacts of retrieving data are constant and don't affect the main memory i.e no memory accesses are performed. To be analyzed, those retrieved data need to be stored and made available after the experiment. The storage procedure can induce huge impacts on the measures, as the results have to be stored on the main memory. For example, the complete AOCS application has around 16 000 cycles which means that if all 6 performance counters and 2 clocks are used a storage space of $16000 * (6 + 2) * 64bits = 1MB$ will be needed. Accessing this space requires accessing memory pages which will generates memory accesses, page faults and other events related to the virtual memory which will be handled by the operating system. To reduce the effects of those memory accesses on the experiment, measures data is placed in local variable placed on the stack of the process and then copy to the memory outside of the measured section. This method will generate less impacts on the executed code but will alter a little the memory hardware state.

Finally, impacts of retrieving measures are constant and storage of the results is made outside the periodic loop. Therefore, to decide if measures impacts are negligible in this context we need to measure the constant part of the impacts. Figure 3 shows the results when no code is executed between two measures points. This chart represents the execution time induced by the measure itself which comprised between $7,5\mu s$ and $9,0\mu s$. As the WCET of the AOCS process is around 1ms we can conclude that the impact of the measurements is negligible.

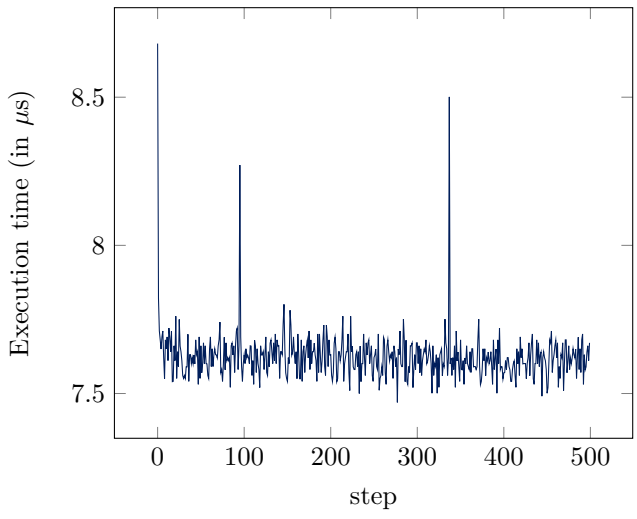


Fig. 3. Measurement of execution time with no code

VI. Experiments

Experiments presented in this paper aim at finding Linux configurations suitable for our context. Linux have a multitude of configurable points which can vary. To find suitable Linux configurations we will focus on varying those configurable points. For example, real time and isolation mechanisms such as the PREEMPT-RT patch or the Linux namespaces are configurable points which can vary from disabled to enabled.

In this section we present 4 experiments making measurements on different configurable points. The first experiment aims at comparing the use of same or different files by the same processes. The second experiment measures an impact of the Linux stock kernel on the execution time of an AOCS process. The third experiment measures a drift in the scheduler wake up date of a periodic process. The last experiment, is the same as the last two but with the PREEMPT-RT patch enabled.

A. Mono file versus multiple files

Files take an important place in our experiments. Not only they are used to store inputs and outputs of the AOCS application, they also contain instruction data. In fact, executables are files stored in the disk (SD card in our specific case). The file management is completely handled by the operating system, through multiple system calls, which try to optimize these accesses. In this context, accessing files is a source of interference from the operating system on applications but also from other applications. For example, two different applications trying to read to the same file will generate interference and the behavior will be different as if the application was alone on the system. The first experiment presented in this paper aims at exhibiting the differences between applications using the same files and applications using completely different files. This configuration could happen when using shared libraries as it is very common is the Linux development world.

In this experiment, two groups of applications were created based on the AOCS code. The first group is composed of AOCS applications using the same executable file and the same I/Os files. For the second group each executable and I/Os files are duplicated. For each group the same experiment was performed and is described as follows. The experiment begins by executing one AOCS application and measuring execution time, process time, wake-up time and performance counters related to memory such as L2 cache refill or bus accesses. The results from this first step is used for comparison and must be equivalent in both groups. The next step of the experiment is to increase the number of parallel execution. The maximum number of launched applications is 30 which represents 90 processes on the operating system (30 AOCS processes, 30 input processes and 30 output processes). In the context of this experiment, measurements are taken on only one AOCS process considered as the victim whereas all others AOCS and I/Os processes are considered attackers.

In the results of this experiment we observed that when the files are the same the performance are better on average. The worst performance is better than for the other group of processes. The figure 4 shows the average execution time for the AOCS process when others AOCS processes are executed in parallel. There is one curve for the first group and the other curve is for the other group. In these curves we can notice

aberrant points as for example, the point 3 of the mono file curve. Linux isn't deterministic and induce a lot of variability in the system, this is why we may have these points. In this paper we are focusing on the curves trends and not the particular values. Note that we did repeat those experiments a few times and these aberrant points aren't always at the same spot. Although, we do not have enough data to make statistics which could lead to removing these points.

This chart shows us that in average the group using the same files is executed approximately two times faster. Moreover, both curves can be separated in two phases. The first phase is between 1 and 4 AOCS parallel while the second phase is from 5 to 30 AOCS in parallel. In the first phase, both curves are increasing and in the second phase they seem constant. The hypothesis behind these observations is that the pivot point of 4 AOCS processes in parallel is related to the cores number of the ARM Cortex A53 the processes are executed on. In fact, many memory hardware resources are shared between the 4 cores of the processor (buses, L2 cache, memory controller, RAM,...). When 2 processes are executed in parallel, they both need to access those resources and thus hardware resources aren't always ready to respond to the cores requests. When executing more than 4 processes in parallel, cores are always occupied. If it's not by AOCS processes they will be by inputs or outputs processes. This means that there is always 4 processes running at the same time and thus memory hardware resources are busier than if there were less than 4 processes in the system. Parallel use of the hardware resources can explained this observation, but these are not the only hardware interference possible in the system.

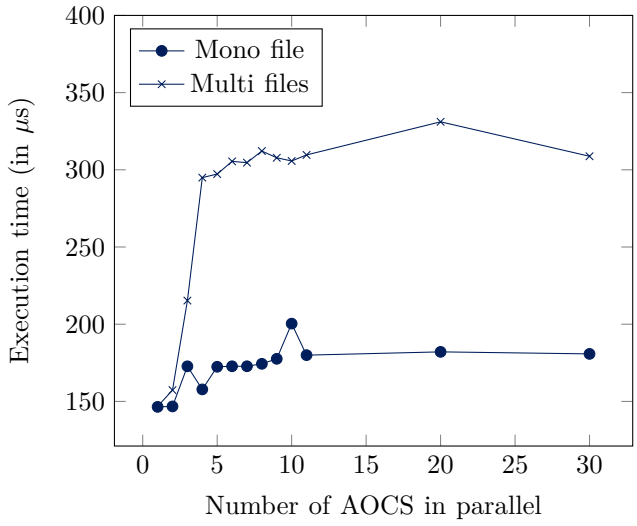


Fig. 4. Measurement of execution time in average comparison of the two processes groups

Modifying the state of a shared memory hardware device is another kind of interference. The figure 5 shows the number of L2 data cache refill i.e the number of times a cache line has to be loaded from the main memory, from the same two groups of processes. As for the previous curve, it has been obtained by executing the AOCS processes in parallel during 16 000 steps at a frequency of 8Hz. The curve is similar to the previous one, but the form of it around the pilot point is smoother. As a matter of fact, we can observe a behavior change around the point 4 but the transition between the two phase is not abrupt. In this case, we still can see an increasing between the point 4 and 8 in both curves but the slope of the curves is smaller and the shape of the curves changed from exponential to logarithmic before becoming constant. In other words the first phase of the curves (between 1 and 4) is similar to the previous curves and thus induces the shape of the execution time curves. In fact, if there are more L2 data refill the execution time of the process will increase. Then, the second phase shows a smaller rise of the curves because increasing the number of parallel processes means increasing accesses and more accesses means more L2 cache refill. The last question raised by those charts is why is the second phase constant ? At first, it does not seem logical that the impact on an AOCS process is the same when there are 10 others AOCS processes executed in parallel and when there are 30 AOCS processes executed in parallel. An hypothesis that could explain this behavior is that the cache could be filled up or at least all the ways accessed by the AOCS process are filled up with other data. In this case, each time the AOCS process is scheduled its data from the last execution has been evicted from the cache.

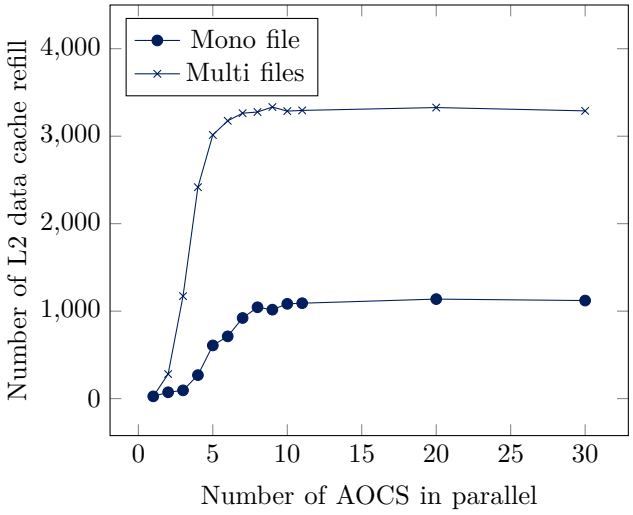


Fig. 5. Measurement of L2 data cache refill in average, comparison of the two processes groups

Now that the shape of the curves is described, let us focus on the differences between the two groups of processes. In both charts we can see a non negligible difference between the mono file curve and the multi files one. The mono file curve shows less L2 cache refill with a maximum of 1000 while the maximum of data cache refill for the multi files curve exceeds 3000. This means that when using the same files, there is a better re usability of the data which can be found in the caches. But address space of different processes in Linux are completely separated. In other words, data from a process A cannot be shared with a process B and thus process B cannot use data from the cache already loaded by process A. Then how can we explained the previous results?

When accessing a file in Linux a process uses system calls. All actions related to a file is then handled by the operating system. Even if there are 30 different processes executing in parallel with different address space, they are all taking their

data from the same file. All reads and writes are performed by the operating system from the kernel address space. Which means that all processes reading or writing to the same file can used the same cache line without needs to refill it.

To conclude, using the same files both as executable and data I/Os for parallel processes change drastically the performance we observed. We showed in this sub section that the impact of using or not the same files in parallel execution of processes is impacted by two distinct phenomenon.

B. Periodic interference peaks

During the first experiment we observed that when an AOCS process was executed alone on the system its execution time was impacted by interference. In fact, periodic peaks are present in the execution time curve as shown in the figure 6. We investigated to know if those peaks comes from the application code itself or if it is interference generated by the operating system. In this experiment, there are no difference between mono or multi files as we are only considering one AOCS process.

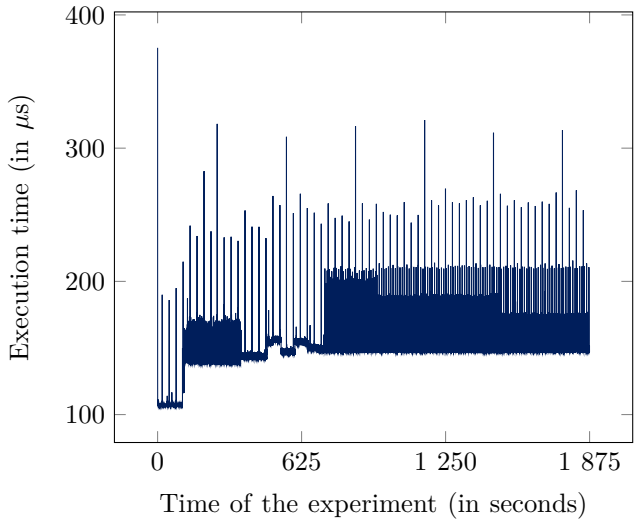


Fig. 6. Measurement of execution time for each step of an AOCS application with a period of 125 ms without any other application running in parallel

To determine the origin of those peaks two observations were made. First, the peaks don't appear at the same steps for every execution of the AOCS process. We know that the AOCS code we use doesn't have the same behavior for each step but one step will always execute the same code if its inputs are the same. In the scope of these experiments, I/Os files aren't modified. After this observation, the hypothesis that the peaks do not come from the AOCS code itself seems plausible. To assure that we had the right hypothesis we try to redo the measurements with a modified period. In this case, the period of the peaks didn't change. The figure 7 shows the results for a period of 50ms. In fact, there are 20 peaks in 625 seconds in both curves.

It is clear that the period of those peaks isn't based on the steps of the AOCS application. Therefore, we can now assure that those peaks come from the operating system. Note that these peaks also appear in the figure 2 which also confirm that the interference don't come from the AOCS algorithm.

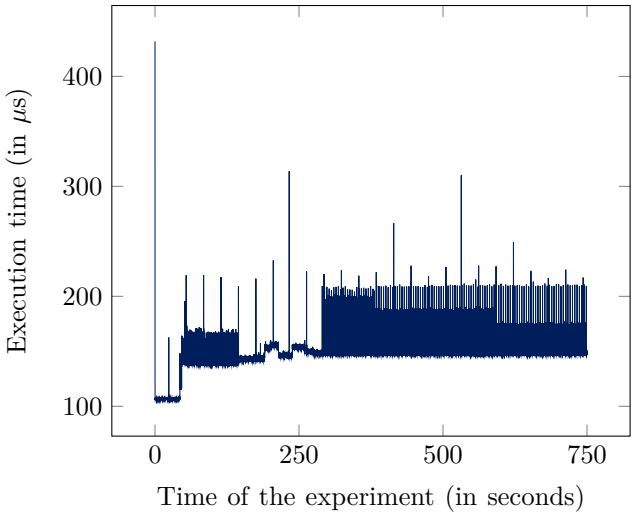


Fig. 7. Measurement of execution time for each step of an AOCS application with a period of 50ms without any other application running in parallel

One other simple hypothesis could be that those peaks come from a context switch. To answer this question an analyze of the difference between processing time and execution time can be made. The execution time counts the time spent by the processor when executing only the AOCS code. The processing time is the difference between the end date and the beginning date of the process. Therefore, if a context switch occurred during the execution of the process, the processing time will be greater than the execution time. We observed that there is no correlation between this difference and the peaks observed in the execution time chart. In fact, the processing time is not greater when there is a peak in the execution time curve. The figure 8 shows the result of the difference between processing time and execution time. Therefore, this hypothesis can be excluded because there are no peaks in this curve.

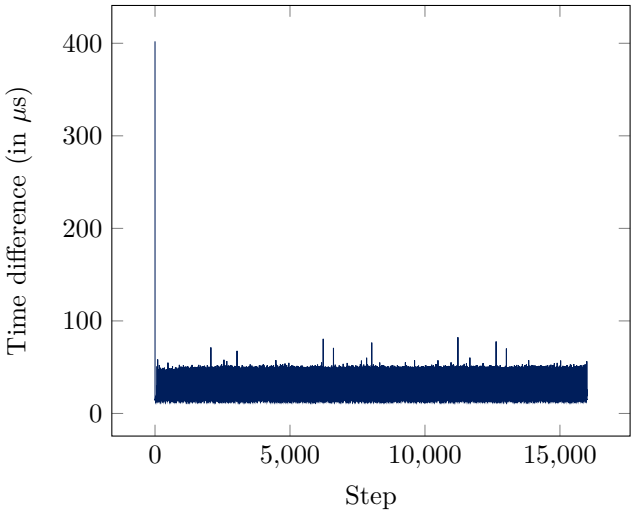


Fig. 8. Measurement of the difference between processing time and execution time.

Now the question is to find where the peaks come from. We can note that there are two types of peaks. There are smaller peaks with smaller period and bigger peaks with bigger period.

We could also note that there are exactly 9 small peaks between two big peaks. It seems that 2 periodic phenomena impact the AOCS process. There are 2 hypothesis for the second phenomenon with the bigger peaks. Either it is a particular execution of the first phenomenon occurring each 10 peaks. Or it can be from a completely different source and the 2 phenomena are synchronised for some reasons.

The measures taken from the performance counters are difficult to analyze because the AOCS algorithm doesn't have the same behavior at each steps. In particular, it doesn't make the same amount of memory accesses and doesn't access the same data in the main memory. Thereby, it is hard to know if the AOCS algorithm play a role in the value of those peaks or if they are only induced by the environment.

This questions lead to create another algorithm to help us find the source of the observed peaks. The new application is reading from an integers array stored in memory. To ensure that the compiler won't remove those unused accesses a sum of each integer of the array is performed. Moreover, this sum is repeated periodically at a frequency of 8Hz and uses the SCHED_DEADLINE policy from the Linux scheduler. We also implemented the same measurements techniques as for the AOCS process. The measures from this new algorithm give the same peaks in the execution time data. This confirmed once again that the operating system is altering the behavior of the process. What's interesting in this case is that the memory accesses and the L1 data accesses are constant at each steps. Each new refill from the L1 and L2 data cache can then be considered an interference from the operating system. As shown in the figure 9 there are also peaks in the L2 data cache refill. Those peaks occur at the same steps as the one observed in the execution time curve.

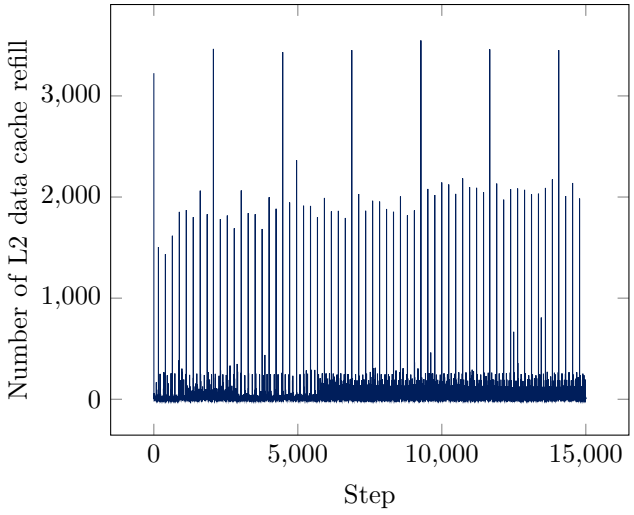


Fig. 9. Measurement of L2 data cache refill for the AOCS application

At this point the main hypothesis is that Linux or a hardware component is flushing all the memory hardware system (L1, L2 and TLB), either because it makes a lot of accesses to the memory or because it performs a hardware flush (for example because of security reasons). After this action is performed the process is woken up and have to refill all its data in the memory system which takes time and generates the peaks we observed. This hypothesis also explains why the curve which shows the difference between processing and execution time presented in the figure 8 is not constant. In fact, as the process as to access the main memory the CPU has to wait, when the CPU is waiting the real time clock is running but not the one counting the time passed in the process. Note that these peaks are observed even for very small array. It means, that even when the process has only a small amount of data in the caches, the operating system ejects it. This also confirms the hypothesis of an entire flush of the hardware memory system.

Actually kernel code responsible of this flush phenomenon hasn't yet been identified. An analysis of the behavior of the system with the Linux Ftrace tool has been made but no correlation has been found.

C. Scheduler wake-up drift

In the embedded space system the wake up date of the periodic process is an important metric to measure. If the scheduler wakes up the process a little later or a little sooner it can induce a drift which will result in a desynchronization of the process and the sensors sending input data. This desynchronization could become a problem, especially since satellites are running for years without being rebooted. We measured the wake up date of the AOCS process with the CLOCK_MONOTONIC of Linux which is a system wide clock. The chart presented in figure 10 shows the relative wake up date (in ns) at each step. The first wake up date is taken as a reference. The rest of the curve is obtained by subtracting $step * 1.25 * 10^8$ at each wake up date. It shows that after 15 000 steps the wake up date is 40 μs sooner that it should have been if the scheduler used an exact period of 125.00 ms.

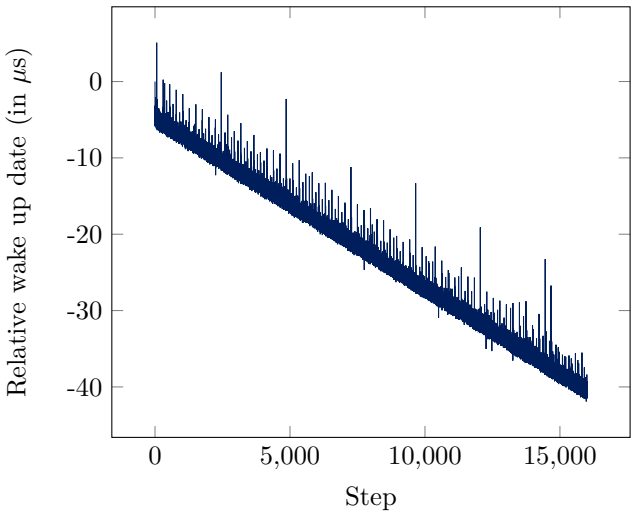


Fig. 10. Measurement of the wake up date for each step relatively to the first wake up date

The first observation made on that curve is that the scheduler seems to induce a drift in the wake up date. One hypothesis to explain this drift could be a desynchronization between the internal scheduler clock and the clock monotonic used in this experiment.

The second observation is that the peaks presented on the section B also appears on the wake up date curve. Those peaks are similar in the figure 10 and in the figure 6 even though they came from different execution. The scheduler drift seems to be linear but can be impacted by the same phenomenon than the AOCS process itself. We observe that on the same execution measures the peaks appear exactly at the same steps on both curves.

To help find what can induce this drift we look at the difference between two consecutive wake up dates. The figure 11 shows for each step the difference between the wake up date of this step and the wake up date of the step before minus $1.25 * 10^8$. In other words, a value of 1 at the step 50 means that the difference of the wake up date between the step 50 and the step 49 is 125ms + 1 μs.

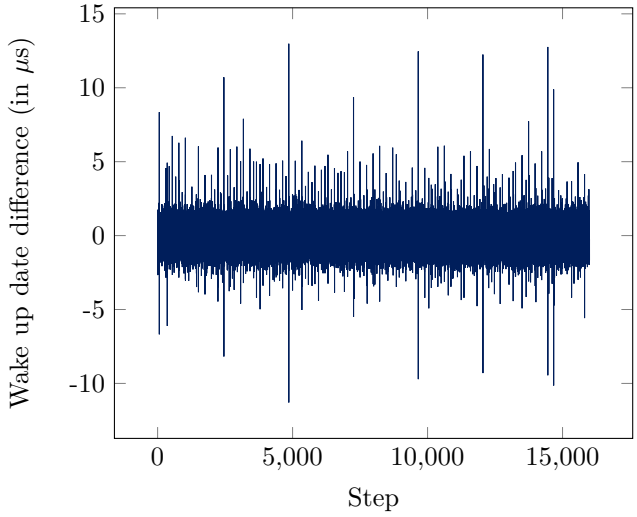


Fig. 11. Measurement of the wake up date difference for each step

In this curve we can remark that there are also peaks. As for the other curves, the peaks appear on the same steps in all different curves. This means that the two phenomenon producing those peaks both in the scheduler and the AOCS process are correlated. Moreover in the figure 11, the positive peaks seem to be correlated to negative ones. The figure 12 shows the same curve but only the first 100 steps. At the step 61 we could see a positive peak of around 8 μs. This peak is followed right after by a negative one at the step 62 of around -6.5 μs. The hypothesis behind this observation is that the scheduler try to catch up its delay at the step 61 by scheduling the next step a little earlier. But the positive delay at the step 61 is greater than the negative delay at the step 62. As a result the scheduler induces a positive delay at the step 63. By looking closely at the figure 11 we observe that this phenomenon is reproduced for each positive and negative peak. Another hypothesis could be that the time spent at executing the operating system code isn't taking into account for calculating the next period.

We can then generalise the rationale for the steps 61 and 62. Thus the scheduler induces a positive delay for each peak which should result in a positive delay in the entire measurement. However, the figure 10 shows that globally there is a negative delay.

Finally measuring wake up dates of the AOCS process showed two things. First, the phenomenon impacting the execution time presented in the last sub section also impacts the scheduling of the process. This impact induces a positive delay in the wake up dates. Second, the wake up date are globally drifting and the process at the step 15 000 is scheduled sooner than it should have been.

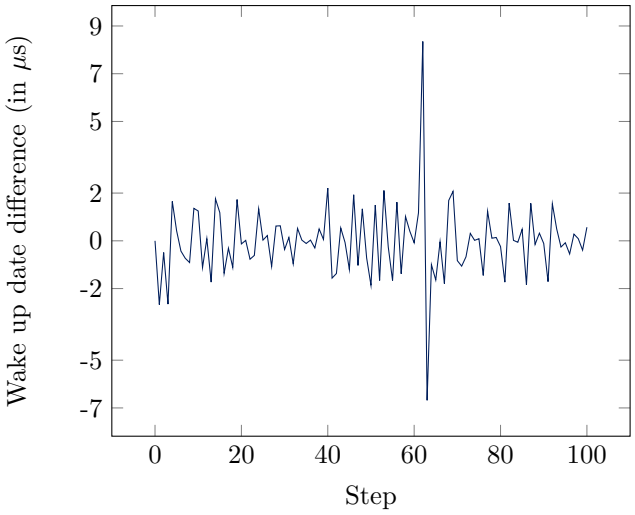


Fig. 12. Measurement of the wake up date difference for each step for the first 100 steps

As explained in the introduction of this sub section, this drift could be a problem if the system is not rebooted very often. This raises the followed question: does the phenomenon result from a desynchronisation between two internal clocks or a software behavior of the scheduler itself?

D. Impact of PREEMPT-RT

One of the first question when Linux is proposed in a critical embedded context is: is the PREEMPT-RT patch of Linux necessary ? We reproduced the 2 last experiments with PREEMPT-RT to find out if the real time patch reduces the observed phenomenon. The next paragraphs describe the obtained results for each experiment.

For the periodic peaks experiment adding PREEMPT-RT doesn't change the actual behavior. The peaks still appear in the execution time curve and are correlated to the peaks in the L2 cache refill curve. However, our observations are described as follows:

- The average execution time is the same with and without the PREEMPT-RT patch enabled. Globally, for each step the execution time isn't modified by PREEMPT-RT. We measured an average execution time of 145 393 nanoseconds without PREEMPT-RT and an average execution time of 146 429 nanoseconds with the PREEMPT-RT patch. This means that there is a difference of around one microsecond between the two average execution time.
- The bigger peaks have the same value with and without the PREEMPT-RT patch enabled.

- The smaller peaks are higher with PREEMPT-RT than without.

The first observation means that the PREEMPT-RT patch doesn't add any overhead to the execution time. However, it seems that the PREEMPT-RT patch induces a phenomena which results in a higher impact on the AOCS process. We don't have any hypothesis at this time of what causes this impact since we aren't sure of what causes the peaks in the first place.

For the scheduler wake up date drift the curves from both cases seem to have the same characteristics. In fact, the drift is still there and its slope is the same. The patch PREEMPT-RT doesn't have any impact on the wake up date drift.

Finally, in these two experiments, adding the PREEMPT-RT patch doesn't seem to modify the behavior except a small change in the heights of the smaller peaks in the execution time curve. In fact, using the SCHED_DEADLINE scheduling policy from Linux makes the AOCS process a high priority process. Without a lot of external interruptions the AOCS process isn't bothered very often. The scheduling policy is then enough for our process to be efficient.

VII. Conclusion and perspectives

A. Conclusion on the observation

The goal of this paper was to observe how the Linux kernel impacts the execution of real-time processes.

The first experiment showed us that system calls provided by Linux to access files seem to be optimised to increase the performance when multiple processes want to access the same files. In our experiment, using shared files produces less interference than using different files when reading the files even if the address space of processes are completely isolated by the operating system.

The next two experiments presented the impacts of the stock Linux kernel on the AOCS process. In the first experiment, we have periodic impacts increasing the execution time. In the second one, we found a drift in the wake up date of periodic processes. Both of these phenomena can be a problem depending on the chosen ϵ mentioned in the introduction. For instance, in the AOCS figure 6 shows that perturbations induced by Linux double the execution time. In this case, if ϵ (i.e margin associated with AOCS) is lower than the isolated execution time then this Linux-based implementation doesn't meet the requirements. However, the impacts described in those two experiments is constant and predictable through time. In fact, in all the measurements we made we found the same impact in both experiments. The execution time peaks are periodic with a constant period. For the scheduler wake up date drift, the slope is constant and around -40 microseconds for 16 000 steps ($\tilde{3}3$ minutes).

Finally, we did the same two experiments with another variation of our Linux kernel. In this case, we enabled the PREEMPT-RT patch to find if it has an impact on the two observed phenomenon. Only a small change in the value of the small peaks has been spotted but the global behavior didn't change. Thus, the PREEMPT-RT patch doesn't seem to impact the observed phenomenon.

B. Future work

For the future, the first thing we want to do is find the root cause of the two observed phenomena. In order to do that, we could add statistics tools to help understand the measured data and extract information on the phenomena. In this paper we focused on a qualitative rationale, introducing statistics will help for producing a more quantitative rationale. Another approach to find the root cause could be to use tools provided by the kernel itself such as *ftrace*. This tool will provide kernel information on what is actually executed by the kernel. Using kernel tools could also give information on core migrations and context switches.

The SCHED_DEADLINE policy used in this paper is based on the EDF algorithm. A future work could be to find a link between the scheduler wake up date drift and the formal equations used to code the SCHED_DEADLINE scheduler.

In this paper, we focused on the AOCS algorithm, in the future using other space applications will be useful to find other interference and characterize the already observed ones.

Finally, we would like to introduce containerization configuration in our kernel such as namespaces and cgroups. In fact, the goal of our work is to reduce interference between multiple applications and their environment in a Linux context. Linux provides isolation configuration and we would like to find if these tools could help seeking a certain isolation in our context.

References

[1] J. Rushby, "Partitioning in avionics architectures: Requirements, mechanisms, and assurance," SRI INTERNATIONAL MENLO PARK CA COMPUTER SCIENCE LAB, Tech. Rep., 2000.

[2] J. Bredereke, "A survey of time and space partitioning for space avionics," 2017.

[3] A. Barbalace, A. Luchetta, G. Manduchi, M. Moro, A. Soppelsa, and C. Taliercio, "Performance comparison of vxworks, linux, rtai, and xenomai in a hard real-time application," IEEE Transactions on Nuclear Science, vol. 55, no. 1, pp. 435–439, 2008.

[4] H. Leppinen, "Current use of linux in spacecraft flight software," IEEE Aerospace and Electronic Systems Magazine, vol. 32, no. 10, pp. 4–13, 2017.

[5] J. H. Brown and B. Martin, "How fast is fast enough? choosing between xenomai and linux for real-time applications," in proc. of the 12th Real-Time Linux Workshop (RTLWS'12), 2010, pp. 1–17.

[6] D. Bristot de Oliveira and R. Oliveira, "Timing analysis of the preempt rt linux kernel," Software: Practice and Experience, vol. 46, pp. n/a–n/a, 05 2015.

[7] D. Bristot de Oliveira, D. Casini, R. Oliveira, and T. Cucinotta, "Demystifying the real-time linux scheduling latency," 07 2020.

[8] C. Emde, "Long-term monitoring of apparent latency in preempt rt linux real-time systems," 2010.

[9] T. Cucinotta, "An efficient and scalable implementation of global edf in linux," 01 2011.

[10] D. Faggioli, F. Checconi, S. Superiore, S. Anna, M. Trimarchi, and C. Scordino, "An edf scheduling class for the linux kernel," 01 2009.

[11] P. Mckenney and D. Sarma, "Towards hard realtime response from the linux kernel on smp hardware," 01 2005.

[12] J. Alglave, L. Maranget, P. E. McKenney, A. Parri, and A. Stern, "Frightening small children and disconcerting grown-ups: Concurrency in the linux kernel," SIGPLAN Not., vol. 53, no. 2, p. 405–418, mar 2018. [Online]. Available: https://doi.org/10.1145/3296957.3177156

[13] J. Kim, P. Shin, S. Noh, D. Ham, and S. Hong, "Reducing memory interference latency of safety-critical applications via memory request throttling and linux cgroup," in 2018 31st IEEE International System-on-Chip Conference (SOCC), 2018, pp. 215–220.

[14] J. Kim, P. Shin, M. Kim, and S. Hong, "Memory-aware fair-share scheduling for improved performance isolation in the linux kernel," IEEE Access, vol. 8, pp. 98 874–98 886, 2020.

[15] L. Liu, Z. Cui, M. Xing, Y. Bao, M. Chen, and C. Wu, "A software memory partition approach for eliminating bank-level interference in multicore systems," in Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques, ser. PACT '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 367–376. [Online]. Available: https://doi.org/10.1145/2370816.2370869

[16] L. Abeni, A. Balsini, and T. Cucinotta, "Container-based real-time scheduling in the linux kernel," ACM SIGBED Review, vol. 16, no. 3, pp. 33–38, 2019.

[17] H. Grip, J. Lam, D. Bayard, D. Conway, G. Singh, R. Brockers, J. Delaune, L. Matthies, C. Malpica, T. Brown, A. Jain, A. Martin, and G. Merewether, "Flight control system for nasa's mars helicopter," 01 2019.

[18] B. B. Brandenburg and J. H. Anderson, "Joint opportunities for real-time linux and real-time systems research," 2009.

[19] L. Abeni, A. Goel, C. Krasic, J. Snow, and J. Walpole, "A measurement-based analysis of the real-time performance of linux," in Proceedings. Eighth IEEE Real-Time and Embedded Technology and Applications Symposium, 2002, pp. 133–142.

[20] (2021) The github repository of the xilinx linux kernel. [Online]. Available: https://github.com/Xilinx/linux-xlnx/tree/xlnx_rebase_v5.4

[21] (2021) Xilinx wiki on yocto. [Online]. Available: https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18841883/Yocto4

[22] (2021) Yocto zeus version repository. [Online]. Available: http://layers.openembedded.org/layerindex/branch/zeus/layers/

Session Th.4.A

Autonomy

Thursday 2nd June

14:00

–

Amphithéâtre

Efficient Use of Systems Theoretic Process Analysis for Automated Driving Systems

Rolf Johansson
Astus AB
Mölndal, Sweden
rolf@astus.se

***Abstract*— This paper describes how to use Systems Theoretic Process Analysis (STPA) for the purpose of being part of a safety case of an Automated Driving System (ADS). A central contribution is the proposed control structure following a decision hierarchy. This enables the generation of a list of efficient unsafe control actions (UCA) and corresponding controller constraints, for which it is possible to cover a complete list of loss scenarios. These results can master the general problem of reaching completeness with respect to all potential unsafe scenarios. In particular this solves some problems highlighted in ISO 21448 (SOTIF), like the so-called Area 3 problem and the problem of triggering conditions. The most important outcome of this paper is that it enables reaching completeness in the verification strategy without running into the problem of "billion miles of driving", which can be the case when the set of loss scenarios leading to UCAs is potentially infinite. Even the "smart miles" argumentation is avoided this way, as the definition of the scenarios related to the UCA of the respective controllers is not formulated such that an enormous number of test miles is required.**

***Keywords*— Autonomous driving, STPA, Safety, ADS**

I. Introduction

It is a very hard problem to provide a safety case for an Automated Driving System, ADS, for several reasons (extreme complexity of traffic behaviour, implicit task to solve, in principle infinity number of critical scenarios, etc.). The differences compared to many other applications are significant, which means that using experiences and best practices from other domains will still call for specific innovations to also cover the ADS cases. In this paper we focus on how to make System Theoretic Process Analysis, STPA, efficient when having an ADS of level 3 and 4 in scope.

The STPA methodology consists of four steps: "Define purpose of the Analysis", "Model the Control Structure", "Identify Unsafe Control Actions", and "Identify Loss Scenarios" [1]. One of the most important properties of STPA needed for a successful application is to identify a control structure efficient for the purpose. This shall depict a dynamic control structure being able to include all causes of failure. What "all" really implies here is one of the central challenges. One of the implications of "all" is that a safety case for an ADS shall have a predictive power, meaning that all root causes that could jeopardize safe traffic behaviour shall be covered. For an ADS this means on the one hand that the very complex environment needs to be captured in the control structure, such that the interaction with other traffic actors can be covered. On the other hand, it shall also capture the essence of what it takes to act autonomously in terms of a decision hierarchy and complex relations to a user.

In this paper, we show how to instantiate the STPA methodology for an ADS, where the connotation of an ADS is following the SAE definition [2]. A central contribution is a control structure built around the decision hierarchy of strategical, tactical, and operational decisions, which can be used to structure both the relations to the user and the relations to the environment, including all other traffic actors. This way, on the one hand we cover the problem denoted "triggering condition" in ISO 21448 [3] and on the other hand we also cover what is denoted the "area 3" problem of hazardous unknown scenarios. It is important to remember that the control structures to be used in STPA are abstract models [1] and should not directly depict any implemented controller functions. There is a recommended by [1], to use a perspective of a dynamic control problem rather than a failure prevention problem, or an insufficiency prevention problem. By applying this recommendation to our problem, we reach the advantage of STPA, showing how to reach a proactive analysis method.

The paper is organized as follows: Sections II-V go through the four steps of the STPA methodology and show how each of them can be adapted to ADS. Section VI describes how the proposed approach to use STPA for ADS relates to the state of the art. Section VII explains how the problems, specifically considered hard for ADS, are mastered by the proposed approach. Finally, section VIII contains a summary and conclusions.

II. Scope for STPA for ADS

The first step of the STPA methodology is to define the purpose of the analysis. In our case it is about a safety case for an ADS, which means that this step is very closely related to the definition of what "Safe" shall mean in the safety case of an ADS. There is so far no agreed global consensus on this, which is not so surprising, as what is considered as a loss depends on what is "unacceptable to the stakeholders" [1], p16. Different ADSs having different use cases being deployed on different markets, may consequently also meet different stakeholders with different understanding of what is (un)acceptable.

One way to formulate what is acceptable in a very distinct way, still leaving flexibility for different use cases, is the formulation of positive risk balance as criteria for what is acceptable: "diminution in harm compared with human driving, in other words a positive balance of risks" [4]. As further elaborated and explained in [5], the important conclusion from this report is that "it agrees with balancing risks against one another rather than with ruling out any calculation at all". Applying this to the STPA sub step of *Identify Losses* means that what the stakeholders today consider as an unsafe outcome in a given traffic environment, should be the base in the formulation of what are the losses to consider.

The outcome of the first STPA step is to produce: *System-level hazards*, *System-level constraints*, and *Sub-hazard constraints*. When applying this to an ADS, it is very important to identify *hazards* in an efficient way. From the STPA handbook, a *hazard* in the context of this methodology is to be understood as: "a system state or set of conditions that, together with a particular set of worst-case environmental conditions, will lead to a loss"[1], p 17, which is essentially the same as in the functional safety standard for road vehicles, ISO 26262: "potential source of harm caused by malfunctioning behaviour of the item"[6]. In both these definitions the intent is to identify something that can be the responsibility of the system itself to avoid. The system design can then get input requirements to be performed such that the hazards are avoided.

In most safety standards, the hazard analysis and risk assessment (HARA) is the prescribed method to identify what are applicable hazards and their related system-level constraints. In ISO 26262, the latter are denoted safety goals. When specifically having ADS in scope, it is possible to make an increase in responsibility of the system itself, compared to how an Item in general can be analysed according to ISO 26262. In a classical HARA, we leave outside of the responsibility of the system to restrict the "particular set of worst-case environmental conditions [that] will lead to a loss"[1], p17. In this paper, we propose to rather include this responsibility to the ADS itself, which means that the hazards as expected for the STPA will directly become the loss scenes (or "crash scene" according to terminology of [2]). This fits very well to the control structure following a decision hierarchy of responsibility, explained in the following section. In [7], there is a detailed description about how to identify system-level safety requirements for an ADS, i.e. generating the assumed outcomes for this first step of the STPA methodology. The quantitative approach presented there is also fully compliant with the recommendation of [4] and [5] as discussed above.

III. Control Structure

A. The Second STPA Step in General

The second step of the STPA methodology is to "Model the Control Structure". It is important to understand that this is not about representing the implementation of the system, but to identify a control structure efficient for the given purpose. More specifically, the handbook says: "A hierarchical control structure is a system model that is composed of feedback control loops. An effective control structure will enforce constraints on the behavior of the overall system"[1], p 22. To get the efficiency of STPA, it is critical to identify "multiple interacting control loops [which] can be modeled in a hierarchical control structure"[1], p23, such that the purpose is supported. In our case, this means that we shall define a control structure where the feed-back loops together can enforce the increased responsibility as suggested in the previous section.

B. The Decision Hiearchy of ADS

A basic conceptual control structure applicable for an ADS is the decision hierarchy used in [2], and inherited from [8]: strategical, tactical, and operational decisions. A higher-level decision controls a lower-level decision in the sense that it defines what the lower-level controller shall aim for. A major result from our STPA pattern is that any lower-level controller shall give continuous feedback concerning its capabilities to fulfil control actions from the higher-level controller. This is a major means to identify controller constraints such that unsafe control actions can be sufficiently avoided. By this way of identifying the constraints on the unsafe control actions (UCA), we can also reach completeness in describing the loss scenarios as further elaborated below.

A general challenge for STPA is to show completeness with respect to all loss scenarios of concern. This is also a major issue in ISO 21448, where all overlooked important scenarios are denoted area 3 [3]. We use the inherent advantage of STPA, avoiding putting the chain-of-failure (capability limits) events as the fundament for the analysis and instead use a conceptual controller structure on a logical level of abstraction. This way we can master the area 3 problem in general, as well as the problem of reaching completeness w.r.t. triggering conditions. A key is to identify what are the constraints on the different levels of control to be able to safely "control" both the ADS user, and all relevant traffic actors. Please note that in the STPA context, to "control" does not imply obedience [1], p26. It is the normal case for STPA that control structures often include components for which executable models do not exist, which obviously is the case when human actors (and animals etc) are part of the controlled process. Still, by separating the responsibilities among the controllers, and putting appropriate controller constraints, we can reach completeness w.r.t. situations of more or less infinite variants of human behaviour.

The general high-level control structure we have identified as being efficient for analysing an ADS, is as depicted in Figure 1. Each of these controllers can then be further functionally decomposed to a more fine-grained controller structure. An example of this is shown in Figure 2.

In this paper we list what is the task of each controller level, and what then becomes the corresponding unsafe control actions. We also explain how this control structure can be used for identifying a set of unsafe control actions, and how the this enables a fairly limited set of loss scenarios. A major outcome of this control structure, is that the corresponding set of loss scenarios can be expressed in a limited way, which is a very valuable result when solving the classical verification problem of an ADS: how to argue that the used set of scenarios is large enough. This is how the area 3 problem is avoided, without running into the trap of calling for a "billion miles of test

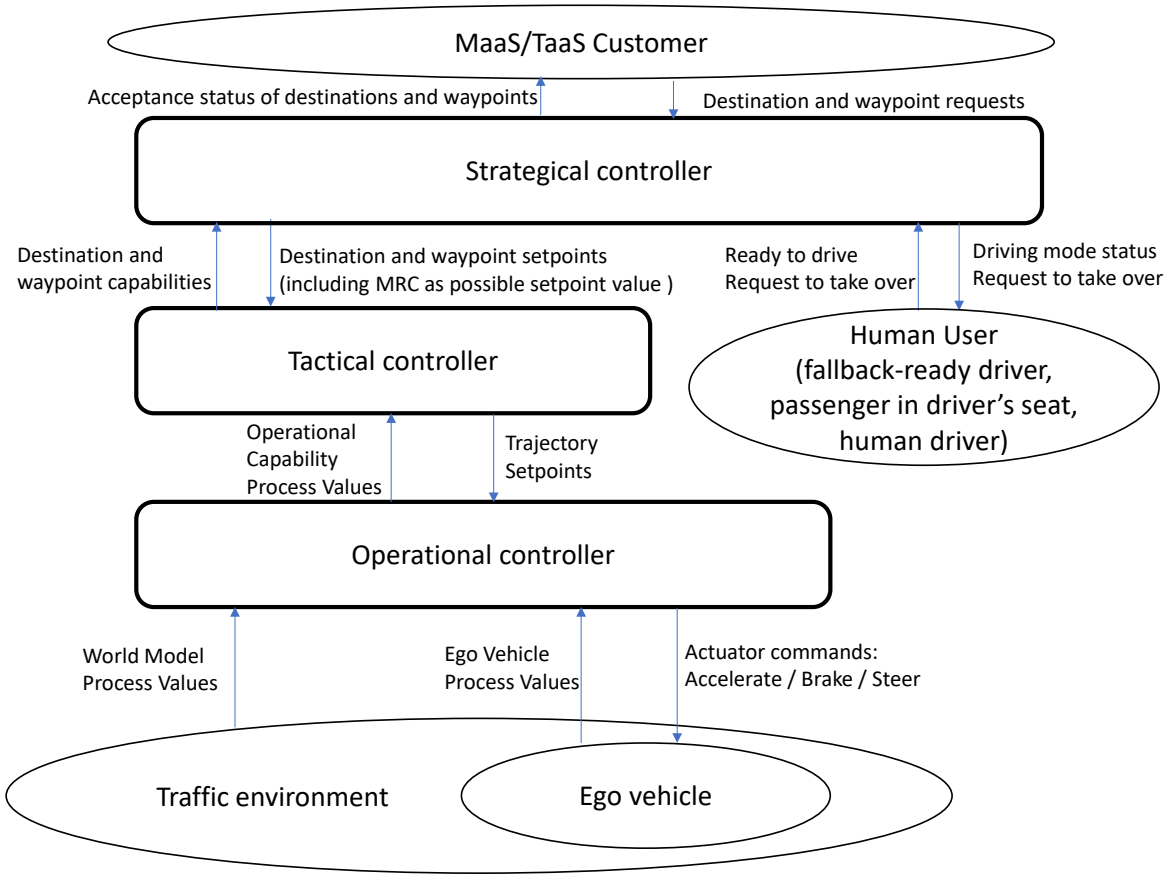


Figure 1 Control Structure of ADS

driving". A more exhaustive description and analysis how these properties are reached are found in section VII below.

C. Strategical Controller

The main task of the strategical decision-level controller is to define the goal of the trip. This includes a negotiation with the Mobility-as-a-Service (MaaS) / Transport-as-a-Service (TaaS) customer, but it also includes the alternatives of never to start, and of interrupting a trip changing its strategical decision to a minimal risk condition, MRC. This means that the MaaS/TaaS customer might come with preferred trip destinations, but it is the ADS strategical controller that makes the decision of what is a safe strategical control action on this level, i.e. formulates the safe control action for the tactical decision level controller to execute. It is important that the final strategical-level controller decision lay with the ADS (and not with the human customer), to be able to avoid any unsafe control action. It is fundamental to formulate constraints on the strategical decision level controller, not to accept any strategical decision that cannot be guaranteed to be able to reach safety. As this is a control-loop responding to feed-back, the strategical decision needs to be reassured constantly, which for example may lead to either suggesting the passenger in the driver's seat to take back the control (both level 3 and 4), request the fallback-ready user to take back the control (level 3), or to change the ADS strategical decision to an MRC (level 4). In section IV below, we show a full set of unsafe control actions for the strategical controller, and how these can be connected to a finite set of loss scenarios.

D. Tactical Controller

The main task of the tactical decision level is to decide how to reach the given strategical goal. This includes decisions like for example: what vehicle speed to aim for, what distances to keep to other actors and objects, what lanes to use, when to initiate an overtake, etc. A critical controller constraint is to guarantee that any future scenario will be able to be handled safely. This way of formulating the problem puts the focus on the power of the tactical decisions, as they are part of creating the scenarios. A major means to handle critical scenarios can be to avoid them in the first place. By formulating controller constraints restricting some possible critical scenarios, the ADS safety argumentation problem can be mastered. In section IV below, we show what this implies in terms of unsafe control actions and how to formulate such controller constraints. We also discuss how this enables the limited set of related loss scenarios. In summary, the unsafe control actions can be formulated as either providing trajectory setpoints outside the constraints as communicated by the operational controller, or by

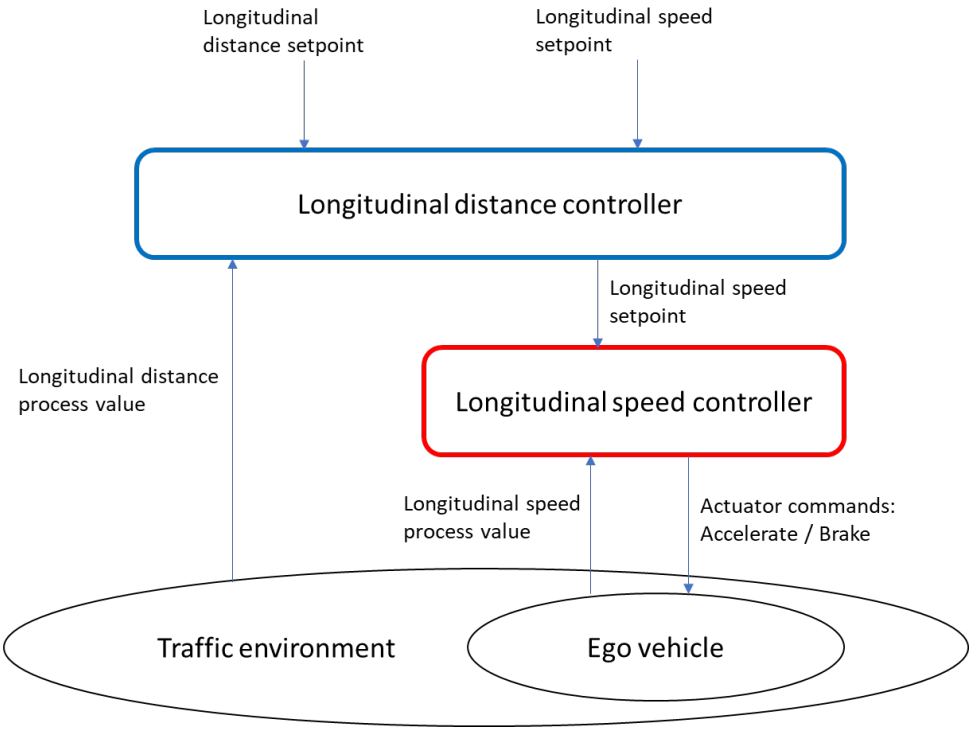


Figure 2 Some Controllers Inside Operational Level

defining trajectory setpoints not being shown as (recursively) safe. This latter property is the very most specific for autonomous driving, and this controller constraint is explained in more detail in section IV below.

E. Operational Controller

On the operational decision level, we find the traditional controller model, that for an ADS is to execute a given trajectory. The controller constraints to avoid unsafe control actions include what it takes to be sufficiently close to the given trajectory. Whether this trajectory is safe or not, is not a question for the operational level controller for which the only task is to execute a trajectory that is deemed safe by the controller on higher level. As for all the controllers, this is a continuous process where new decisions are given and new feed-back from the controlled process appears all the time. The important thing is that the controller constraints are identified such that the feed-back loops of the controllers can guarantee the ADS to stay safe. By limiting the task for the operational controller to what is complementary to the two other controllers, it is possible to limit the set of loss scenarios, and thus to formulate a feasible verification strategy avoiding the 'billion miles problem'.

Please note that this controller then can be further refined and depicted as the composition of a set of cascaded controllers, which are following the traditional pattern of cascaded controllers. It is important not to mix up the controller hierarchy emphasized in this paper with the identification of a traditional cascaded controller as further detailing the operational decision level controller. The operational decision level controller has the task of fulfilling both position and speed in both longitudinal and lateral directions. The longitudinal control part can be regarded as similar to an adaptive cruise control function, with the set points for distance and velocity, respectively, coming from the tactical decision level controller. Such an 'ACC controller' can then be depicted with a traditional cascaded controller hierarchy with the distance controller on top of the velocity controller as shown in Fig. 2.

IV. Unsafe Control Actions

A. The Third STPA Step in General

In STPA, the term Unsafe Control action (UCA) is defined as: "a control action that, in a particular context and worst-case environment, will lead to a hazard"[1], p35. By identifying a complete set of UCAs, safety can be achieved by assuring controller constraints guaranteeing the absence of all the UCAs. In other words, "A controller constraint specifies the controller behaviors that need to be satisfied to prevent UCAs"[1], p40. It is also important to remember that the UCA is only there to make sure that completeness can be reached when identifying a set of controller constraints. It is only about necessary conditions to cause a hazard, not to identify sufficient conditions (an UCA does not guarantee that a hazard will always result [1], p18).

Identifying the UCAs for the respective controllers gives a guide how to design a safe system. An efficient pattern for identifying UCAs for an ADS, is the one that both assures completeness and is feasible to implement.

In the previous, we say that we increase the responsibility of the ADS, compared to a general feature, to also include the task of making sure that the scenarios that may occur can also be handled safely. This means that we need to include this broader responsibility when determining what is a complete set of UCAs for the controller structure of the ADS. Doing so is not a problem, but rather what are the expected safety requirements on the controllers higher up in the decision hierarchy. A central task for the strategic and tactical controllers, respectively, is to make sure that only such scenarios may appear, which can be safely handled. More specifically, the UCAs and their transformations into controller constraints can be formulated for each of the controllers as detailed in the following sub sections.

Please note that in the STPA methodology, the control actions of concern include the possible outputs on all the interfaces of each controller. This is important to remember for a complex structure with interacting controllers. As specifically noted in the STPA handbook, when discussing the possible reasons for a UCA to occur: "Unsafe control inputs from other controllers can also cause UCAs. These can be found during the previous step when identifying Unsafe Control Actions for other controllers", [1], p 46. This implies that it is important to include UCAs of a controller in relation to all other possibly affected controllers, when identifying all possible interfaces and their control actions as candidates for UCA.

B. Strategical Controller

As mentioned in section III, the task of the strategical controller is to determine the (current) destination of the trip. As depicted in Fig. 1, there are two main possible interfaces where the strategical controller has the potential to perform an Unsafe Control Action (UCA):

- Setpoints to Tactical controller
- Status info and takeover request to Human User (fallback-ready driver, passenger in driver's seat, human driver)

A mistake from the strategical controller to act in any of these two interfaces may cause the ADS to become unsafe. Failure in the third interface may highly irritate and confuse the MaaS/TaaS Customer of the ADS, but there will be no safety consequences:

- Acceptance status to Maas/TaaS Customer

As a general guideline, the STPA handbook recommends using classical Hazop guidewords to derive the UCAs from the control actions expected on each interface, [1], p 36. However, the applicability of this advice is dependent on the "data type" of the control action to consider. For the controllers on the higher levels of the proposed ADS control structure, this Hazop technique is found less applicable, and formulating the UCAs of the strategical controller can be listed as follows (including a short discussion of each):

- UCA_S_1: Prescribing a destination (including MRC) or waypoint to the tactical controller, not possible to reach safely.

This is the main responsibility of the strategical controller in relation to the other controllers in this control structure. The responsibility for the tactical controller is conditional on the assumption that the strategical decisions are possible to reach safely.

- UCA_S_2: Providing driving mode information to the human user (fallback-ready driver, passenger in driver's seat, human driver), causing mode confusion.
- UCA_S_3: Providing request to take over in a tactical complicated situation, causing unfair transition.
- UCA_S_4: Failing to timely follow the protocol for request to take over, causing a stuck in transition.

All these three UCAs are in relation to the I/F to the human user (fallback-ready driver, passenger in driver's seat, human driver). It is outside the scope of this paper to analyse how this interface can cause the ADS to become unsafe. How all these three UCAs of *mode confusion*, *unfair transitions*, and *stuck in transition*, relates to safety of ADS, are further elaborated in [9].

C. Tactical Controller

The tactical controller has, according to the control structure depicted in Fig. 1, two interfaces for which UCAs are possible:

- Setpoints to Operational controller
- Capability information to Strategical controller

A mistake from the tactical controller to act in any of these two interfaces may cause the ADS to become unsafe. Formulating the UCAs of the tactical controller can be listed as follows (including a discussion of each):

- UCA_T_1: Prescribing a trajectory to the operational controller, not possible to follow safely.

This is the main responsibility of the tactical controller in relation to the operational controller. The responsibility for the operational controller is conditional on the assumption that the tactical decisions are possible to execute safely. Please note that the implication of this UCA when it comes to formulation of controller constraints guaranteeing absence of UCAs, should be done in a non-trivial way to serve as efficient guidance in the ADS design. Even if it is outside the scope of this paper to go into these details, in short it can be described as that one pattern for related controller constraints can be formulated as prescribing a trajectory that is all of: *reachable*, *updatable*, and *recursively safe*, as further elaborated below.

Reachable is directly requiring that the trajectory prescribed to the operational controller shall be inside what is declared possible to reach from the operational controller. *Updateable*, relates to the feedback process in the controller, requiring that continuous updating of the trajectory has to respect what are possible update changes in each step. Finally, *recursively safe*, requires that neither the current path, nor its updates, shall describe an unsafe scene. Directly avoiding prescribing a trajectory containing a loss scenario might be considered as

trivial. However, the key task for the tactical controller is to constantly update the trajectory as a consequence of the feed-back loop structure, and it hence needs to constrain the prescribed trajectory set-points such that it guarantees itself to perform all these updates safely. Hence, the above UCA_T_1 is to be covered by the triple of controller constraints guaranteeing the properties of *reachable*, *updatable*, and *recursively safe*, respectively.

- UCA_T_2: Providing capability information to the strategical controller, enabling unsafe destinations.

This is the main responsibility of the tactical controller in relation to the strategical controller. The responsibility of the tactical controller is conditional on the assumption that the strategical decisions are possible to reach safely. However, to enable the strategical controller to perform its part, it is critical for the tactical controller to tell the limits of what it can control itself.

D. Operational Controller

When coming down to the operational controller, this is a level where many descriptions of STPA has been done before. To a large extent, these still apply in this proposed ADS solution. The major difference is the responsibility for the operational controller in its relation to the tactical controller.

According to the control structure depicted in Fig. 1, there are two interfaces for which UCAs are possible:

- Actuator commands to Ego Vehicle
- Capability information to Tactical controller

A mistake from the operational controller to act in any of these two interfaces may cause the ADS to become unsafe. Formulating the UCAs of the operational controller can be listed as follows (including a short discussion of each):

- UCA_O_1: Prescribing an actuator command leading to an unsafe scene.

In a detailed STPA for an ADS, the pattern recommended in the STPA handbook of using Hazop guidewords for each detailed control action, is applicable on this operational level controller. The above UCA can hence be expanded to a large set of UCAs. However, for the scope of this paper it is sufficient to summarize these to one generic UCA as above.

- UCA_O_2: Providing capability information to the tactical controller, enabling unsafe trajectories.

This is the main responsibility of the operational controller in relation to the tactical controller. The responsibility for the operational controller is conditional on the assumption that the tactical decisions are possible to reach safely. However, to enable the tactical controller to perform its part, it is critical for the operational controller to tell the limits of what it can control itself.

V. Loss Scenarios

The main purpose with the fourth and final step of the STPA methodology is to identify why any UCA can occur, which includes why control actions may be improperly executed. It is about finding general answers to these questions applicable on this abstraction level, not trying to identify single implementation causing factors, which would rather become a traditional FMEA [1], p51. Once again, the main point is to reach completeness to be able to efficiently use this methodology. For each controller, the question is how to describe its relation to the controlled process such that it can be analysed under what conditions the UCAs may cause unsafe consequences. The expected outcome of this step is twofold:

- Identify scenarios that lead to an UCA
- Identify scenarios for a control action improperly executed or not executed

As for the previous step, the major contribution in this paper is related to the higher-level controllers, but also for the operational controller there is a significant value in the proposed solution.

The scenarios for a given controller is to be expressed in the context of that controller, which means that the controlled process is to be separately identified for each controller. For the operational controller, the controlled process is the traditional one covering the behaviour of ego vehicle and the entire traffic system as depicted in Fig. 1. For the tactical controller, the controlled process is the operational controller and for the strategical controller the controlled process is the tactical controller plus the behaviour of the human user (fallback-ready driver, passenger in driver's seat, human driver). The way these contexts are defined makes it possible to reason about the scenarios in a general way, which would be very, very hard otherwise.

For the operational controller, the responsibility is to execute the trajectories it has dynamically claimed itself that it can execute. This means that it has full freedom in its implementation to be conservative in its dynamic claim of possible scenarios to handle, then leaving to the tactical controller to ensure that the resulting scenarios are restricted accordingly. Instead of having the very complex problem of walking through all possible traffic scenarios, in this context the identification of scenarios is related to the possibility for the operational controller to claim (near-time predict) its own capabilities. It is still a hard problem, but it can be decoupled from the much more complex problem of analysing all possible complex scenarios of the traffic system.

For the tactical controller, the scenarios of the controlled process can be transformed to the problem covered by the control constraints as above of *reachable*, *updatable*, and *recursively safe* trajectories executed by the operational controller.

For the strategical controller and the controlled process being the human user (fallback-ready driver, passenger in driver's seat, human driver), the problem can be transformed to always guaranteeing absence of *mode confusion*, *unfair transitions*, and *stuck in transition*, as described above and further elaborated in [9]. For the controlled process being the tactical controller, the problem of identification of all affected scenarios can be transformed to the problem covered by the control constraints as above of guaranteeing a destination

(including MRC) or waypoint that is possible to safely reach by the tactical controller.

Another way to express the consequence of the control structure and their corresponding controller constraints is to say that they become non-sensitive to a detailed description of scenarios in the total traffic environment. We can directly use the set of controller constraints identified in the third step, without further needing to put each of them in a certain scenario context. We do not need to constrain the UCAs to be considered under only certain scenarios, but can claim them to be valid in all scenarios. For a conventional system this is not considered as efficient enough for the purpose of this fourth STPA step. But for an ADS with the suggested increased responsibility, and when applying the proposed control structure and corresponding controller constraints, it is still efficient. For the upper-level controllers, this follows from the limited ways of expressing scenarios in the local context of the respective controller. For the operational controller, this follows from the limited responsibility for the operational controller, as a consequence of the responsibilities put on the high-level controllers. As the operational controller can assume that it will only run into situations it can handle safely, the problem of linking UCAs to possible scenarios can be mastered.

VI. Relation to State-of-the-ART

The way to apply STPA for ADS presented in this paper goes beyond the state of the art by addressing how a carefully chosen control structure together with corresponding controller constraints, can master all the problems mentioned above in the introduction. This proposed STPA control structure focuses on the hierarchical decision structure that is specific of ADS, as depicted in [2], which in turn is building on previous characterizations of the driving task [8].

It is clearly stated in the STPA handbook [1] that the intention of identifying a control structure for STPA is not to depict any real implemented controller, but to conceptually cover the essence of the functional model.

For levels of automation lower than an ADS, i.e. level 0-2, the focus for safety analysis lies inside the operational level of decisions. When applying STPA for such systems this means that the focus is on identifying a control structure within the operational level. An example of an STPA of such ADAS function can be found in [10]. This is still the state of the art for lower levels of automation.

To apply STPA on ADS has been done by [11] which propose to decompose the problem in three levels of abstraction of the architecture. The reason for introducing these abstraction levels is to comply with the ISO 26262 prescribed abstraction levels. However, the decision hierarchy aspect is not covered in this approach and consequently all the suggested architecture levels are applied to the analysis of the operational decision problem, meaning that the task of generating a feasible set of loss scenarios is not addressed.

The aim for the safety analysis in [11] can be seen as rather to show compliance to the at that time defined safety standards than to argue for predictable power of an analysis covering all causes. This is the same as in other attempts, e.g. [12] to describe how STPA fits into safety analysis of ADS. What has happened after these publications is that there is a new proposed application standard for safety of ADS (ISO TS 5083), which is to cover all causes of becoming unsafe, including security. This standard aims to explain how evidence of a complete safety case of an ADS is to be collected from both particular existing standards like for example ISO 26262, or ISO 21448, and by other means. The proposed control structure of this paper can be seen as to address safety of ADS covering all root causes at the time, without needing to separate the problem as a set of different compliances. This way, it goes beyond the earlier publications on STPA for ADS and it is aiming for fitting compliance of the ISO TS 5083, yet to be defined.

Even if the automated driving of a road vehicle differs from an autonomous ship, the essence of modelling autonomous vehicle control is the same. A recent publication elaborating how to use STPA for autonomous ships is [13]. There is a very ambitious control structure of more than 20 controllers. However, there is a clear indication of three main controllers in a decision hierarchy: Shore-based control centre, Autonomous ship controller, and Autonomous Navigation systems. These could intuitively be identified as corresponding to the decision hierarchy as used in [2] and applied in this paper. However, there is no strict analysis of what constitute the unsafe control actions from the perspective of complementary responsibilities of the controller levels. Instead of reaching the completeness by definition, using the role separation between the different decision levels, they have used experienced personnel to describe what they consider as important for a certain level. In this paper we go one step beyond [13] by proposing a division of responsibility between the controllers such that an identification of unsafe control action can be made complete by definition.

A control structure somewhat similar to the one proposed in this paper can be found in [14]. Instead of identifying the decision hierarchy as in [2], they denote their two upper levels of controllers: Global path planning, and Local path planning, respectively. Below that, there is not just one controller but three: Kinematics models, Obstacle detection classifier, and Localisation, respectively. To a certain extent, they also identify the need for lower-level controllers to inform the higher-level controllers about their capability limitations. However, this is far from completely performed and there is no argumentation that by consequently applying this pattern, the set of unsafe control actions (UCAs) can be deemed complete and also that the set of loss scenarios this way can be identified as a limited set. A major reason for that paper not going so far could be their proposed control structure which does not represent the responsibilities of the decision hierarchy. The middle controller of Local path planning looks as a mixture of tactical decisions and operational decisions, which is not surprising as this is what a standard trajectory planner does. This is a reason to once again emphasis the importance of proper abstraction, efficient for their purpose, when identifying the control structure to use in STPA.

Instead of solving the problem of completeness by applying the decision hierarchy, the authors of [14] address the problem of a non-finite number of loss scenarios by introducing “smart number of miles”. Instead of identifying the role of predictors and of prescribing the sensing capabilities in a complete way,

they address all these in one large problem of identifying a large enough set of loss scenarios. In contrast to [14], in this paper, the large problem is decomposed in the control structure and the responsibilities between the controllers is strictly formalized such that the set of UCAs can be shown complete and the set of loss scenarios can be limited.

VII. Discussion

A basic problem in safety argumentation for an ADS is how to address completeness in different ways. It is about completeness w.r.t. both the aspects of:

- root causes
- traffic scenarios to consider

Completeness here is not to be interpreted in an absolute manner, but rather complete enough for the purpose. As safety is about showing a low enough remaining risk, completeness here implies avoiding that the claimed remaining risk is significantly underestimated because of something not considered. This means that when generating a safety case, the claims shall have a predictive power with a precision and granularity high enough for the order of magnitude for the claimed remaining risk.

None of the existing safety standards gives guidance how to master these two aspects of completeness. One of the reasons for introducing ISO 21448, was that it identified ISO 26262 as lacking completeness w.r.t. to root causes.

The problem of reaching completeness w.r.t. root causes is a complex issue, especially if all the root causes are assumed to be separately handled when performing the identification of the system-level safety requirements. None of ISO 26262 or ISO 21448 are claiming to cover the full root-agnostic scope. The two standards are seen as complementary w.r.t. root causes, but still there are several root causes not considered in any of them, as for example security.

In both these two safety standards for road vehicles, the problem for their users to reach completeness w.r.t. traffic scenarios is well articulated, but none of them gives a clear guidance how to master it (only requiring the user to solve the problem).

In ISO 26262 there is a requirement to show that the completeness w.r.t. all possible traffic scenarios is achieved in the HARA, and the hidden assumption is that this can be done because the limited dependency on the complexity of the traffic scenarios for the feature (*Item*) in scope. However, in principle ISO 26262 could be applicable for very complex features, which would call for a methodology addressing how to reach completeness w.r.t. complex traffic scenarios.

In ISO 21448 the completeness problem w.r.t. traffic scenarios is addressed and denoted *Area 3*. The task for the user of that standard is to argue that the set of hazardous unknown scenarios (*Area 3*) is sufficiently enough transformed to hazardous known scenarios (*Area 2*). Which is the traditional task for any hazard analysis. ISO 21448 has a bit more complicated way to analyse to what extent the frequency of different traffic scenarios has an impact to the resulting risk. In the ISO 26262 framework there is just one assumed frequency measurement of scenarios, and that is the estimation of the exposure frequency to be used in the HARA. In ISO 21448 there are two separate scenario frequencies: the occurrence of triggering conditions and the exposure of scenarios where a hazard cand lead to harm. The formal explanation of the triggering conditions is: “specific conditions of a scenario that serve as an initiator for a subsequent system reaction contributing to either a hazardous behaviour or an inability to prevent or detect and mitigate a reasonably foreseeable indirect *misuse*”. The total effect of how to address the completeness problem w.r.t. traffic scenarios in the ISO 21448, is that it becomes a bit complicated if the aim of the user is to further refine the problem and collect evidence to safety case inside the system design.

Similarly to ISO 26262, in ISO 21448 there is no general guidance how to master this completeness problem w.r.t. traffic scenarios. It is easy to get the impression that the only way to show that *area 3* is sufficiently small, is by extensive exposure of the complete system in its operational environment (showing fulfilment of the *validation target*), even if this is not explicitly formulated that way.

Anyhow, the dominating problem for efficiently generating a safety for generating a safety case for an ADS is how to avoid this “billion miles of driving” as a necessary piece of evidence. This is especially important in an industrial context of continuous deployment, where the ADS is to be updated at a high pace assuming a similar pace for generating safety cases. A necessary condition is then to master the completeness issues already in the higher-level analysis. This paper gives a solution how this can be done by means of the adapting the STPA methodology to ADS.

A first observation is that the STPA methodology very well suits to cover the problem of completes w.r.t. all root causes. It is rather the case that STPA requires to be root-cause agnostic as discussed above in section V.

Regarding the completeness w.r.t. all traffic scenarios, what is shown in this paper is how to use all the steps of the STPA methodology. This means firstly to challenge the responsibility and the system-level constraints such that the ADS gets an increased responsibility compared to traditional automotive features. It gets the role to itself address in runtime the problem of restricting itself to situations it can safely handle. Then secondly the STPA is modelled on a conceptual abstraction by means of a controller hierarchy implementing a decision hierarchy. By separating the roles of these three controllers, the complete responsibility of the ADS is divided such that what are the possible unsafe control actions, UCAs, for each of these can be expressed with significantly less dependencies on possible individual traffic scenarios. On the one hand the control structure and its related controller constraints completely solves the responsibility of the ADS. On the other hand, the UCAs and the corresponding controller constraints can be analysed individually in their local context, which means that they can be decoupled from all being dependent on a full description of all possible traffic scenarios, to argue completeness.

The resulting controller constraints then can serve as a backbone in the argumentation structure of the ADS safety case.

Because their limited dependency to describing all possible traffic scenarios, the full ADS safety case can rid of the “billion miles of driving” as a necessary piece of evidence.

VIII. Summary and Conclusions

We describe a general control structure as depicted in Figure 1 which enables us to generate a list of Unsafe Control Actions (UCA) listed in Section IV. For each controller, the listed UCAs and corresponding controller constraints enables the generation of a complete list of scenarios possibly leading to these UCAs. This way, we master the problem of Area 3 as well as of the problem of triggering conditions, as formulated in ISO 21448 [3]. The most important outcome of this analysis pattern is that it enables reaching completeness in the verification strategy without running into the problems of “billion miles of driving”, as can be the case when the set of scenarios leading to a UCA is potentially infinite. Even the “smart miles” argumentation is avoided this way as the definition of the scenarios related to the UCA of the respective controllers is not formulated such that a large number of miles is required, smart or not.

The decomposition of controllers according to the decision hierarchy, as is depicted in SAE J3016 [2], directly leads to a set of UCAs for each of the controllers, decomposing the otherwise extremely large situations space for UCA to a limited set of situations for each of the controllers. As a pattern for defining controller constraints guaranteeing the absence of the UCAs, there is the ability for a lower-level controller to express its current capabilities to the higher-level controller. This enables the decomposition of responsibilities between the controllers. By allocating a significant part of the responsibilities on the higher-level controllers, the remaining responsibility of the lowest level, operational controller, can be analysed without needing an extremely large set of scenarios when challenging how to reach any of the UCAs.

References

[1] N.G. Leveson, J.P. Thomas, 'STPA Handbook, MIT, March 2018, http://psas.scripts.mit.edu/home/get_file.php?name=STPA_handbook.pdf

[2] ‘SAE J3016:APR2021, Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles’, April 2021

[3] ‘ISO/PAS 21448:2019 - Road vehicles— Safety of the Intended Functionality’, January 2019

[4] Ethics Comission, 'Automated and connected driving', German Federal Ministry of Transport and Digital Infrastructure, Report, June 2017

[5] C. Luetge, 'The German Etics Code for Automated and Connected Driving', in Philosophy & Technology · September 2017

[6] 'ISO 26262:2018 - Road vehicles - Functional safety’, December 2018

[7] F. Warg et al, 'The Quantitative Risk Norm - A Proposed Tailoring of HARA for ADS', in 50th Annual IEEE/IFIP Int. Conference on Dependable Systems and Networks Workshops (DSN-W). IEEE, 2020, pp. 86–93.

[8] J. A. Michon, 'A Critical View of Driver Behavior Models: What Do We Know, What Should We Do?' In Evans, L. and Schwing, R.C. (Eds.). Human behavior and traffic safety (pp. 485-520). New York: Plenum Press, 1985.

[9] R. Johansson, et al, 'Safe Transitions Between a Driver and an Automated Driving System', International Journal on Advances in Systems and Measurements, vol 10 no 3 & 4, 2017.

[10] S. M. Sulaman, et al, 'Hazard Analysis of CollisionAvoidance System using STPA’, in Proceedings Information Systems for Crisis Response And Management (ISCRAM) , 2014

[11] A. Abdulkhaleqa, et al., ‘A Systematic Approach Based on STPA for Developing a Dependable Architecture for Fully Automated Driving Vehicles’, 4th European STAMP Workshop, 2016.

[12] K. Czarnecki, ‘On-Road Safety of Automated Driving System (ADS) Taxonomy and Safety Analysis Methods’, Technical Report · July 2018, Waterloo Intelligent Systems Engineering (WISE) Lab University of Waterloo Canada.

[13] M. Chaal et al., ‘A framework to model the STPA hierarchical control structure of an autonomous ship’, in Safety Science, Volume 132, December 2020.

[14] S. Khastgir et al. 'Systems Approach to Creating Test Scenarios for Automated Driving Systems', in Reliability Engineering & System Safety, volume 215, November 2021.

Software fault propagation patterns for model-based safety assessment in autonomous cars

Yandika SIRGABSOU[1*], Claude. BARON[1] , Laurent PAHUN[2], Phillipe ESTEBAN[1]

[1]LAAS-CNRS, Toulouse, France
[2]Renault Software Factory, Toulouse, France

*Yandika.sirgabsou@laas.fr

Keywords:
MBSA, MBSE, Safety analysis, Embedded software, Fault propagation pattern

Abstract:
The development of driver assistance and autonomous driving systems for vehicles has started to revolutionize the transportation sector, promising comfort, and safety. While significant technological progress has already been made in this area, many challenges remain. Among these challenges, ensuring safety has become even more critical due to the increasing use of complex, communicating, and reconfigurable embedded software. Current solutions to address safety include the use of model-based approaches for safety analyses instead of the traditional document-based safety analysis that is both informal and inefficient when faced with complexity. To this end, and in the context of automotive embedded software, we propose to rely on the use of fault patterns to improve the construction of software models used to conduct safety analyses. This paper makes a methodological proposal that improves current practices in terms of facilitated model construction and reusability, and that has been validated on the study of an automotive software component.

I. Introduction

The rapid development of embedded systems has led to numerous innovations in various systems in our modern society such as autonomous vehicles and highly computerized systems in airplanes. The technological challenges related with complexity and societal needs for guaranteeing safety induced by this trend opened new avenues for research in systems engineering but also exacerbated existing problems as they relate to the use of critical software and its contribution to systems safety.

To cope with these issues, industrials developing safety critical systems are looking for new methods and tools for designing and sharing design ideas more efficiently while ensuring system safety as required by standards and regulations. In the past decades, most of systems and software engineering development processes relied on document-based methods that relied on informal design documents to convey design ideas and artefacts from one development stage to the other. The informal aspect of these practices (such as manual analysis based on informal documents that are subject to the interpretation of the safety analyst) makes them prone to errors and less efficient in regard of the complexity of today's systems architectures. Although they are still widely used, these methods are now being challenged and model-based are more and more favored. In this context, embedded systems manufacturers are turning towards model driven engineering as part of both their systems and software development as well as systems safety assessment. In Systems Engineering (SE), this had led to the adoption of MBSE (Model-Based Systems Engineering), a systems engineering practice aimed at describing both a problem (need) and its solution through models, concepts and languages [1]. Its adoption can now be considered a success story as we witness that more industrials developing safety critical systems are turning towards the MBSE approach in Systems and Software Engineering. Examples include Dassault with its integrated 3DS MBSE solution or SIEMENS that integrates MBSE within its Product Lifecycle management (PLM) solution. In Systems safety, a similar trend has led to the development of Model based Safety Assessment (MBSA), a practice that enables the capture, through specific formalisms and languages, of a systems safety related model (that describes the failure behavior and unifies all the safety property of a system in a single model), on the basis of which different safety analyses can be made.

However, despite the discipline being an early pioneer in the use of models, the wide adoption of model-based approaches for safety assessment has remained embryonic. In automotive, the ISO 26262 standard [2], titled "Road vehicles – Functional safety", requires performing safety analyses not only at system level but also at software level,

to ensure the safe behavior of the embedded software. Moreover, in the context of autonomous driving, embedded software assumes various critical safety functionalities. Unfortunately, today, the currents practices in safety analyses do not focus enough on embedded software even though the software implements the logic of some of the critical safety mechanisms. As a result, in the software context, safety analyses are either not performed or if performed, only done through traditional document-oriented approaches. Therefore, there is a need for more focused and rigorous methods for safety analyses at software level.

This paper aims to propose a methodology, specially aimed at improving the practice of automotive embedded software safety analysis thought the use of fault patterns within the MBSA approach. In the next section, a state of the art of current MBSA approaches is presented. In section 3, a methodological proposal based on the use of fault pattern is made and applied to a case study in section 4. The results are discussed in section 5 and a conclusion in made in section 6.

II. State of the art

In systems safety, safety assessment has been dominated by document-centric methods and processes since the 60s. Classical methods such as Fault Tree Analysis (FTA) and Failure Modes and Effects Analysis (FMEA) have been complementarily used by experts in the industry for systems safety analyses. In these classical approaches, safety analyses are manually performed based on paper-style artefacts such as systems drawings or spreadsheets that are found in design documents. While these practices use a sort of well-defined semantics (such as Boolean operators and FTA symbols), their representation is often very far from the systems they describe.

With the introduction of Model-Based Safety Analysis (MBSA) starting around early 90s, through the earliest model-oriented safety analysis techniques such as FPTN [3], Figaro [4] or the AltaRica [5] language, the focus has shifted to model-based approaches that no more base safety analyses on paper-style documents, but on a formal model of the system under design. This move has led, nowadays to an academic trend that seeks to address the interdisciplinarity and the consistency of the use of broad models through MBSE and MBSA throughout Systems Engineering and Systems Safety. In this trend one possibility is to directly associate the MBSA safety models with the MBSE system models. This allows conducting safety analysis directly on an extended version of the MBSE model. Examples of methods that are based on this extended approach include the approach included in the xSAP/Nu-SVM [6] [7] safety analysis tool. Another example is the Hierarchically Performed Hazard Origin Studies [8] based on extended SIMULINK models [9]. Although a clear advantage of the extended approach resides in the consistency (between system design and safety) it enables through the use of a shared system and safety model, it is argued that basing safety analysis on an extended model can lead to false assumptions and thus leading to hidden safety flaws (despite this claim not being shared by all systems engineering and safety practitioners). Furthermore, the safety analysis resulting from such model could be difficult to exploit because of the complexity of the extended model. In practice, safety assessment models are primarily used for mathematical or probabilistic calculations such as minimal cuts or monté carlo simulations. The more complex the model, the more computing power is required to effectively perform these calculations. Furthermore, the result of the safety analysis can be difficult to exploit due to the source models being blurred and overloaded (making them lose their ability to support a seamless communication) while the generated formal models are incomplete and uselessly complex [10].

As an alternative to the described first approach, a more appealing trend in academic research is to build a separate MBSA model that needs to be kept consistent with the MBSE model through additional measures such as model synchronization [11]. This last approach is often based on dedicated safety modeling languages such as AltaRica [12], Figaro [4] or SAML [13]. These safety focused languages allow unambiguous representation of systems for the needs of safety analysis using well defined syntax and semantics. However, early experiments feedback suggests that, in the case of the dedicated model approach, the MBSA model construction can be challenging for safety analysts especially for complex systems. In such case, it is imperative to find the right level of details in modeling for fear of having a model too complex that may well be at the limit useless or irrelevant. Furthermore, some analysts don't necessarily see the advantage and gain in time of modeling a separate dysfunctional architecture for safety analyses. However, as argued by Rauzy and Haskins in [10], systems description and safety analysis models are different by nature and efforts to unify such models in one single super model remains unrealistic. Based on this argument, the right direction is to keep a consistent separate MBSA model but make its construction easier. Nevertheless, the separate model construction can be challenging for large systems (what details, what modeling strategy etc.).

To ease the construction of dysfunctional models to conduct safety assessment, efforts have been done mostly using generic libraries of system elements that exhibit some safety properties. An example is the Safety Architecture Pattern(SAP) approach proposed by Kheren in [14]. In this approach, a library of SAP (components that highlights useful system's attributes from a safety point of view) are developed and coded using the AltaRica language. The

generic library is then reused to easily prototype safety-oriented systems architectures that can be reused to perform safety analysis using tools such as the OCAS Workshop[15]. Nevertheless, the proposed approaches are mostly aeronautic systems oriented. The developed libraries are often dedicated to avionics systems (such as pumps, electrical motors, valves, or control units). While these libraries can be used for modeling physical systems at system level in automobile, they are less suitable for modeling dedicated safety architectures of the embedded software. Moreover, although there are ongoing works that aim to apply the MBSA approach to automobile at system level, less focus is being put on embedded software. However, in the automotive context, ISO 26262 recommends conducting safety analyses not only at system level, but also at software level[2]. In this context, if the model-based approach is to be used for safety analysis at software architecture level as the commended by ISO26262, safety analyses can be made easier if patterns or libraries of safety related components (such as safety mechanisms used in software) can be developed drawing from the same principles as those described in the case of systems SAP-oriented approach.

III. Methodological proposal

The goal of the methodological proposal is to construct a fault library of reusable software safety mechanisms that are commonly found in safety related software components, drawing form the SAP library-oriented approach described in the state of the art and given the need for improving safety analysis practices at software level in the automotive context. Our choice has been to use the dedicated model approach coupled with dedicated languages such as AltaRica as described in [16]. However, as stated earlier, building a dysfunctional model can be challenging especially for complex systems. To address this concern, our solution has been to focus on selecting and including in the dysfunctional model only components that are safety-related as described in our previous work [17]. Even then, the failure propagation logic of these components must be manually written by the modeler (which can be time consuming for large systems).To address this concern, our first hypothesis is that making the MBSA model construction easier can both benefit its adoption by companies and improve the quality of safety analysis. A second hypothesis is that limiting the MBSA model to safety related components is sufficient to carry out meaningful safety analysis and can improve both efficiency and the quality of safety analysis. Therefore, the position of this paper is to make less painful for safety analysist the construction of MBSA models by proposing a set of predefined reusable libraries of software fault models to ease the dysfunctional model building process and improve the quality and consistency of the analyses. Its scope is limited to the context of automotive safety analysis at software level consistently with ISO 26262. Our methodology proceeds in 3 basic steps. The first one is to identify the safety mechanisms and their related software failure modes. The second step is to write the failure propagation logic through the safety mechanisms based on their functions and the identified failure modes; and to store the components in a library. The last step consists in reusing the elements stored in the library to build a dysfunctional model and conduct safety analyses.

In the first step we proceed by identifying the software failure modes and associated safety mechanisms found in automotive software architectures. In automobile, these failure modes and associated safety mechanisms are well defined by IO 26262 [13, Annex E, Annex D]. They are further detailed by two annexes in AUTOSAR [18] [19]. These failure modes are clustered into 4 categories. They include “data integrity, initialization & configuration data” “data exchange”, “timing & control flow” and “data processing”. The “data Integrity” category summarizes all failure modes related to corrupted memory or initialization and configuration data related to the corruption of software data at one memory address such as the corruption of memory content, memory partitioning fault or memory access fault. The “Data Exchange” category covers failures related to data transmission between sender and receiver such as between different ECUs (Electronic Control Units) or software components. The “Timing and Control Flow” category covers failure modes related to the timing of execution and scheduling. To model these fault categories, we start by describing the software components’ behavior through a generic abstract template using states and transitions. An example of such a template consisting of a software component with generic failure modes is provided in in Figure 1. It shows 4 states (Inactive, Nominal, Erroneous and Failed), representing the execution status of the software component linked by several possible transitions (Minor fault, Major fault, Recovery etc.). The inactive state is the idle or initial state preceding the initialization where the software component is not solicited or does not provide its function. From this state, the 3 outgoing transitions labeled “Activation”, “Major fault” and “Minor fault” will lead the component to the “Nominal”, “Failed”, or “Erroneous” states respectively. In the nominal state, the software component executes and delivers its intended function. From this state, the component can revert to the Inactive state as indicated by the transition label “Deactivation” or move to the “Failed” or “Erroneous” states if a major or minor fault occurs as indicated by the transitions. In the erroneous state, the software component provides erroneous results while, in the “Failed” state, it fails to provide its intended function. From the generic pattern, more specific patterns related to specific fault categories as described by ISO 26262 can be easily derived.

As an illustrative example, let us consider a piece of software that reads some critical data from memory, performs some critical calculations, and stores the result back to memory. Based on its function, the failure modes of this software component could include memory access related faults (such as read and write errors) as well control flow and timing related failure modes (such as execution failure, or untimely execution). Depending on the exact safety requirement, safety mechanisms to prevent the failure of this software component's function could include a protection against unwanted writing and a watchdog timer. Based on this information, the elements that will be necessary to model the safety related behavior of this software component, are the failure mode related to the memory access and the two safety mechanisms that will be modeled though states and transitions within this software component.

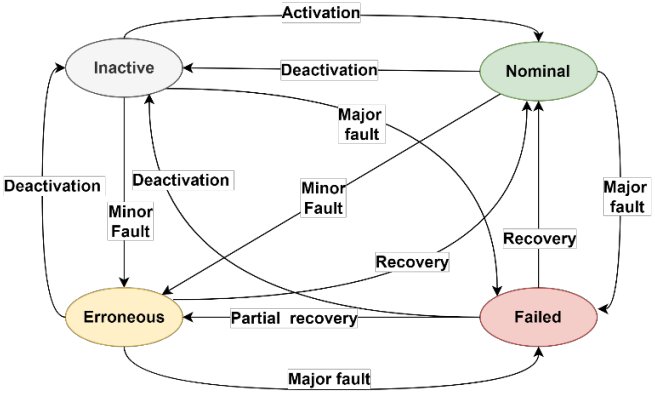


Figure 1. Failure modes of a generic software component

The second step focuses on writing failure propagation logic of the identified safety mechanisms knowing their function and considering the associated failure modes identified in the first step. To do so, we first need to study the safety mechanism and identify their basic behavior, what their inputs are, and the result they produce in normal or faulty execution. Once done, we can write the failure propagation logic of the identified safety mechanisms. To this end we use Failure Truth Tables (FTT) that we introduced in our early work [16]. FTTs are dysfunctional failure propagation tables consisting of discrete input and output variables whose possible values are defined depending of the failure behavior of the components function. FTTs can be used to capture the dysfunctional logic of a function based on its inputs. Depending on the expression of the safety requirements that the identified mechanisms are to fulfill, logical operators such as "or, and" and program control structures such as "if-then-else" can be useful to express the function or the combination of several mechanisms within a software component. Therefore, we also need to write and add to the library, the failure propagation logic through theses operators and control structures. This will result in a library of safety mechanism, operators and control structures that will be used in the last step to construct the dysfunctional model of a system.

The last step consists in using the elements stored in the library to construct the dysfunctional model. This step requires having a tool that offers library support. Using the elements stored in the library in an appropriate MBSA tool, one can easily model the dysfunctional architecture of a systems through drag and drop. This is possible since the safety mechanisms self-contain their propagation logics as well as the associated fault modes. Therefore, there is no need to write the failure propagation logic code in this step.

IV. Implementation in SimfiaNeo and case study

The objective of the case study was to first develop a library of safety mechanisms and used it to model the dysfunctional architectures. To this end, we are using SimfiaNeo [20] , an MBSA modeling tool based on the AltaRica language and developed by APSYS-Airbus. It offers a graphical modeling interface based on Eclipse and implements the dataflow version of the AltaRica language. SimfiaNeo allows to graphically build the AltaRica model of a system and to directly perform various safety analyses based on minimal cuts or in the form fault trees and FMEAs directly from the AltaRica model. Furthermore, the tool in its latest version offers library features that make it suitable for our proposal. It allows the modeling, storing and instantiation of custom components in a library.

In the context of Advanced Driver Assistance Systems (ADAS) and Autonomous Driving (AD), we aim to apply the proposed methodology on a practical case study, the longitudinal control software component. The longitudinal control is a function of the ADAS technology. It is a software component whose purpose is to ensure speed and braking control in autonomous driving mode. It is built around the ACC (Adaptive Cruise Control), a speed and distance control system that calculates how fast the vehicle can travel while remaining in a safe situation

with respect to certain predefined events (turns, traffic jams, stop signs, etc.). To ensure its safe activation, the longitudinal control relies on an internal monitor (a subcomponent called supervisor that manages its states) and a failsafe controller (a subcomponent that places the longitudinal control in some predefined safe states when some faults occur). Safety requirements associated with the longitudinal control are documented in a safety concept through safety goals that are declined to Unwanted Software Events (UWSE) at the software level. In this case study, we focus on one single UWSE related to the occurrence of an unintentional acceleration above the permissible acceleration limit (defined by ISO 22179) during travel ($v \neq 0$ km/h) entailing longitudinal control. Other constraints also exist (e.g., the user must be able to deactivate the ACC at all times).

1. Pattern prototyping and dysfunctional model construction with SimfiaNeo

Using our described approach, we modeled the basic structures of software components using the previously described fault categories and safety mechanisms. First, we declared the necessary AltaRica domains in SimfiaNeo. To this end, we declare 5 domains with different literals encompassing data integrity, Data exchange, Safety mechanism, generic state, and generic data. Using these 5 domains we modeled bricks of components representing the elements of a software safety architecture including elements such as generic software components, generic safety mechanisms, Boolean operators, and program control structures.

Description the fault patterns

In automobile, software fault modes and associated safety mechanisms are clustered into 4 categories. Including include “data integrity, initialization & configuration data” “data exchange”, “timing & control flow” and “data processing”. In this subsection we present a fault pattern related to the data exchange as an illustrative example and a generic behavioral pattern for safety mechanisms.

As described by ISO 26262 in the software scope, the “Data Exchange” category covers failure modes related data transmission between sender and receiver such as between different ECUs or software components. The pattern presented on Figure 2 shows how the execution of a receiving software component subjected to this category can be affected. Like the pattern shown in Figure 1, it has 4 states: “Init,” “Nominal”, “Erroneous”, and “Failed”. From the initial state (blue state on Figure 2), the function will either move to the “Nominal” or “Erroneous” states as indicated by the outgoing transitions depending on a successful or unsuccessful data transmission initialization. In the nominal sate, the software component executes normally and fulfils its function. If a data transmission initialization fault has led the function to the “Erroneous” state, an execution of a safety mechanism can bring the function to nominal if successful or to the “Failed” state (red state on Figure 2) if unsuccessful. The function can also move to from the nominal state to the “Erroneous” state with the occurrence of inconsistencies of the transmitted data (such as corruption, incorrect data value, out of range data values or incorrect sequence of data). In the “Failed” state, the software component fails to provide the expected function due to missing or loss of transmitted data or due to the safety mechanism failure to recover from the “Erroneous” state. As it can be seen in Figure 2, this pattern is built on the generic pattern presented in Figure 1. However, it differs by the specificity of its failure modes expressed in the transitions that are specific to “Data Exchange” fault category. Based on this pattern, another related pattern was derived to cover the specificity of other data exchange related faults such as delayed data transmission that will cause the function’s execution to be delayed. It such case, the “Erroneous” state was further split into several states depending on the specificity of the software component.

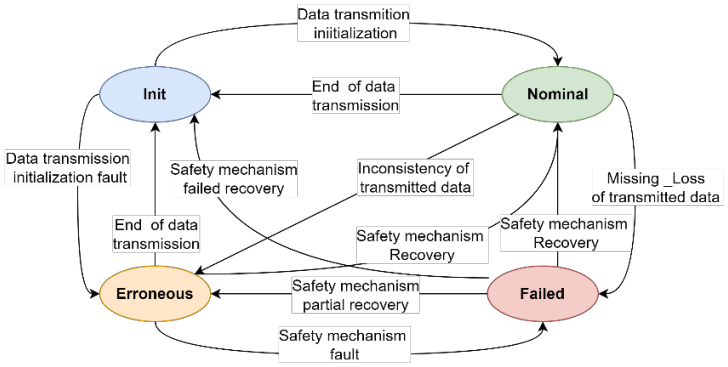


Figure 2. Data exchange fault pattern

The next pattern aims to capture the behavior of a generic software safety mechanism. The 4 states (Nominal Inactive, Nominal Active, Misleading and Failed) represent the execution state of the safety mechanism. In the “Nominal inactive” state, the safety mechanism is in its nominal execution state with no fault detected. When a fault

is detected, it moves to the “Nominal active” state. In this state, the safety mechanism is successful in reacting and correcting the effect of the fault. From this states, an erroneous or failed reaction will lead the safety mechanism to “Misleading” or “Failed” states respectively.

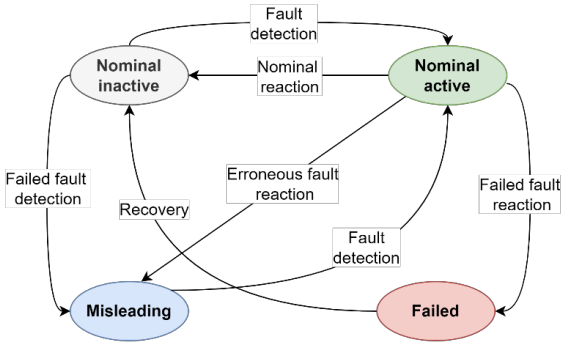


Figure 3. Generic safety mechanism fault pattern

The behavior of the safety mechanism is completed with writing failure propagation logic knowing their function and considering the associated fault modes identified earlier. To do so, we first need to study the safety mechanism’s function and identify their basic behavior, what their inputs are, and the result they produce in normal or faulty execution. Once done, we can write the failure propagation logic of the identified safety mechanisms. Depending on the expression of the safety requirements that the identified mechanisms are to fulfill, logical operators such as “or, and” and program control structures such as “if-then-else” can be useful to express the function or the combination of several mechanisms within a software component. While these operators are already part of the AltaRica semantics and can be expressed in assertions, modeling them in virtual bricks allows to graphically associate the components without rewriting the propagation logic. Therefore, we also need to write and add to the library, the failure propagation logic through theses operators and control structures. This will result in a library of safety mechanism, operators and control structures that will be use in the last step to construct the dysfunctional model of a system.

Modeling

Having opted for an approach based on a dedicated model, we must first identify the information necessary for its construction, starting with the software architectural design documents and the Technical Safety Concept (TSC) resulting from the system level safety assessments. The TSC is an aggregation of safety requirement specifications (often in textual and tabular format) from the system, as well as their allocations to hardware and software components and associated information (text, diagrams or sketches), which justify that safety measures and mechanisms are in place. Based on the TSC and the definition of the items, we can identify the safety-relevant components and interfaces to model, as well as the requirements and safety mechanisms to evaluate in the context of the dysfunctional architecture. In this way, we can represent in the dysfunctional architecture only those components that impact the safety goals, which are high-level safety requirements resulting from the preliminary risk analysis at the vehicle level (see ISO 26262-1 3.139). This will also avoid overloading the dysfunctional model with elements unnecessary for safety.

We used the readily available model bricks to model the patterns and their states in the tool, assigning the previously created domains to them. An overview of the modeled system along with the fault patterns is presented in Figure 4. Depending on the function of the pattern, different domains were used. Through the creation of AltaRica events in SimfiaNeo, we then modeled the transition firing conditions using the events identified in the state machines previously presented in the methodological proposal section. An AltaRica transition is characterized by a guard (condition to fulfill before triggering the effect), an effect (action resulting from the state change), and potentially a distribution (exponential, Dirac etc.) associated to the event. For each event, the SimfiaNeo tool allows to specify a probability that will be used during calculations. However, given in the context of software faults, such values are irrelevant and a probability of 1 was used instead. For each pattern, we wrote fault propagation logic linking the corresponding inputs (if they exist), internal states and outputs. We described the states of the inputs and outputs using the AltaRica domain that we named “Generic Data” and that incorporates four states: Nominal, Lost, Delayed, Erroneous. Using these states, we were then able to model the dysfunctional information flows between components trough AltaRica expressions. Using the elements stored in the library in an appropriate MBSA tool, one can easily model the dysfunction architecture of a systems through drag and drop. This is possible since the safety

mechanisms self-contain their propagation logics as well as the associated fault modes. Therefore, there is no need to write the failure propagation logic code in this step. We constructed the model presented on Figure 4 using the elements stored in the library as shown by the library icon in some of the components (e.g., component identified as ETH at the bottom left of the models). As an examples the components labeled CAN and ETH in Figure 4 were both modeled thorough an instantiation of the data exchange fault pattern described earlier as well as some internal components in the "vehicle-status-input" component that receive these data. Similarly, the "memory" component shown on Figure 4 as well as some subcomponents in the "longitudinal controller" component that read data from the memory were modeled though the instantiation of the data integrity fault pattern. Through these reusable libraries, we are able to model the dysfunctional behavior and failure propagation without having to manually rewrite their AltaRica code.

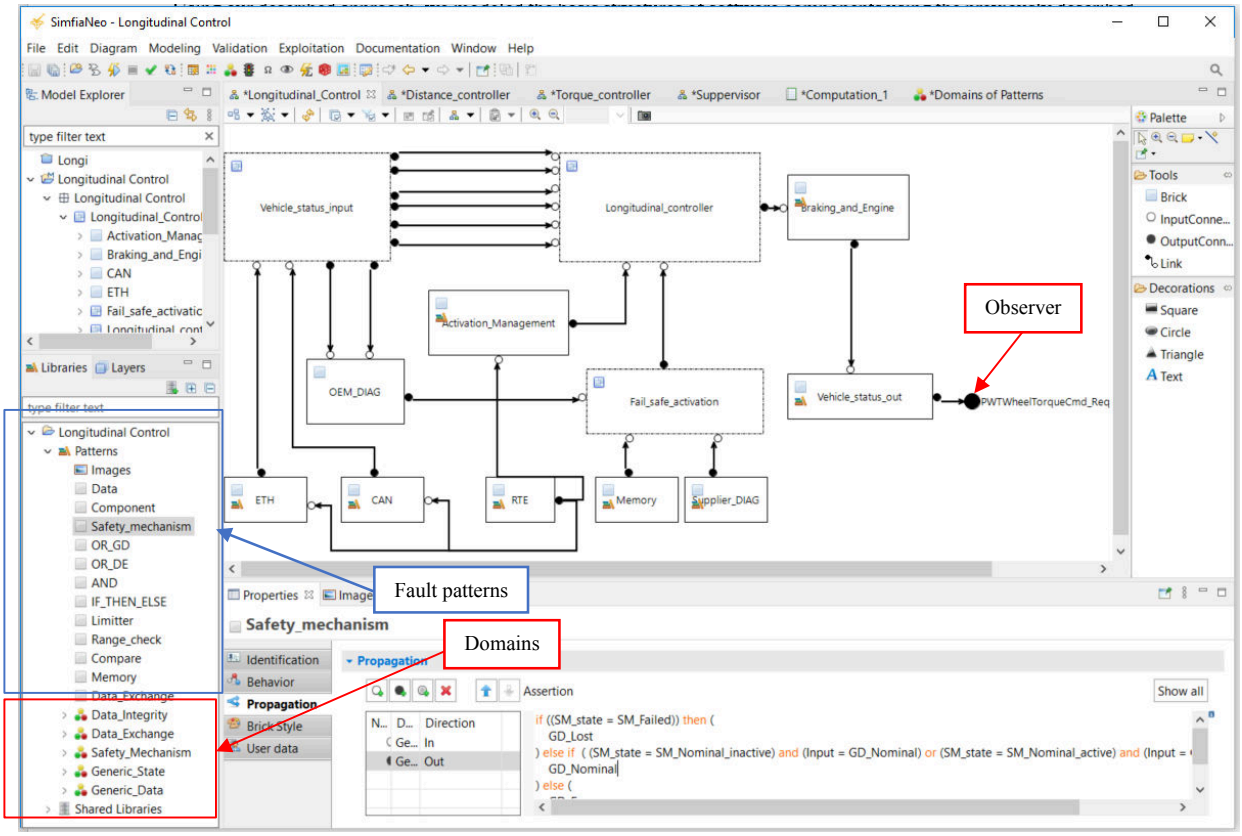


Figure 4. Pattern prototyping and dysfunctional model construction with SimfiaNeo

2. Safety Analyses

Using the SimfiaNeo Tool, we performed various safety analyses including step by step simulations, minimal cuts, FTA and FMEA based on the dysfunctional model constructed using the previously developed fault patterns. Having completed the modeling of the longitudinal control software component and the components that interact with it, the objective was to perform safety analyses from the dysfunctional model. For this purpose, we set up AltaRica observers on the outputs we were interested in (as shown in Figure 4). An AltaRica observer is an indicator that can be associated with a failure condition or feared event that we wish to capture. For example, let us consider, the UWSE that we have chosen for our case study related to an "unintended Acceleration > ISO 22179 acceleration limit while travelling (v $\neq$ 0 km/h) requested by the longitudinal control feature". In our model, we identified that the acceleration target and request in the speed controller subcomponent (Speed-Ctrl) are limited to 0.2G until vehicle speed vehicle is above 10 km/h. We also identified that the final value of the acceleration target is transmitted to the engine through the engine management command 'PWTWheelTorqueCmd' (Powertrain Wheel Torque Command). Thus, any erroneous value of 'PWTWheelTorqueCmd' can result in the violation of the safety goal and the occurrence of the UWSE. Therefore, the observers predicate to capture the occurrence of this UWSE can simply be 'PWTWheelTorqueCmd =Erroneous'. Having added the expression of the observer to the model, the objective was to verify, by means of the simulation, FMEAs, minimal cuts and fault trees, whether we could

determine the events or failures that could lead to the transmission of this erroneous command that can result in the violation of the chosen UWSE.

Simulations

Having run a series of simulations, we observed the propagation of failures to observers through the visualization of components in different colors (red: in the presence of a failure; orange: in error state; green: in nominal state), as shown in Figure 5. However, hierarchical components (consisting of blocks containing several subcomponents) appeared without color during the simulation. This step—although not overly formal—allows the model to be verified as it is being built. The analyst can then use it to quickly evaluate the dysfunctional architecture by visualizing how all the components and observers react to the presence of one or more failures at specific locations. The simulation can be used to confirm and demonstrate (for communication purposes) the feared scenarios identified with the classic methods (FMEA and fault trees) that we will discuss in the following subsections.

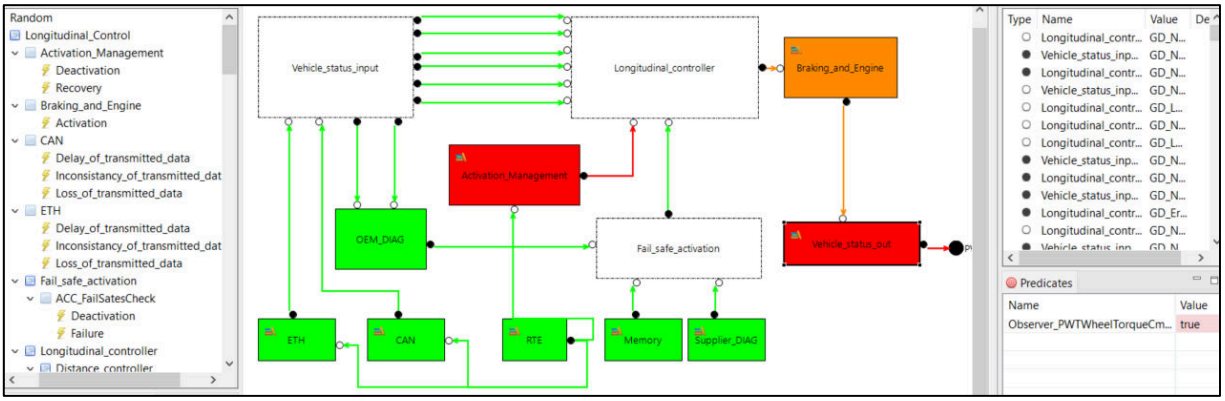


Figure 5. Simulation

Failure Mode and Effects Analysis

We used SimfiaNeo to generate the FMEA tables from the dysfunctional model we had built. The FMEAs list all the events resulting in the violation of a safety goal (or of a created observer), doing so for each component of the model. Figure 6 shows an excerpt from an FMEA, containing a certain number of elements typically found in these tables. The first column (Event) lists the events causing the violation. In the following columns, we can find the Local Effect (effect of the event on the output of the initial component), the Intermediate Effect (effect of the event on all intermediate components between the initial component and the final observer), and the Final Effect (effect on the output of the model; in here, the effect on the observers described previously). For instance, in relation to our chosen UWSE, the excerpt from Figure 6 shows how faults in the vehicle status input component related to vehicle speed can affect other components and the final observer.

Event	Local effect	Local effect val...	Intermediate ef...	Intermediate ef...	Final effect	Final effect value
Vehicle_status_input.Vehicle_speed_estimation.Loss_of_transmitted_data	Vehicle_status_input.Vehicle_speed_esti...	GD_Lost	Vehicle_status_i...	GD_Lost	PWTWheelTorq...	GD_Lost
Vehicle_status_input.Vehicle_speed_estimation.Delay_of_transmitted_data	Vehicle_status_input.Vehicle_speed_esti...	GD_Erroneous	Vehicle_status_i...	GD_Erroneous		
Vehicle_status_input.Vehicle_speed_estimation.Inconsistancy_of_transmitted_...	Vehicle_status_input.Vehicle_speed_esti...	GD_Erroneous	Vehicle_status_i...	GD_Erroneous		
Vehicle_status_input.ACCEngin_Perf_Status.Loss_of_transmitted_data	Vehicle_status_input.ACCEngin_Perf_Stat...	GD_Lost	Vehicle_status_i...	GD_Lost		
Vehicle_status_input.ACCEngin_Perf_Status.Delay_of_transmitted_data	Vehicle_status_input.ACCEngin_Perf_Stat...	GD_Erroneous	Vehicle_status_i...	GD_Erroneous		
Vehicle_status_input.ACCEngin_Perf_Status.Inconsistancy_of_transmitted_data	Vehicle_status_input.ACCEngin_Perf_Stat...	GD_Erroneous	Vehicle_status_i...	GD_Erroneous		
Vehicle_status_input.Parking_Brake.Loss_of_transmitted_data	Vehicle_status_input.Parking_Brake.Output	GD_Lost	Vehicle_status_i...	GD_Lost		
Vehicle_status_input.Parking_Brake.Delay_of_transmitted_data	Vehicle_status_input.Parking_Brake.Output	GD_Erroneous	Vehicle_status_i...	GD_Erroneous		
Vehicle_status_input.Parking_Brake.Inconsistancy_of_transmitted_data	Vehicle_status_input.Parking_Brake.Output	GD_Erroneous	Vehicle_status_i...	GD_Erroneous		
Vehicle_status_input.Status_input_swc.Failure	Vehicle_status_input.Status_input_swc.O...	GD_Lost	Vehicle_status_i...	GD_Lost	PWTWheelTorq...	GD_Lost

Figure 6. FMEA

SimfiaNeo can export this document as an Excel spreadsheet, allowing for better data processing and sharing. This is an important asset of the tool, considering that MBSA tools are not necessarily used by many but every engineer manipulates Excel files. Note, however, that Figure 6 represents only a very small excerpt from the initial FMEA, which has more than 18,000 lines. Therefore, if the goal is to obtain usable details or to make a synthesis, this representation is not ideal.

To analyses the usefulness of this FMEA, a comparison with a manually performed FMEA would have been interesting. However, in the context of current practice in our case, there are no software FMEAs performed using

the traditional approach. In contrast to fault trees that focus on one feared event, FMEAs are systematic and constitute a great way of showing that all failure modes have been accounted for within the system. This remains a difficult task for the safety analyst especially in the software context where failure modes can be plethoric. Despite the absence of a comparison with a manually performed FMEA, we argue that our approach allows performing this type of analysis that is otherwise difficult to perform manually.

Minimal cut set and fault tree

The generation of the minimal cuts is achieved through the configuration of the cut set and sequence calculation engine for a given observer. Thus, for an observer we can choose the maximum order of the cuts, the filter type (minimal cut or minimal sequence) and the generation type (combination, permutation or stochastic) which will be used during the cut set or sequence calculations. The maximum order corresponds to the maximum number of primary events in a sequence. To choose the maximum order, we experimented with values ranging from 2 to 5. We observed that SimfiaNeo load on the processor and memory consumption remained relatively constant (respectively close to 30% and 700 Megabyte) regardless of the maximum order value, while the computation time increased exponentially from under 2 minutes for order 2 to 7 hours for order 5. Meanwhile the maximum order in the resulting minimal cut set remained equal to 3 even if we consider sequences of size 4 or 5. We chose order 3 for our case study—the higher the order, the longer the generation of the cut will take. An order of 3 is therefore a good tradeoff between computation time and accuracy. The choice of the filter type is also important; we have chosen the "minimal cuts" option since it makes fault tree generation possible. Lastly, the choice of the generation type specifies the combinatorial or stochastic sequence (based on random simulations) used during the generation of the cut.

	Elements	Order ▲	Probability
1	RTE.Memory_access_fault	1	1.0
2	Longitudinal_controller.Speed_controller.Failure	1	1.0
3	Longitudinal_controller.Fail_safe_controller.Upper_range_value.Memory_access_fault	1	1.0
4	Longitudinal_controller.Torque_controller.Preprocessing.Loss_of_transmitted_data	1	1.0
5	Longitudinal_controller.Fail_safe_controller.Upper_range_value.Nominal_deactivation	1	1.0
6	RTE.Nominal_deactivation	1	1.0
7	Longitudinal_controller.Fail_safe_controller.Range_check.Failed_reaction	1	1.0
8	Longitudinal_controller.Torque_controller.Engine_torque.Failure	1	1.0
9	Longitudinal_controller.Fail_safe_controller.lower_range_value.Nominal_deactivation	1	1.0
10	Vehicle_status_out.Loss_of_transmitted_data	1	1.0
11	Vehicle_status_input.Target_speed.Loss_of_transmitted_data	1	1.0
12	Longitudinal_controller.Torque_controller.Dynamique_saturation.Failure	1	1.0
13	Longitudinal_controller.Fail_safe_controller.lower_range_value.Memory_access_fault	1	1.0
14	Braking_and_Engine.Failure	1	1.0
15	Vehicle_status_input.Status_input_swc.Failure	1	1.0
16	Vehicle_status_input.Vehicle_speed_estimation.Loss_of_transmitted_data	1	1.0
17	Longitudinal_controller.Distance_controller.Vstop_Vmin_Strategies.Failed_reaction	1	1.0
18	Longitudinal_controller.Distance_controller.Limitter.Failed_reaction	1	1.0
19	ETH.Loss_of_transmitted_data & CAN.Loss_of_transmitted_data	2	0.9999

Figure 7. Minimal cut

As an example, we considered the chosen UWSE linked to the transmission of an erroneous torque command to the engine. For this UWSE, we generated a minimal cut by choosing "order 3" as value of the maximum order, the "minimal cut" filter and "permutation" as the generation type. The generated minimal cut is shown in Figure 7. It shows combinations (of order 1, 2) of basic events that could cause the specified UWSE, as well as their associated probabilities (added by default). We can see that the cut highlights the events and the hierarchical components, enabling traceability of the components at high level (as shown in Figure 7). For dysfunctional models where several subcomponents have identical nomenclature, this traceability allows to clearly identify the origin of each event.

During the execution of these calculations, however, several compilation errors occurred, some of them due to the presence of loops in the failure propagation chain. This is a known issue related to the dataflow version of the AltaRica language. To solve this problem, we modified the assertions of the failure propagation involved in these loops. In the case of a redundant evaluation of a variable (where one of the assertion is part of the loop), removing the redundant evaluations of the involved variable allowed to break the loop. For this purpose, we considered a loop and identified the self-dependent variables in the chain of assertions constituting this loop. One possibility was to remove this variable from one of the assertions if it was already considered in another assertion. If this was not possible without modifying the validity of the assertion, the second possibility was to remove the dependency link

and successively assign to the state variable all possible values and perform the calculations with each scenario. In the latter case, it was necessary to manually change the value of the variable to include the scenarios which were excluded by assigning it a fixed value. In both cases, the dysfunctional logic of the assertion remains valid.

For a defined feared event, SimfiaNeo allows the generation of FTAs from the equivalent minimal cut. Through its tree structure and logical combinations, FTAs illustrate how basic events (located at the bottom of the tree) can lead to the feared event (at the top of the tree). In other words, FTAs highlight the causal chain between the basic events at the component level (at the bottom of the tree) and the high-level feared event (at the top of the tree) through a tree structure represented in graphic form. In SimfiaNeo, it was possible to generate an equivalent reduced fault trees from minimal cuts for a defined feared event. Nevertheless, we did not identify any added value through this generation as the minimum cuts in our opinion present the same information in a more concise and readable format. Furthermore, generating FTAs from minimal cuts can be considered counter intuitive as in practice safety engineers use the reverse process (they use FTAs to compute minimal cuts).

V. Discussion

The results obtained from the application to the case study shows that using fault patterns and the adequate tool, the dysfunctional model construction is made easy and safety analyses can benefit from this alternative new analysis method. The specificity of the case study demonstrates that the use of software-oriented fault patterns can benefit the application of MBSA in the automotive software safety context. Furthermore, tools such as SimfiaNeo relieve the safety expert of manual calculations while allowing him to concentrate on modeling. The benefits are reflected in terms of reusability of the models. Once the fault patterns are built, they can be reused to build the dysfunctional model and make it possible to conduct analyses with different parameters for many UWSEs based on the same model. In the traditional approach, the safety analyst spends much of their effort interpreting various design documents to manually construct classical model's safety models such as FTAs or FMEAs. The analyst would need to manually construct as many FTAs as there are feared events. If the design evolves, they will need to individually update all the fault trees and the FMEAs. Through our proposal, the dysfunctional model is easy to construct. If the system design evolves, the dysfunctional model can be updated, and updated safety analyses can be automatically derived. In addition, representing the behavior of the system without ambiguity is possible through the formal semantics of AltaRica, turning it into a possible candidate in a certification context. All these elements make this safety analysis method an interesting alternative that has the potential not only improve current practices but to contribute to the adoption of the model-based approach. Nevertheless, we noted some limitations on the method. One of the difficulties brought about by a dedicated dysfunctional model is maintaining its consistency with the design model when the latter evolves; this problem was already true when working on analyses based on fault trees and is not addressed by our proposal. Implementing additional measures is therefore necessary to guarantee consistency. Another limitation, related to the AltaRica dataflow, is the inability to natively manage loops; our solution was to modify some assertions in order to remove these loops. Finally, as the case study involves a system of reduced complexity, the scaling up of our methodological proposal is yet to be evaluated and we will need to perform a comparison with safety analyses results obtained through the traditional manual approach to fully evaluate the proposed approach.

VI. Conclusion

This paper made a methodological proposal based on fault patterns that can be used to build a dysfunctional model, and from which it is possible to derive classic safety models. Using the SimfiaNeo tool and the AltaRica language, the methodology was applied on a case study, building a dysfunctional model of a software from which we were able to generate FMEAs and minimal cuts. These results are encouraging and demonstrate the usefulness of patterns to facilitate MBSA model construction. More generally, they show that it is possible to apply an MBSA approach to evaluate software safety, especially in automotive applications. They also highlight the benefits of generating safety analyses from a dysfunctional model (time saving and reusability). Building on these results, the study must now continue to evaluate the complexity of the systems for which the methodology and the tooling can be reasonably applied. Since the proposal of this paper is based on a dedicated dysfunctional model, it will also be essential to supplement the method with a mechanism that ensures consistency between the design models and the safety models.

Reference

[1] C. Baron and V. Louis, “Towards a continuous certification of safety-critical avionics software,” *Comput. Ind.*, vol. 125, p. 103382, Feb. 2021, doi: 10.1016/j.compind.2020.103382.

[2] ISO, *ISO 26262 2018 Ed2 — Road vehicles — Functional safety*. ISO, 2018.

[3] P. Fenelon and J. A. McDermid, “An integrated tool set for software safety analysis,” *J. Syst. Softw.*, vol. 21, no. 3, pp. 279–290, Jun. 1993, doi: 10.1016/0164-1212(93)90029-W.

[4] M. Bouissou, H. Bouhadana, M. Bannelier, and N. Villatte, “Knowledge Modelling and Reliability Processing: Presentation of the Figaro Language and Associated Tools,” *IFAC Proc. Vol.*, vol. 24, no. 13, pp. 69–75, Oct. 1991, doi: 10.1016/S1474-6670(17)51368-3.

[5] A. Arnold, G. Point, A. Griffault, and A. Rauzy, “The AltaRica Formalism for Describing Concurrent Systems,” *Fundam. Informaticae*, vol. 40, no. 2,3, pp. 109–124, 1999, doi: 10.3233/FI-1999-402302.

[6] M. Bozzano and A. Villafiorita, “Improving System Reliability via Model Checking: The FSAP/NuSMV-SA Safety Analysis Platform,” in *Computer Safety, Reliability, and Security*, vol. 2788, S. Anderson, M. Felici, and B. Littlewood, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 49–62. doi: 10.1007/978-3-540-39878-3_5.

[7] B. Bittner *et al.*, “The xSAP Safety Analysis Platform,” in *Tools and Algorithms for the Construction and Analysis of Systems*, vol. 9636, M. Chechik and J.-F. Raskin, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 533–539. doi: 10.1007/978-3-662-49674-9_31.

[8] Y. Papadopoulos and J. A. McDermid, “Hierarchically Performed Hazard Origin and Propagation Studies,” in *Computer Safety, Reliability and Security*, Sep. 1999, pp. 139–152. doi: 10.1007/3-540-48249-0_13.

[9] N. Jiang, G. Li, and B. Liu, “Model-based safety analyses of embedded system using stateflow,” in *2016 11th International Conference on Reliability, Maintainability and Safety (ICRMS)*, Oct. 2016, pp. 1–6. doi: 10.1109/ICRMS.2016.8050084.

[10] A. B. Rauzy and C. Haskins, “Foundations for model-based systems engineering and model-based safety assessment,” *Syst. Eng.*, vol. 22, no. 2, pp. 146–155, 2019, doi: 10.1002/sys.21469.

[11] A. Legendre, A. Lanusse, and A. Rauzy, “Toward Model Synchronization Between Safety Analysis and System Architecture Design in Industrial Contexts,” in *Model-Based Safety and Assessment - 5th International Symposium, IMBSA 2017, Trento, Italy, September 11-13, 2017, Proceedings*, 2017, vol. 10437, pp. 35–49. doi: 10.1007/978-3-319-64119-5_3.

[12] G. Point and A. Rauzy, “AltaRica: Constraint automata as a description language,” 1999. Accessed: Nov. 28, 2019. [Online]. Available: http://www.altarica-association.org/ressources/ARBib/PointRauzy1999-AltaRicaConstraintLanguage.pdf

[13] M. Gudemann and F. Ortmeier, “A Framework for Qualitative and Quantitative Formal Model-Based Safety Analysis,” in *2010 IEEE 12th International Symposium on High Assurance Systems Engineering*, Nov. 2010, pp. 132–141. doi: 10.1109/HASE.2010.24.

[14] C. Kehren *et al.*, “Architecture Patterns for Safe Design,” *AAAF 1ST COMPLEX SAFE Syst. Eng. Conf. CS2E 2004 21-22 JUIN 2004*, [Online]. Available: http://130.203.136.95/viewdoc/summary;jsessionid=AF8F5506E5BFE636FDEC5DACA3E7DD02?doi=10.1.1.77.488

[15] P. Bieber, C. Bougnol, C. Castel, J.-P. H. Christophe Kehren, S. Metge, and C. Seguin, “Safety Assessment with Altarica,” in *Building the Information Society*, Boston, MA, 2004, pp. 505–510. doi: 10.1007/978-1-4020-8157-6_45.

[16] Y. Sirgabsou, C. Baron, C. Bonnard, L. Pahun, L. Grenier, and P. Esteban, “Investigating the use of a model-based approach to assess automotive embedded software safety,” presented at the 13th International Conference on Modeling, Optimization and Simulation (MOSIM20), Nov. 2020. Accessed: Feb. 01, 2022. [Online]. Available: https://hal.laas.fr/hal-02942695

[17] Y. Sirgabsou, C. Baron, L. Grenier, L. PAHUN, and P. Esteban, “L’ingénierie dirigée par les modèles pour assurer la sécurité des logiciels embarqués en automobile,” Grenoble, France, May 2021. Accessed: Sep. 29, 2021. [Online]. Available: https://hal.laas.fr/hal-03232108

[18] “Overview of Functional Safety Measures in AUTOSAR.” [Online]. Available: https://www.autosar.org/fileadmin/user_upload/standards/classic/43/AUTOSAR_EXP_FunctionalSafetyMeasures.pdf

[19] “Safety Use Case Example.” [Online]. Available: https://www.autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_EXP_SafetyUseCase.pdf

[20] M. Machin, L. Sagaspe, and X. de Bossoreille, “SimfiaNeo, Complex Systems, yet Simple Safety,” in *9th European Congress on Embedded Real Time Software and Systems*, 2018, p. 4.

"Pave the way for connected & autonomous driving at level crossings"

Virginie TAILLANDIER[1], Romain DEMARETS[2], Richard DENIS[3], Brouk WEDAJO[4]

1. SNCF, Innovation and Research department, France, virginie.taillandier@sncf.fr
2. VALEO, Autonomous Driving department, France, romain.demarets@valeo.com
3. VALEO, Autonomous Driving department, France, richard.denis@valeo.com
4. VALEO, Autonomous Driving department, France, brouk.wedajo@valeo.com

Abstract

France has among the highest number of level crossings in Europe (more than 15 400), representing 1% of road fatalities and more than 37% of railway fatalities (excluding suicide).
As increasingly intelligent, connected & automated vehicles are emerging on the roads, the SNCF (French Railway Operator) and VALEO (automotive equipment supplier) have joined forces to study how to prepare the arrival of such automated/autonomous vehicles at level crossings.
To enable a safe driving in such level-crossing areas by these automated/autonomous vehicles, impacts on both vehicle and infrastructure sides have been studied & demonstrated succesfully.
In particular :
- Wireless communications between vehicles & level crossings (called "V2X" communications) have been used in combination with exteroceptive sensors (e.g. camera, etc.)
- Railway & automotive functional safety and cybersecurity approaches have been mixed.
This article aims at :
- presenting some of the use cases & scenarios that need to be addressed by the automated vehicles in level crossing areas
- describing examples of countermeasures on both vehicle and infrastructure (level crossing) sides, that contribute to a safe automated driving at level crossing areas, and the general methodology to derive such countermeasures.

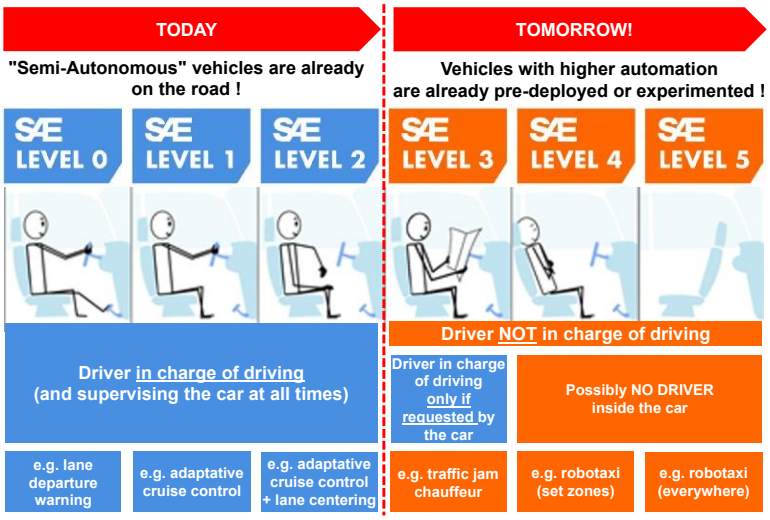
Key words :

Connected & Autonomous Vehicles, Autonomous Driving, Smart Level Crossing, Vehicle-to-Everything (V2X), Internet of Things (IoT), Cooperative Intelligent Transport System (C-ITS)

1. Introduction

1.1. Highly automated & autonomous vehicles are upcoming!

There are 6 different levels of driving automation (from level 0 to level 5), according to SAE (Society of Automotive Engineers). A step-by-step deployment from driving assistance to autonomous driving is expected.
As of today, many cars on the road offer a level of partial automation that allows the driver to delegate control of speed and/or steering (e.g. adaptative cruise control "ACC", ACC coupled with lane centering, etc.).
Cars with a higher level of automation are upcoming or even start being deployed, in which the "driver" may read a book (while the car is driving by itself), but still has to remain ready to take back control at any moment (only if requested by the car).
Moreover, vehicles with no driver inside, such as autonomous shuttles or robotaxis, are also being experimented and are expected to be deployed in a close future.



1.2. Level crossings : a quite complex type of intersection to handle

In France, the infrastructure managed by SNCF Réseau accounts for more than 15,400 level crossings (LC) on the operated railway lines (source : SNCF Réseau, 2019[1]).
France is one of the countries with the highest number of level crossings in Europe.

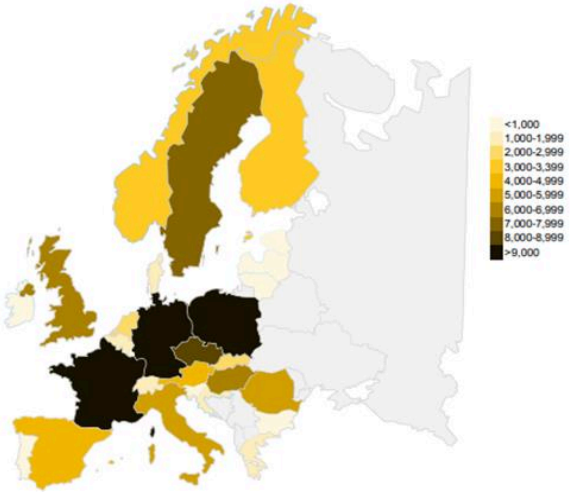


Figure 1 : Total number of level crossing in Europe, 2014 (ERA 2016)

On average, each year on the National Rail Network (NRN) of SNCF Réseau, there are 100 to 150 collisions (involving a train), leading to 25 to 30 fatalities.
Accidentology at level crossings has characteristics similar to accidentology on other roads, both in terms of driver profile (18-25 year old; those over 70; men) and in terms of cause (in particular, excessive speed or violation of traffic

[1] https://www.prevention-ferroviaire.fr/page/les-differents-passages-niveau

rules). Typology and configuration of level crossings (LC) does not appear as a particular factor of accidents : indeed, statistically 1% of accidents at LCs occur in LC registered in the national safety program (PSN)[2] each year. Moreover, whereas less than 6% of accidents out of LCs result in injuries and just 0,02% of them in fatalities, from 30% up to 50% of accidents at LCs (collisions with a train) lead to death.
The high probability of fatality in case of collision with a train currently discourage most car-makers to include the level crossings in the scope of operation of their automated vehicles (called "Operational Design Domain, ODD").
In a constant concern to ensure the safety of all road users who navigate at level crossings, SNCF started in 2018 a collaboration with VALEO to tackle accidents of conventional vehicles (ie manually driven), most of them being caused by voluntary or involuntary violation of traffic rules, human driving mistakes or distraction. For that purpose, V2X communications between the vehicle and the level crossing were investigated to enable driver warnings.
Faced with the rise of automated vehicles, a new collaboration between SNCF and VALEO started in 2020 to address these types of vehicles, and explore how to ensure their safe automated driving at level crossings.

1.3. State of the art (level crossings vs automated & connected vehicles)

Different projects or standardization initiatives that involve intelligent vehicles navigating at level crossings have been identified :
1) In 2018, Valeo and SNCF investigated V2X communications between a level crossing and a connected vehicle (manually driven), in order to provide information and/or warning to the driver (e.g. about a risk of collision with a train). This work was presented at the ILCAD conference and the at ITS European Congress in 2019.
2) Rail2X is a project launched in 2019 in Germany with members such as Siemens, Deutsche Bahn or DLR.
A target use case is to enable connected vehicles (manually driven) to request automatically the opening of a particular type of level crossing (with inverted operation) via a V2X communication. For this particular LC, barriers are closed by default and only open on road user's request, after validation by a remote supervision center if no train is on approach.
3) SNCF joined in 2019 the C-ROADS France consortium as an associated partner. SNCF has then led the specification of the "level crossing" use cases, in which the infrastructure communicates with vehicles (in V2X) for driver information & warning purposes.
As a summary, automated driving at level crossings is a topic not targeted yet by the C-ITS ecosystem (Cooperative Intelligent Transport System), which is rather focusing currently on conventional (manually driven) connected vehicles.

[2] “The list of LCs registered in the national security program” is the term that replaced in 2014 the “list of LCs of concern”. This list was created in 1997 following the Port Ste Foy accident to list the most “accident-prone” LCs, namely LCs with 3 collisions and more, or 15 collisions and more, or 1 collisions and 11 collisions minimum, or 2 collisions and 10 collisions minimum over 10 years or a time greater than 1,000,000.

2. Assumptions of this study

2.1. Assumptions on the level crossings (LC)

In France, the ministerial decree of March 18, 1991 defines 4 categories of level crossings[3]:

- 1st category comprising automatic level crossings with 2 half-barriers or 4 half-barriers (10,327), and guarded level crossings (757)
- 2nd category comprising unguarded level crossings ("St Andrew cross LC" with or without STOP sign) and SAL 0 (traffic lights only) (2 806)
- 3rd category: pedestrian level crossings (681)
- 4th category: private level crossings (834).

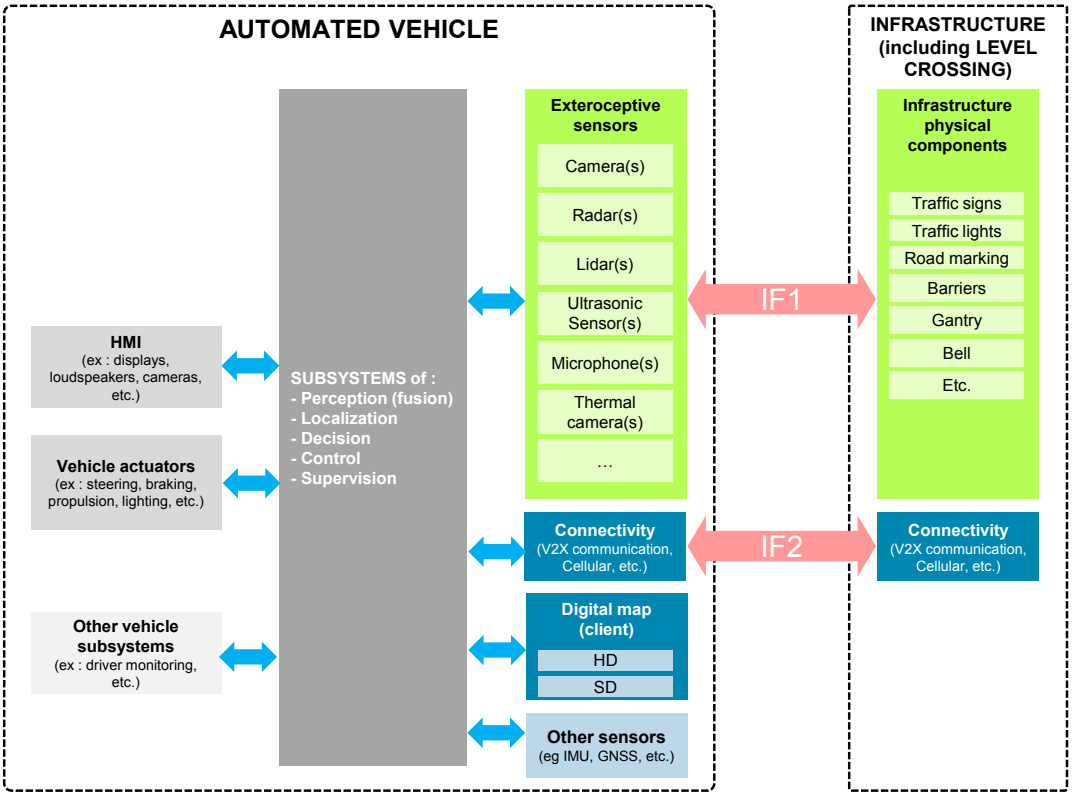
Each year, around 60% of collisions with a train occur at active/automatic LCs equipped with 2 half-barriers, where rail and road traffic is much more dense than at passive/St Andrew cross LCs. Such active LCs represent 60% of LCs.
In this project, we thus considered **active level crossings** in France (equipped with an automatic luminous signaling system with half-barriers).
To be noticed :
- even if the Geneva Convention aims at ensuring a certain homogenization in terms of level crossings equipments (e.g. fixed or flashing red lights, traffic signs, etc.), there are still differences between countries regarding their physiognomy (e.g. red & white barriers in France/Germany but yellow & red in Norway/Sweden) or their modes of operation (e.g. various flashing frequency for the traffic lights, various closing time, etc.)
- The delimitation of responsabilities between the managers of road and railway infrastructures
differs depending on the country. In France, the railway infrastructure manager is in charge of the level crossing facilities and its position signage, but the early signage (e.g. signs announcing a LC at 150 meters) is under the responsibility of the road infrastructure manager.

(a) active LC with 4 half-barriers & red flashing lights (b) passive LC with STOP sign (in France, from left to right)

2.2. Assumptions on the automated vehicles

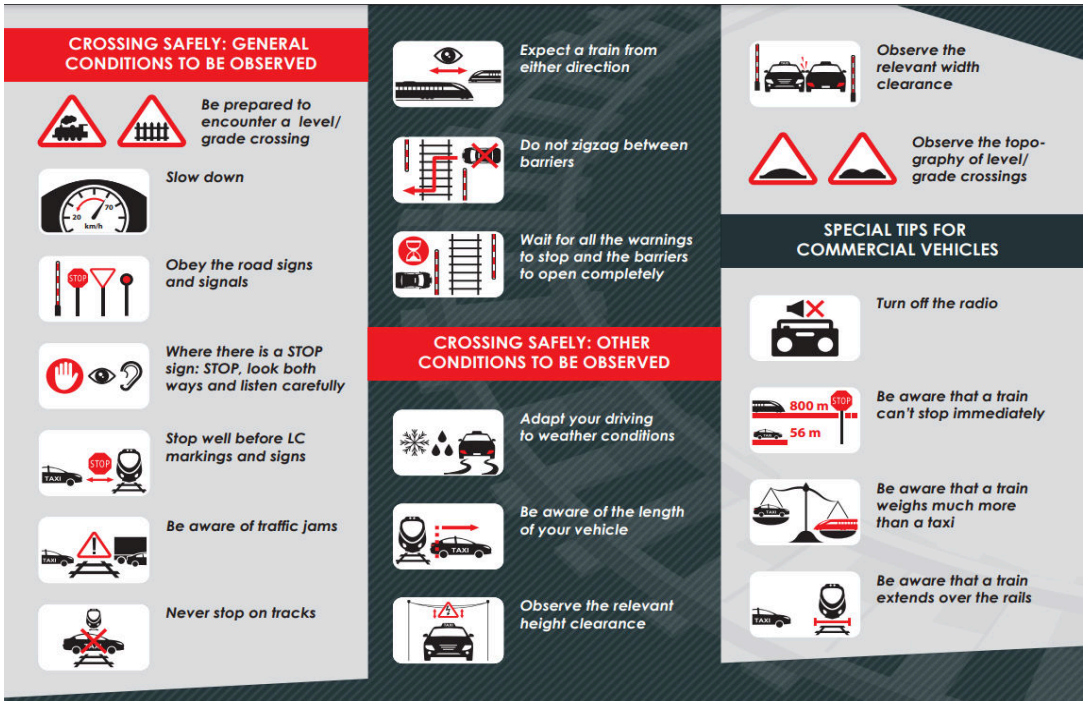
In this study, we considered :
- level 3 and level 4 automated vehicles, capable to cross an active level crossing
- level 3 automated vehicles, NOT capable to cross an active level crossing (because not designed for that purpose or because affected by a problem).
We also assumed the below preliminary technical architecture, in which the automated vehicle can interact with the infrastructure (including the level crossing) through 2 interfaces :
- IF1 : interface between its exteroceptive sensors and the physical infrastructure elements (eg road signs, etc.)
- IF2 : connectivity interface, through which information can be exchanged via a wireless communication.

[3] Source 2019



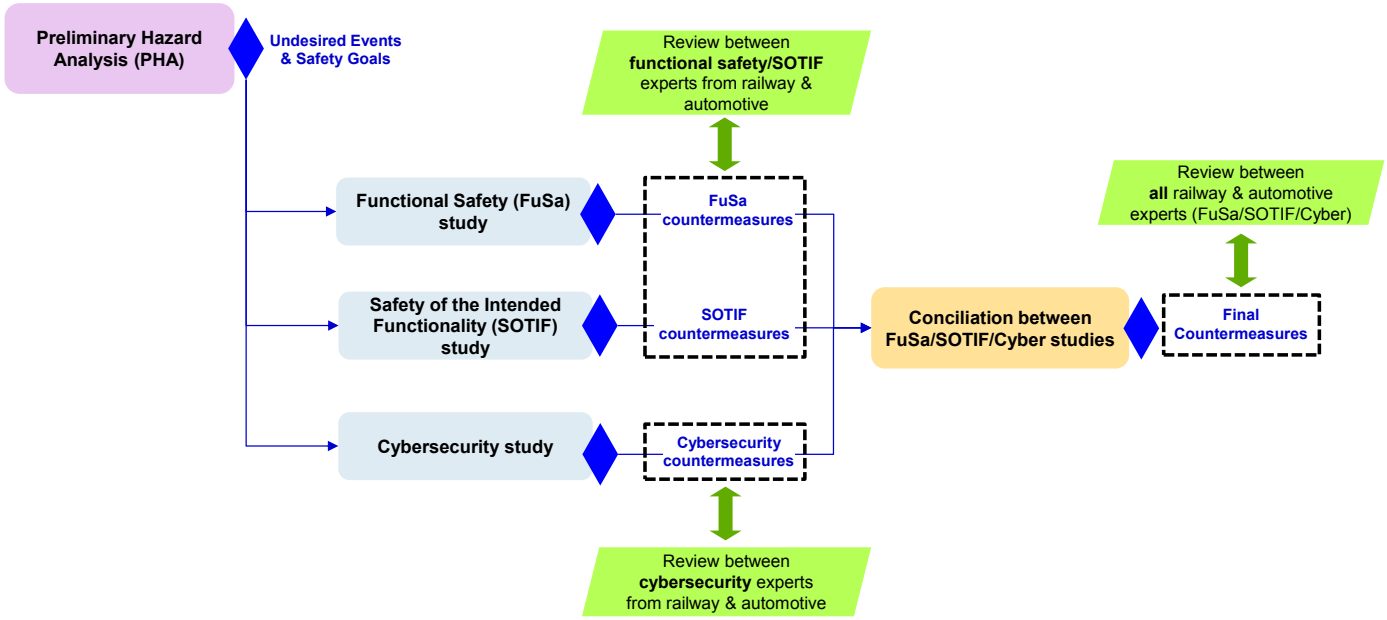
2.3. Expected behavior of the Automated Vehicle at a level-crossing

In level crossing areas, the automated vehicle is expected to obey traffic rules and have a careful behavior, as recommended in the following leaflet from the UIC (International union of railways).



3. Approach used to design a safe automated driving system

In order to design an automated driving system (ADS) capable to navigate safely at level crossings, the following approach has been followed :
1) An initial "Preliminary Hazard Analysis" (PHA) has been conducted to identify all hazardous behaviors (called "undesired events") of the ADS
2) Then 3 different types of dependability studies have been done in parallel :
A "functional safety" study, a "safety of the intended functionality (SOTIF)" study and a "cybersecurity" study, whose goals were to :
- identify & evaluate risks due to hazardous behaviors of the ADS that are caused by (respectively) random hardware failures, performance limitations of subsystems (especially sensors)[4], and cyber-attacks/malicious human behaviors
- define countermeasures that allow to reduce such risks at a "reasonable" level.
A focus has been done in this project on dysfunctions of the ADS perception of road & level crossing.
The methodology framework used for these 3 studies are based on (respectively) ISO 26262, ISO 21448 and ISO 21434 standards.
3) A conciliation phasis has finally been done between these 3 types of countermeasures, to ensure that they were consistent/not contradictory/complementary between each other.



This approach has been "vehicle-centric" : countermeasures have been investigated first on vehicle side, then complementary assistance from road infrastructure (especially from level crossings) has been explored.
In parallel, a functional safety analysis has been done in order to protect specifically the level crossing against new risks carried by the V2X connectivity.

This paper shows a non-exhaustive list of identified risks & associated countermeasures.

4. Hazardous behaviors of the automated driving system

Among all hazardous behaviors of the ADS ("undesired events") identified in the PHA, we propose to focus on the 2 below ones :

# ID	Undesired Event (at the Automated Driving System level)	Effect on vehicle level (in Level Crossing area)	ASIL ranking
UE-01	Cross the level-crossing when inappropriate (eg level-crossing is closed, exit not cleared, etc.)	Collision with train	ASIL A
UE-02	Cross the level-crossing with an inappropriate trajectory	- Run-off-road accident - Head-on collision or Side-collision with other road users	ASIL B

[4] SOTIF also addresses other causes of hazardous behaviors (e.g. misuse, etc.), not considered in this project.

5. Identified Risks

Among all hazardous scenarios and threats (cyber-attacks or malicious human behaviors with harmful intent) that may trigger a hazardous behavior of the ADS, especially due to performance limitations or vulnerabilities of the ADS perception of road & level crossing, the following ones have been selected.

5.1. Hazardous scenarios & SOTIF triggering conditions related to level crossing areas

ID	Hazardous scenarios / SOTIF triggering conditions	Effect (at ADS perception level)	Effect (at vehicle level)
TE_01	Crossing a level-crossing / Rail tracks with high contrast compared to road pavement	Rail tracks misinterpreted by cameras as road boundaries (due to high contrast with road pavement)	Cross the level-crossing with an inappropriate trajectory (UE-02) **=> Run-off-road accident or Head-on collision or Side-collision with other road users**
TE_02	Crossing a level-crossing / Traffic lights specific to Level-Crossings ("R24" traffic light)	R24 traffic lights not detected by cameras (due to specific physiognomy compared to conventional traffic lights)	Cross the level-crossing while inappropriate (UE-01) **=> Collision with train**
TE_03	Crossing a level-crossing / Phenomenons masking the level crossing traffic signs, warning & protection systems	Level crossing presence not detected or Level crossing status misinterpreted (due to phenomenons masking the level crossing equipments eg traffic lights/traffic signs/barriers/etc.)	Cross the level-crossing while inappropriate (UE-01) **=> Collision with train**

Illustration of hazardous scenarios in level crossing areas (from left to right) :
(a) rail tracks with high contrast compared to the pavement (b) R24 traffic light (c) parked vehicle masking R24 traffic light & barrier (d) vegetation masking the traffic sign

5.2. Threats to automated vehicles

ID	Threat
TH_01	Attack on integrity & availability of information received via V2X communication from the level crossing
TH_02	Attack on integrity & availability of information returned by the exteroceptive sensors

For illustration of the threat TH_02, McAfee managed in 2020 to spoof cameras of Tesla vehicles[5] by using "adversarial stickers", so that a "STOP" sign was misclassified as an "ADDED LANE" sign.
This kind of attack is known as "adversarial attacks" and exploit vulnerabilities of machine learning/deep learning algorithms.

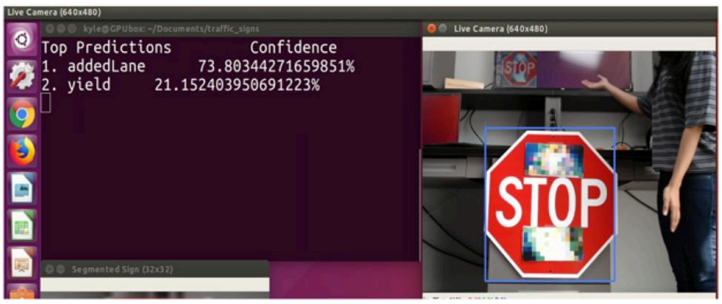


Illustration of an adversarial attack (source : McAfee)

[5] https://www.mcafee.com/blogs/other-blogs/mcafee-labs/model-hacking-adas-to-pave-safer-roads-for-autonomous-vehicles/

6. Countermeasures

6.1. Countermeasures on vehicle side

ID	**Countermeasure** (example)	**Risk addressed** FuSa	SOTIF	Cyber
CMV_001	Improve capability of cameras to : - detect the road boundaries in the presence of rail tracks with high contrast compared to the road pavement - detect the level crossing (presence/status) despite phenomenons masking the level crossing signs & systems (eg traffic lights/traffic signs/barriers/etc.) or despite particular physiognomies (eg R24 traffic light vs conventional tricolor traffic lights, etc.) (ex : algorithm improvement, etc.)		x	
CMV_002	Protect exteroceptive sensors against attacks on integrity & availability of information returned by the exteroceptive sensors (ex : protect cameras against machine-learning/adversarial attacks)			x
CMV_003	Protect V2X communications against attacks on integrity & authenticity of the received V2X information (ex : check signature & certificate of V2X information received from level-crossing)			x
CMV_004	Use redundancy for the perception of road & level crossing (ex : use multiple sensors with diverse technologies, use V2X communication, use a digital map, etc.)	x	x	x
CMV_005	Detect a non-capability of the automated driving system (ADS) to cross a level crossing and : - (if the ADS not activated yet) do not allow the ADS activation when approaching a level crossing - (if the ADS is activated) request the driver "early enough" to take back control when approaching a level crossing (for a L3 system) or stop the vehicle in a safe area before the level crossing or use an itinerary circumventing level crossings (for a L4 system)	x	x	x

6.2. Countermeasures on infrastructure side (e.g. level crossing)

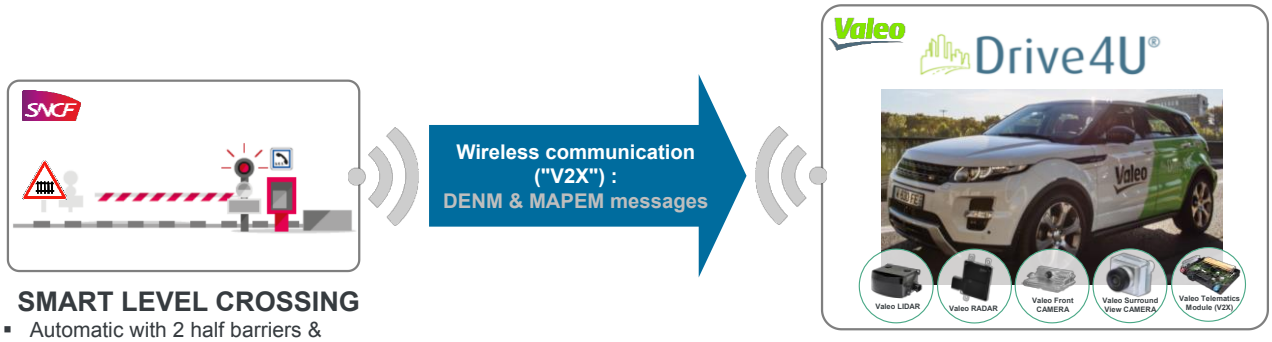
ID	**Countermeasures** (example)	**Risk addressed** FuSa	SOTIF	Cyber
CMI_001	Ensure that the road marking (eg edge lines) is present with a sufficient quality within the level crossing area		x	
CMI_002	Provide to automated vehicles : - information about presence & status (eg open/closed/etc.) of the level crossing - detailed description of the road within the level crossing area (ex : via V2X communication, digital map, etc.)	x	x	x
CMI_003	Protect V2X communications against attacks on integrity & authenticity of the emitted V2X information (ex : use signatures & certificates in V2X messages sent to automated vehicles)			x

7. Demonstrations

2 demonstrations were carried out in 2021 on a level crossing to validate the relevance of the previously described analysis and studies.

7.1. Set-up

The demonstrations took place at the level crossing "PN 449" in Brec'h (in Brittany, France), which is an active LC equipped with flashing lights and two half-barriers.
The level crossing management system was connected to a "Road Side Unit" (RSU), in charge of sending information about the level crossing area ("V2X messages") to the automated vehicle, via a wireless communication ("V2X communication"), standardized according to the ETSI ITS communication protocol stack.
The automated vehicle was also equipped with V2X connectivity, and could detect in particular the level crossing by using its exteroceptive sensors (e.g. camera) and a digital map in addition of such V2X link.



SMART LEVEL CROSSING

- Automatic with 2 half barriers & flashing lights
- Equipped with V2X connectivity

AUTOMATED VEHICLE

- Equipped with V2X connectivity
- Equipped with sensors (e.g. lidar, camera, radar, ultrasonic, GNSS, etc.)
- Detects the level crossing by using its sensors, digital map and V2X

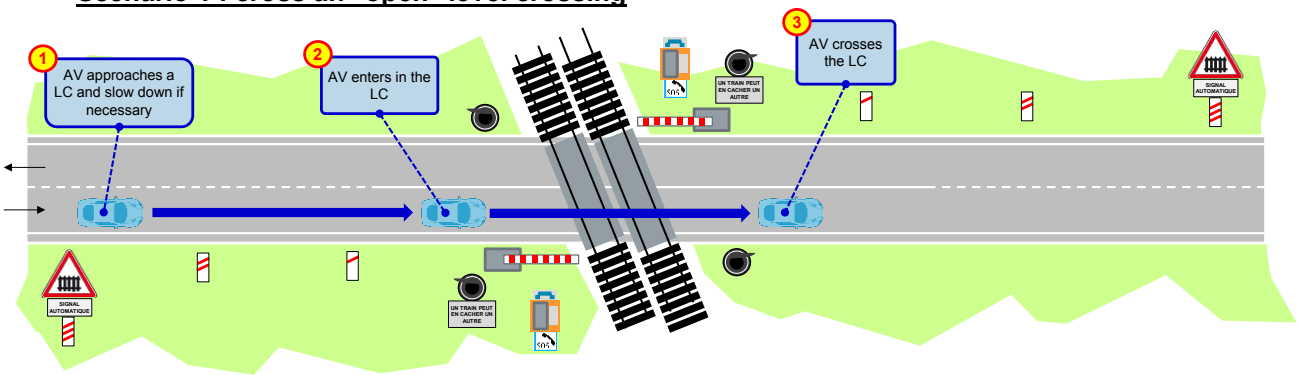
	V2X Message (EU)	Information provided	Standard
DENM	"Decentralized Environmental Notification Message"	Level crossing presence & status (open, closed, abnormal state, in maintenance)	ETSI EN 302 637-3
MAPEM	"MAP extended message"	Digital map of the level crossing area (road boundaries, level crossing entry/exit boundaries, etc.)	ISO TS 19091

Figure 2 Level crossing "PN 449" of Brec'h

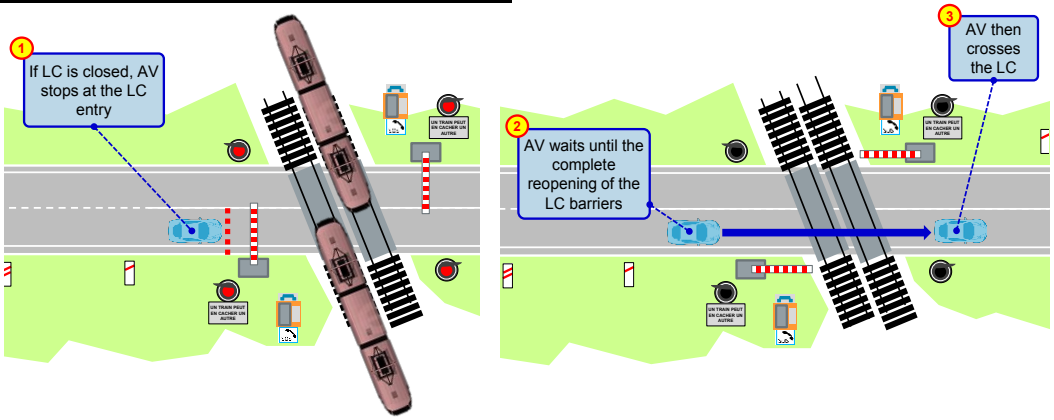
7.2. Target scenarios

A first set of scenarios were demonstrated, in which the "level 3" or "level 4" automated vehicle (AV) was assumed to be capable to cross the level crossing (i.e. without any driver's intervention or even without need of a driver) :

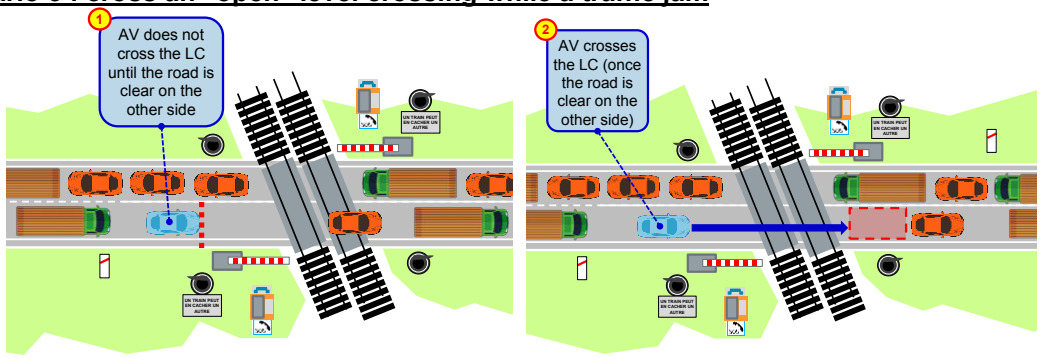
- **Scenario 1 : cross an "open" level crossing**



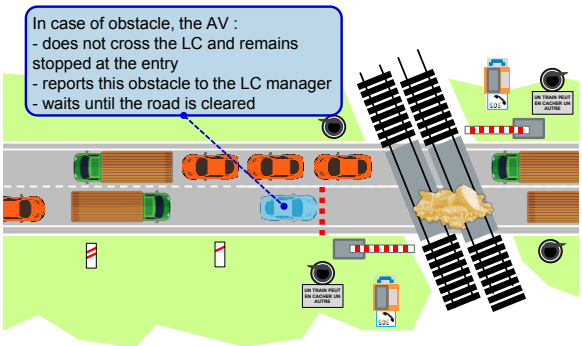
- **Scenario 2 : cross a "closed" level crossing**



- **Scenario 3 : cross an "open" level crossing while a traffic jam**

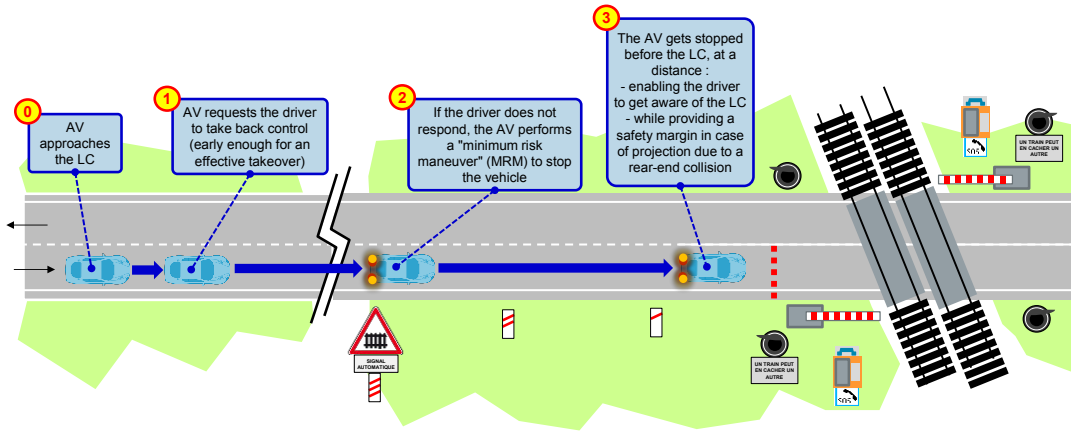


- **Scenario 4 : cross an "open" level crossing with obstacles blocking the way**



A second set of scenarios was also demonstrated, in which a "level 3" automated vehicle was assumed NOT to be capable to cross the level crossing, either because the automated driving system :
- is not designed for such purpose
- or is affected by a major dysfunction, e.g. sensor loss (before entering in the level crossing)

- **Scenario 5 : stop before the level crossing in case of non-capability**



7.3. Demonstration results

The target scenarios could be fulfilled by the automated vehicle with the expected performance in terms of functionality & safety (i.e. no hazardous behavior).
In particular, the assistance brought by the level crossing via the V2X communication link turned out to be especially useful to ensure the redundancy of the level crossing perception.
Indeed, the R24 traffic lights were not detected by one of the cameras of the automated vehicle[6].
Moreover, the use of already existing V2X standards (DENM/MAPEM), with a profile adjusted to the level crossing use case (i.e. selection of the relevant data fields & values among all offered by the DENM/MAPEM standards) allowed to manage all the target scenarios.
In particular, the range of such V2X communication (>300m during the demonstations) was sufficient to handle the minimum risk maneuver (MRM) scenarios, considering a "takeover request" duration of 10s and a deceleration at -4m/s².

8. Conclusion & Perspectives

This study has highlighted the need of redundancy for the automated vehicle perception of the level crossing, to ensure a safe driving at such level crossings, robust to both random hardware failures, sensors/systems limits of performance and cyber-attacks/malicious human behaviors.
In particular, the assistance from the level crossing via a V2X communication link to support the automated driving operations (both for the crossing and the minimum risk maneuver) is strongly recommended to speed up the deployment of automated vehicles at level crossings (by decreasing the implementation effort to be done on sensors and thus their cost). Indeed, reception of information about the level crossing and the road in the area offers perception redundancy to the automated vehicle, enabling them to compensate possible performance limitations of sensors (e.g. no R24 traffic light detection by cameras, etc.) or by reducing the probability of cyber-attack/malicious human behaviors (by increasing the level of complexity of the attack necessary to affect the ADS).
However, to be useful, the authenticity and integrity of this V2X communication (in particular the quality of the content of V2X messages broadcasted) must be protected.

Possible perspectives of this study could be :
- to iterate this complete safety analysis (as the technical architecture taken as input may have to be updated following this 1st round of analysis), until risk is reduced under a "reasonable" threshold
- to investigate the applicability of this study results to other countries than France. Indeed, even if the Vienna Convention ensures some consistency between a large number of countries, national differences in the signage of level crossings can be noticed. This point of attention must be taken into account for reasons of interoperability, especially in border areas
- to expand the list of possible countermeasures (both on the automated vehicle & level crossing sides), enabling to reduce the risk due to additional hazardous scenarios or attacks (taking into account national specificities)
- to investigate the different level of assistance that may be brought by the infrastructure/level crossing to support automated driving operations : indeed, depending on countries, the will of infrastructure/level crossing managers may differ (due to cost/benefit ratio of implementation of such assistance)
- to investigate how to conciliate two possibly contradictory points of view : providing assistance without exposing itself to liability risks / transfers of responsibility (from infrastructure managers' perspective), and receiving assistance from infrastructure with a sufficient level of quality to reduce the cost of automated driving systems (from automotive industrials' perspective).

[6] The level crossings of the SNCF network have 3 light technologies (diode module, LED lamp and filament lamp, currently being replaced by LED lamp).
Brec'h's LC 449 was equipped in March (Demo 1) with filament lights and in June (Demo 2) with LED lights. Before testing at Brec'h, Valeo carried out pre-tests on an LC from Bobigny fitted with diode module lights. It turned out that one of the automated vehicle cameras did not detect the R24 traffic lights with filament lamp nor LED lamp, but detected those equipped with diode module lights.

Session Th.4.B

Multicore

Thursday 2nd June

14:00

–

Room Lauragais

MASTECS Multicore Timing Analysis on an Avionics Vehicle Management Computer

Raúl de la Cruz*, Philip Harris*, Samuel R. Thompson†, Christos Evripidou†,
Tim Loveless§, Juan M. Reina‡, Mikel Fernandez‡, Enrico Mezzetti‡, Francisco J. Cazorla‡
*Collins Aerospace Applied Research & Technology, Ireland
†Rapita Systems L.t.d, UK
§Lynx Software Technologies, UK
‡Barcelona Supercomputing Center, Spain

***Abstract*—Driven by the increasing compute performance required by modern autonomous systems, high-integrity applications are moving to multi-core processors as their main computing platform. Using multi-core processors in avionics is particularly challenging since the timing behavior of the software is not only affected by its inputs but also by software running simultaneously on other cores. To address this challenge the MASTECS project has developed a methodology for multicore timing analysis together with a supporting toolset. In this work we show the results of evaluating this methodology and tools on a representative avionics use case.**

***Index Terms*—Multicore Timing Analysis, Airborne Software, Robust Partitioning, CAST-32A**

I. Introduction

Autonomy features such as Advanced Air Mobility, Single Pilot Operation, and others, are driving commercial avionics systems towards multicore processors (MCPs) in pursuit of higher compute performance. MCPs are increasingly the central element of computing platforms for commercial avionics systems [15], [20]. As described in the CAST-32A position paper [6][1] and DOT/FAA/TC-16/51 report [11], significant MCP-related challenges such as software timing analysis have to be addressed before MCPs can be fully embraced. The key challenge with MCPs is the timing behavior of a piece of software is not only affected by its inputs but also by the software running simultaneously on other cores.

The development process of modern avionics systems is a very costly and time-consuming activity, taking as long as 5 years from requirements to the final certification [9]. To meet safety certification, the deterministic timing behavior of newly-developed avionics systems must be extensively proven to airworthiness authorities (e.g. FAA, EASA). Software certification is a time-consuming and largely manual process whose cost dwarfs the typical embedded development process. The long process to generate evidence exacerbates the burden of certifying MCP products for aerospace suppliers.

The MASTECS project [1] has developed a structured analytical approach (methodology and tools) to produce evidence about multicore timing behavior. The MASTECS test methodology is specifically designed to capture the impact of multicore contention on application behavior. It hinges on the use of micro-benchmarks (highly-targeted qualifiable interference generators) to simulate configurable resource contention, while using built-in performance monitoring functionality on the processor to capture application response. Tests are carried out on-target using the RapiTime timing analysis tool, with automation of test execution and analysis from the RapiTest tool. The test framework incorporates the capability to trace tests and results back up to specific verification goals.

In this work we present the application of the MASTECS Multicore Timing Analysis (MTA) methodology and some specific tools matured during the project to an avionics use case provided by MASTECS partner Collins Aerospace. The use case, which runs on an NXP T2080 platform with the LynxSecure Separation Kernel Hypervisor, is an adaptable-baseline DAL-A (flight-critical) Vehicle Management Computer (VMC) able to host third party and legacy applications. Specifically, we show the results of making an iteration of the MASTECS seven-step testing process based on a V-shaped verification model. This includes multicore Critical Configuration Settings (CCS), Interference Channels (ICH), and Hardware Event Monitor (HEM) analysis; identification of timing requirements; test case design; implementation of test procedures; evidence gathering (testing); results analysis; and results validation and generation of documentation.

The rest of this work is structured as follows: Section II introduces the commercial state of the art in MTA and introduces some relevant academic works. Section III develops the MASTECS approach to address MTA challenges. Section IV introduces the main elements of the MASTECS tool chain used in each step of the proposed methodology. Section V describes the roles of each partner during the project. Section VI introduces the use case and its timing requirements, shows how the toolchain has been applied to address those challenges, and show the results obtained. Section VII performs a retrospective TRL analysis on the MTA tooling and its potential application on industrial cases. Section VIII presents the main conclusions of this work.

II. Positioning

MCPs, along with other accelerators integrated on the same System on Chip, can in theory provide increased compute performance, power efficiency, and space efficiency, as required in future avionics systems. However, tasks executing on different cores can slow each other down due to competition

[1]In 2022, CAST-32A will be supplemented/superseded by AMC 20-193, a joint effort by EASA and the FAA. See Notice of Proposed Amendment [5].

for shared resources like caches, interconnection transactions and bandwidth, and hardware accelerator utilization. This situation is even worse when deriving worst-case execution time estimates as allowance must be made to account for worst case scenarios which are unlikely but feasible.

One of the challenges for MTA lies in quantification, demonstration, and documentation of the impact of multicore contention. This entails the definition of an analysis and testing approach that produces evidence of the timely execution of the software on the multicore platform. However, mitigation mechanisms can have a large overhead, so there is a delicate balance to be struck between demonstration of high determinism and detrimental loss of performance.

Recent works focus on providing a certification framework that helps applicants to prepare their CAST-32A certification activities [4]. The proposed framework uses graphical notation diagrams to organize the argumentation. It also proposes developing evidence via automated analysis.

Typical successful approaches to timing analysis of single-core systems are based on either static analysis or test-driven on-target measurement. However, scaling these to complex MCPs has proven challenging. The complexity of current and upcoming MCPs has been acknowledged to present a complexity wall for static timing analysis solutions [19]. On the software side, increasing complexity, for example to promote autonomous features [18], challenges structural and syntactical analyses. On the hardware side, hardware complexity and opaque IP conspire to render derivation of accurate timing models intractable. Measurement-based tools also present specific challenges for multi-core timing analysis [2]. Those are related to developing a methodology, producing the required evidence and traceability, and generation of stress scenarios to derive trustworthy timing bounds. In this line several works focus on improving platform observability. That is, they propose measurement environments with low-intrusiveness that allow collecting detailed information about event monitors with support for visualization [10], [12]. This requires tight interaction with the RTOS or the debug technologies available in the underlying hardware [12].

To the best of our knowledge, no available tool in the market provides these capabilities, namely, analyzing the timing behavior of applications running in a multicore and capturing the needs of safety standards and certification requirements.

III. MASTECS MTA Methodology

MASTECS' goal is to deliver an industrial MTA solution for safety-critical embedded systems. In this Section we introduce the MASTECS methodology for MTA and in Section IV how it is implemented via the MASTECS toolchain. Throughout this paper we use several terms that we introduce in Table I.

TABLE I: Terms, acronyms, and their definition.

Term	Definition
AMP	Asymmetric Multi-Processing
CCS	Crtical Configuration Setting
COTS	Common Off The Shelf
CSP	Certification Support Package
DMA	Direct Memory Access
GPU	Graphic Processing Unit
HEM	Hardware Event Monitor
ICH	Interference Channel
IFC	Intended Final Configuration
MCP	Multicore Processor
MPSoC	Multi-Processor System on Chip
MTA	Multicore Timing Analysis
SBC	Single Board Computer
SoC	System on Chip
QoS	Quality of Service
RVD	RVS Database
RVS	Rapita Verification Suite
TCM	Task Contention Model
TRM	Technical Reference Manual
VMC	Vehicle Management Computer
VM	Virtual Machine

A. Platform Analyses

Building a solid understanding of the underlying hardware platform is instrumental as the first step in any MTA project. It helps gain insight on the existing ICHs and potential mitigation actions that can be implemented to control the impact applications can suffer due to contention on those ICH.

Platform analysis mainly builds on the available Technical Reference Manuals (TRMs) for the board, the System on Chip (SoC), and any other I/O controller that can be used by the target application. Besides the TRMs, some hardware vendors provide Certification Support Packages (CSPs) that provide specific information useful from critical domains from more detailed descriptions of hardware blocks to hardware fault related information. It is noted that CSPs might not be free of charge, indeed they can be quite expensive, and require signing specific NDAs. Also some hardware vendors also provide consultancy support to solve specific questions that may arise during the platform analysis as long as it does not reveal any protected information on the hardware behavior.

There are three main elements to be identified in the analysis: CCS, ICH and HEMs.

CCS analysis. It aims to determine the set of platform control registers that hold configuration data such that an unintended modification can result in the hosted software not to comply with its functional, performance and timing requirements. The goal is to use mechanisms to protect them from unintended modifications or propose appropriate means of mitigation if CCS are inadvertently altered.

HEM analysis. It captures the observability of the platform's ICH. This step aims to determine the particular HEMs that help understand how applications exercise different ICH. These allow the measurement of the load an application or micro-benchmark puts on the ICH, to show whether an ICH is mitigated by means of specific measures that may be in place. It is worth mentioning that HEMs are used as a main building block to provide evidence that micro-benchmarks work as expected, i.e. to validate micro-benchmarks. However, HEMs should not be automatically trusted to work according to their described behavior in the corresponding TRM. Some work [3] already reports mismatches between the definition of some monitors in the corresponding TRMs and the values

observed for specific experiments in the NVIDIA Jetson and Xavier MPSoCs, and the A53 in the Xilinx Zynq UltraScale+ MPSoC. Also errata documents [16] capture scenarios for the NXP iMX6 architecture in which certain performance monitors may not count events with precision. For the ARM A53 implementation in the Xilinx UltraScale+, some issues have also been reported with the instruction retired, store retired, and unaligned load/store retired event counters (among others) [21].

ICH analysis. MPSoC platforms in embedded critical domains already incorporate complex, high-performance, and Commercial Off-The-Shelf (COTS) hardware components including decentralized and distributed interconnects, deep shared cache hierarchies, DMAs, GPUs and other specialized, vendor-specific accelerators. Shared hardware resources are the root cause of multicore interference and so are the focus of timing analysis. ICH analysis builds on available technical information to first identify the main hardware shared resources in the platform (e.g. caches, interconnects, memories) and develop a description of the potential ICHs that exist in those shared resources.

The challenge lies in the fact that critical information for timing characterization is either not fully disclosed (to protect IP) or scattered across several documents. Furthermore, such data as is available can be relatively unclear and sometimes subject to errata. As a result, the list of potential ICHs identified have to be validated and characterized empirically.

Platform Analysis, i.e. CCS, HEM, and ICH analysis, drives the whole experimentation performed when following steps of the MTA process including the assessment of the impact CCS change on the application, the proposal of an ICH quantification plan, and the proposal of a HEM validation plan.

B. IFC Seclection

A smart selection of values for the identified CCS can both mitigate many ICHs while optimizing the performance of applications. As the hardware platforms become increasingly complex integrating more components, the number of CCS increases in every MPSoC generation. Also the variety of hardware features that can be controlled is wide. As representative examples, in the NXP T2080 [7], [8] the user can control the number of ways each core is allowed to use in the shared L2 cache, while the Xilinx Zynq UltraScale+ CCS allow control of quality of service (QoS) features that prioritize and route requests [17]. In the former case, deploying cache way partitioning prevents some of the ICHs in the L2 cache. In the latter request prioritization and routing prevents conflicts on the paths between different sources (e.g. computing elements application processing unit and the real-time processing unit) to a target (e.g. memory).

While intuitively cache partitioning, for instance, helps mitigate contention, it also can cause an application running in a core and confined to use a subset of the L2 cache to increase its number of L2 misses. As a result, the particular number of ways to assign to each core – which is configured via control registers (CCS) – depends on the particular application under consideration. Fixing CCS involves an iterative process in which several values are empirically evaluated until a good balance between isolation and performance is found. In MASTECS, we built on micro-benchmarks and the TCM to automate this process as covered in Section IV-C.

C. V Model

MASTECS methodology follows the standard V-model for software development life cycle with a well-defined set of analysis and test design activities, on the left side of the V, matched with corresponding verification and validation steps, on the other side. The fundamental steps in MASTECS MTA process model are summarized in Figure 1.

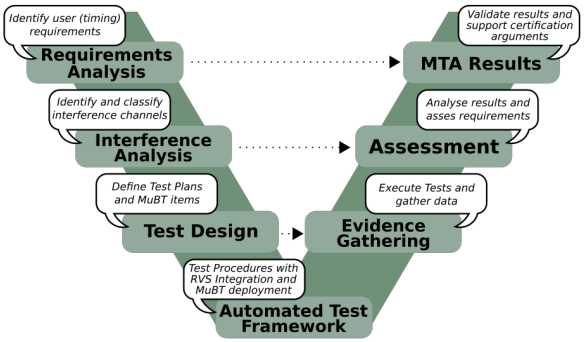


Fig. 1: Steps proposed by MASTECS for MTA in the V-model software development process.

① Firstly, the user timing requirements are identified, which allows the remainder of the MTA process to be scoped appropriately. These can range from validating that a given cache partitioning mechanism prevent data evictions between applications to show that a micro-benchmark puts the desired level of load on a given ICH.

② The second step entails identifying ICHs through which interference between cores can take place. Also at this stage, the HEMs necessary for the analysis are identified, as are the CCS present in the platform. Note that Section III-A builds the knowledge on the potential ICHs in the platform, while this step specifically focuses on those ICH related to the timing requirement being addressed. It also leverages HEM analysis to restrict the analysis to the HEMs that capture the load on the ICHs identified as relevant.

③ The third step in the MASTECS methodology is to develop test cases to verify hypotheses supporting the user requirements, which includes defining the micro-benchmarks that will be used to exercise the ICH. Alongside ICHs, and HEMs, micro-benchmarks are the main elements in the test case argument.

The ④ fourth and ⑤ fifth steps focus on the implementation of the test procedures and their automated execute on the platform to gather test evidence. MASTECS exploits the Rapita Verification Suite (RVS) framework, from project partner Rapita Systems Ltd (RPT), to automate the execution of large batches of tests and the collection of raw information from the execution of the program under analysis on the real platform and configuration.

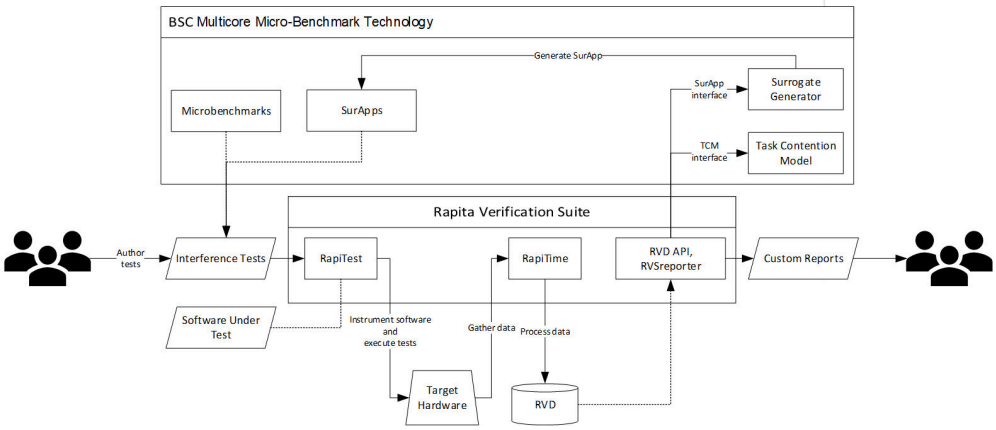


Fig. 2: MASTECS Toolchain.

In step ⑥, raw numbers are analyzed by technical expert to assess whether they prove (or otherwise disprove) the verification requirements. While the analysis step is only partially automated, it greatly benefits from RVS framework capability of providing different views and statistics of the gathered data.

The last step ⑦ involves a review of requirements, generation of certification artifacts to support the safety argument of the system. A characteristic feature of MASTECS MTA is that the whole analysis process is oriented towards fulfilling qualification and certification requirements as defined by domain-specific standards and regulations. Raw numbers and analysis results are presented as fundamental evidence to support a domain-specific certification arguments.

A key theme in the MASTECS MTA approach is the combination of the efforts of software timing analysis experts and hardware experts. This allows provision of the required insights into the behavior of modern complex MCPs running complex software. Hardware experts identify the ICH in the different hardware shared resources and any configuration options that may affect them – CCS according to CAST-32A. When it is determined than an ICH can be exercised by an application under test, hardware experts propose suitable HEMs to track contention in those ICH and a micro-benchmark design to put load on ICHs.

IV. MASTECS MTA Toolchain

The MTA toolchain supports the MTA methodology. It builds on the combination and integration of the RVS and Barcelona Supercomputing Center (BSC) Multicore Micro-Benchmark technology (MμBT) and multicore hardware knowledge. As shown in in Figure 2, Rapita's RapiTest lets the user to define the tests to carry out that usually involve the software under test (i.e. the application) and a micro-benchmark [2]. Rapita's RapiTime takes care of instrumenting the application according to user specification that captures the points of instrumentation and the specific HEMs to record at each point. The data collected from the execution is loaded in to an RVS Database (RVD). The RVS Exporter queries the RVD to provide information according to user's desired views. Information from the RVD is also provided as feedback on demand to BSC tools like the TCM and the Surrogate (Application) Generator that work iteratively.

[2]The Surrogate Applications (SurApps) are a type of micro-benchmarks that aim to copy the load that a reference application puts on the ICHs. In this work we do not assess SurApps.

A. Hardware Analysis

The hardware analysis process presented in Section III-A cannot be automated, that is, it is not possible to process TRMs to automatically extract CCS, HEM and ICH information. Hardware analysis is to be performed manually by hardware experts who should be providing insightful information about the hardware behavior.

Assessing the accuracy of the analysis is also difficult. Not only does it depend on the information made available to the hardware experts via the TRMs, CSPs, and consultancy support from the hardware provider, but also different hardware experts can produce slightly different conclusions in terms of ICHs. This risk can be mitigated by performing the analysis by different set of hardware experts which then combine their analyses into a single hardware analysis document encompassing CCS, ICH, and HEM analysis. It is also the case that robust guidelines on how to perform the analysis and reference analysis documents derived from previously analyzed processors can significantly help.

It is worth mentioning that, excepting the CSPs, none of the information used for ICH, HEM, and CCS analysis is intended for the purpose for which it is used in MTA. For instance, no section in the TRM captures ICH specifically. Instead, TRMs provide descriptions of how different hardware blocks interact. This description is provided to the level needed by software engineers to optimize the average performance of its applications or provide some QoS, as such, the TRM description is insufficient to provide all the details needed for ICH analysis.

This can be mitigated by performing a solid set of experiments to complement the analyses. HEMs are intended

for general performance analysis and debugging purposes and usually lack descriptive information on exactly what events they track [3]. Previous experience and experimentation is needed to consolidate a set of trusted HEMs to use in the rest of the MTA process.

B. Multicore Micro-Benchmark Technology

MμBT is a suite of software tools that cover the low-level (hardware) aspects of MTA. In this section we cover micro-benchmarks, which are some of the main building blocks for MTA. Micro-benchmarks are single-behavior pieces of code that stress a specific ICH (shared resource), see Figure 3. By running the micro-benchmark against an application, one can assess the sensitivity and aggressiveness of the application to contention in a given ICH. Micro-benchmarks are specifically tailored to the hardware/software target configuration (IFC), and are a key tool to determine the bounds on the impact of ICHs and for assessing the effectiveness of interference mitigation techniques.

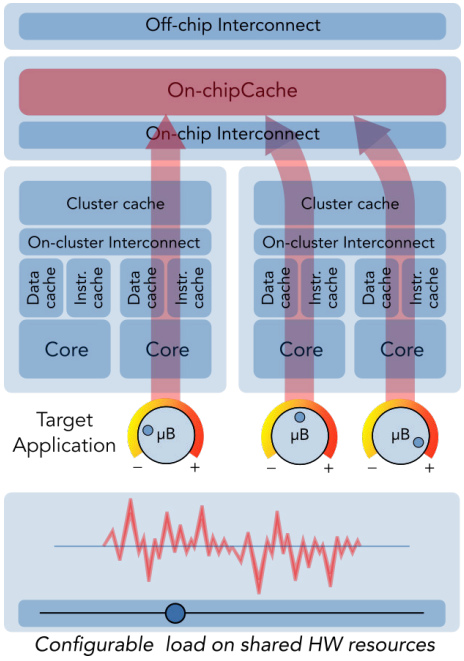


Fig. 3: Micro-benchmarks.

HEMs are used to assess the load micro-benchmarks put on ICHs and in general to validate the correct behavior of the micro-benchmark. Besides HEM-based validation a test harness is performed for the functional validation of each micro-benchmark under different scenarios.

PMUlib is a low-level library for configuring and reading of performance monitor counters serving as an access point to the available HEMs. It also supports an interface with tracing/debug units where present in processors (e.g. MultiCore Debug Solution in the Tricore AURIX TC39xx family) or external (Lauterbach). Reading HEMs can be performed in-band, i.e. from the software system under evaluation or via out-of-band debugger facilities [12] to prevent any probe effect.

C. Task Contention Model

The profiling information collected over the application in isolation can be conveniently exploited to provide early bounds on multicore timing interference incurred by the same application when deployed in a specific multicore scenario and under a given process schedule. The TCM exploits information on both the target HW and task model run-time to build a conservative analytical model for computing multicore contention. The model is not meant to provide exact estimates of multicore contention but early figures that can steer design and deployment decisions. The model, which is parametric on tasks' profile and schedule, allows fast exploration of any possible system configuration in view of reducing interference and optimizing the system makespan. The TCM aims at identifying a subset of candidate system configurations on which to focus the verification and validation efforts.

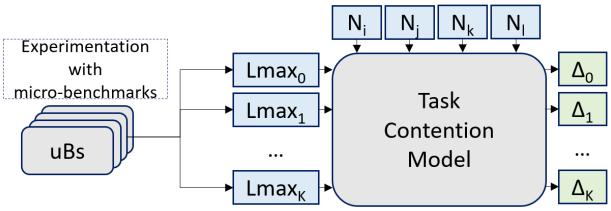


Fig. 4: Task Contention Model.

More in detail, the TCM builds on a timing estimate in isolation C_i^{isol} of a given task τ_i to derive a estimate of τ_i's execution time (C_i^{mcp}) when deployed in a specific multicore workload derived as $C_i^{mcp} = C_i^{isol} + \Delta_i$. The latter addend, Δ_i, is the composition of two elements, see Figure 4, (i) the longest contention different request types can suffer when accessing a shared resource r_i, $Lmax_i$,which is derived via execution of micro-benchmarks; (ii) the maximum number of requests, n_i, performed by τ_i and its contenders running in the other cores that are derived using PMULib (note that N_i is the maximum number of access of task τ_i to each resource, i.e. $N_i = \{n_{i,0}, n_{i,1}, ..., n_{i,k}\}$, where k is the number of shared resources. For instance, for a two task workload and one shared resource, the number of requests from τ_i that collide with co-runner τ_j in the access to the resource r_0 is defined as $min(n_{i,0}, n_{j,0})$. Hence, we can derive Δ_i as $min(n_{i,0}, n_{j,0}) \times Lmax_0$. The interested readers are referred to [13] for more information details on the fundamentals of the TCM.

D. RapiTest

RapiTest is a test-harness generator, capable of generating and driving both unit tests and system tests on-host and on-target.

For the MASTECS case studies, RapiTest was used to generate test harnesses and perform the necessary code injections to execute micro-benchmarks and collect timing data and HEMs as defined in the input interference tests.

RapiTest supports a range of native test input formats, in addition to automatic converters for a range of widely-used

test formats. For the work documented here, RapiTest was configured to use two test formats designed explicitly for multicore testing. These formats allow simple description of locations at which micro-benchmarks should be injected to generate contention, the functions and call-trees that should be instrumented, and the data that should be collected at these locations.

E. RapiTime

RapiTime is an on-target timing measurement tool, which integrates static analysis of source code with on-target instrumentation for both timing and resource usage into a single hybrid timing analysis tool. Using RapiTime, it is possible to configure the automatic injection of instrumentation based on one or more pre-selected instrumentation profiles into certain functions, syntactic structures, or even whole call-trees. The instrumentation can be tightly controlled to minimise overhead, for example by only instrumenting certain locations, or by minimising the number of instrumentation points that make higher-overhead resource-usage measurements.

Many targets (including the T2080 used in this study) have a hardware limitation on the number of HEMs that can be collected at a time. To support this, RapiTime incorporates the concept of *metric groups*. A *metric group* is a selection of HEMs that can be collected simultaneously. If more than the maximum supported number of metrics is required, then RapiTime can split these resources into metric groups, repeat tests per metric group, and aggregate the data into a single report database.

RapiTime results are stored into an RVD database, which can be graphically interrogated using the RVS report viewer, or programmatically using the Python API. Custom text-based reports can be generated by the accompanying *rvsexporter* tool using a simple combination of Markdown and Python.

F. RVS Reporting Tools

All the RVS tools generate report files as RVD databases. To allow these databases to be interrogated, a few reporting tools are provided.

1) RVS Report Viewer: **RVS Report Viewer** features an interactive user interface that allows the user to explore the test results stored in an RVD database. For RapiTime reports, the report viewer gives access to (among other things):

- **Execution time:** Statistics for the maximum, average, minimum, and high watermark execution time for each instrumented function.
- **Contribution time:** The contribution to the overall execution time of the nominated roots made by each of the instrumented functions.
- **Invocation Timeline:** Per-invocation execution time data for each instrumented function.
- **Execution Time Profile:** Histograms showing the distribution of observed execution times for each instrumented function.
- **Metrics:** A range of visualizations for metrics collected from any available HEMs.
- **Coverage:** While RapiTime doesn't give the depth of information that RapiCover does, RapiTime instrumentation allows determination of which instrumented functions were executed and which were not.
- **Report Comparison:** Two reports containing measurements of the same thing can be compared. For example, a RapiTime report containing data from a test run when nothing was executing on the other cores of a system could be compared with another report when the same software was executed, but interference generators were running on the other cores.

From the RVS Report Viewer, it is possible to generate generic exports using the built-in default exporters. It's also possible to copy data tables directly into other software (e.g. a spreadsheet) for further analysis.

2) RVS Exporter: **RVS Exporter** is a means to generate custom report exports. RVS exporter takes an input template as a markdown file. This markdown report can contain embedded Python code that is evaluated when the template is processed by RVS Exporter. This embedded Python code has access to all the data in the results database, and the API also provides useful utility functionality to make accessing, processing, tabulating, visualising, and reporting on the data as simple as possible.

If some tests are repeated, it is a simple matter to run the RVS Exporter tool again to re-generate the report using the latest data. The modular structure of the reports also facilitates reuse of report fragments or processing algorithms between reports without unnecessary duplication.

V. MASTECS Partners

The MASTECS consortium comprises two technology providers (BSC and RPT) and two end users (Collins Aerospace and Marelli Europe) in avionics and automotive who assessed the readiness the MASTECS MTA technology by evaluating it on their corresponding use cases. While it is not a partner of MASTECS, Lynx Software Technologies is also listed below as they contributed to the study presented in this work.

Barcelona Supercomputing Center (BSC) (Coordinator) is a leading research center in high-performance counting and embedded systems. BSC led the hardware analyses and matured the micro-benchmark technology, including the PMUlib and the TCM, to reach a high level of industrial readiness. In order to ensure a clear exploitation path of its technologies BSC created a spin-off company, Maspatechnologies S.L., during early stages of the project.

Rapita Systems Ltd. is a leading provider of software verification tools and services globally to the embedded aerospace and automotive electronics industries. Rapita has lead the productization of the technologies by developing tooling, processes, DO-178C documentation, tests and commercial infrastructure to bring a whole solution to an exploitable position within the market.

Collins Aerospace Applied Research & Technology (Collins-ART) is the innovation organization of Collins

Aerospace, a Raytheon Technologies company, leader in providing advanced solutions for the global aerospace and defense industry. Collins-ART has actively participated in MASTECS as end-user for the aerospace industry. Its main role was on setting avionics requirements; providing a representative aero case study; and demonstrating the effectiveness and soundness of the MTA toolchain during the evaluation.

Marelli Europe SpA. Marelli is one of the world's leading global independent suppliers to the automotive sector. With a strong and established track record in innovation and manufacturing excellence, Marelli's mission is to transform the future of mobility through working with customers and partners to create a safer, greener and better-connected world. Marelli has actively participated in MASTECS as end-user for the automotive industry. Its main role was on setting automotive requirements; providing a representative automotive case study; and demonstrating the effectiveness and soundness of the MTA toolchain during the evaluation.

Lynx Software Technologies specializes in real-time embedded safety-critical software. Lynx's contribution to the project was the LynxSecure product – a type 1 (bare-metal) hypervisor – as well as design, integration and support assistance. Such a hypervisor provides robust space partitioning without needing an RTOS, thus reducing HEM noise and allowing ICH (time partitioning) to be studied with higher fidelity. MASTECS used LynxSecure to partition the T2080 SoC's RAM, cores, peripherals and L2 cache hardware into bare-metal VMs.

VI. VMC Case Study

This section provides a summary of the case study evaluated including the platform where it runs, the particular instantiation of the MASTECS tool chain to cover the case study's requirements, and the results obtained.

A. Introduction to the Case Study

The case study builds on a redundant Flight Control System designed to replicate workload reduction applications at different levels of the system architecture: integration unit (VMC) and Single Board Computer (SBC). Each SBC board runs the system in an Asymmetric Multi-Processing fashion (AMP), and is able to deploy and run simultaneously mixed-criticality applications with different assurance levels (DAL-A/C).

Figure 5 shows the software architecture implemented for the VMC. Core 0 runs a process dedicated to I/O scheduling and data marshalling using FIFO queues and shared memory regions. The remaining cores of the SBC are dedicated to host applications accessing I/O through the queues provided by Core 0. Tasks are fully virtualized and executed on a Virtual Machine (VM) by the LynxSecure hypervisor, providing task domain isolation. The hypervisor allows the T2080's cores to be oversubscribed to host the 16 VMs of our case study. Each hosting core runs a VCPU manager (blue circle) that orchestrates and schedules computational tasks (green circles) in a pre-defined order to enforce data flow consistency. Lynx's Z-Scheduler is used on each VCPU manager to implement

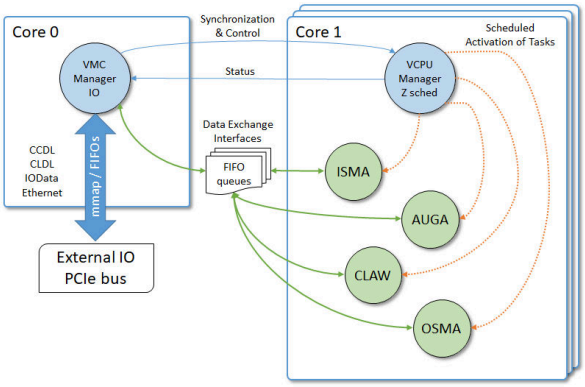


Fig. 5: Software architecture for each SBC of the VMC.

custom VM scheduling via time donation and as a convenient integration point for the HEMs and RapiTime tool.

B. The Target Platform

The use case runs on an NXP T2080 processor [8], see Figure 6 which comprises 4 cores each its own private instruction and data cache. The L2 cache, CCF and DDR memory are shared among cores.

The LynxSecure hypervisor configures L2 cache using the T2080's hardware support for cache partitioning so that each core gets access to one fourth of the 16 cache ways (i.e. 4 ways). To that end the proper values are set to registers `L2PIRn`, `L2PARn`, and `L2PWRn`. In Figure 6 in the array in the L2 block, rows represent cache sets and columns represent cache ways.

Note that the CoreNet Coherence Fabric (CCF) is the main SoC interconnect and along with and the memory controller the focus for VMC case study. Interference caused by I/O activity is not included in this study.

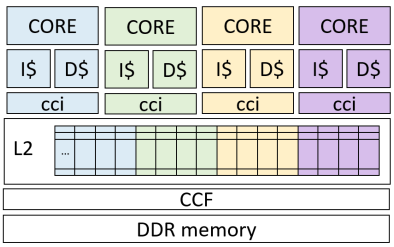


Fig. 6: Block Diagram of the main path from the cores to the DDR memory in the T2080. D$ and I$ stand for data and instruction caches, respectively; cci for core-cluster interface; and CCF for CoreNet Coherency Fabric.

C. Tool Chain Instantiation

Below we summarize how the MASTECS toolchain has been instantiated to address the VMC requirements.

① *Hardware analysis*: Hardware experts from BSC analyzed the T2080 processor [14]. Since the L2 cache is partitioned among cores it was concluded that it is not the main source of contention. The main sources are the CCF and main

memory. Next, HEMs were identified to track activity on those resources and specific micro-benchmarks were designed to stress those resources. These include several counters in the L2 cache and in the Bus Interface Unit.

② *Verification requirements*: The particular requirements addressed in the scope of the evaluation include:

- REQ1. Determine whether idle cores may generate some noise: baseline time characterization experiments require a pristine configuration. Such default configuration (CCS) must ensure that no accesses are produced from unused devices to any shared resources.
- REQ2. Determine the overhead of HEMs reading: an accurate profiling of the tasks under analysis is fundamental to avoid incurring excessive probe effect and to discard any potential outliers in the computation of WCETs. The LynxSecure hypervisor, the PMUlib, and RVS components are assessed and configured to provide accurate instrumentation.
- REQ3. Assess the increase in execution time and HEM values due to contention triggered by well-designed micro-benchmarks: for this purpose, the taskset must be instrumented and monitored both in isolation, to capture application timing baseline, and under stress scenarios where multicore timing interference arises.
- REQ4. Obtain early estimates of the impact of multicore contention on the application timing behaviour: the TCM shall allow the generation of task scheduling schemes where interference and makespan are reduced.

③ *Test Cases*: In order to capture REQ1 and REQ2 we designed test cases in which the task under analysis is run in isolation. REQ3 and REQ4 also require experiments in multicore scenarios, which specific micro-benchmarks running in different cores with or without the application under analysis.

④ Test procedures and their execution: an executable test procedure is generated for each test case allowing automated execution of the test cases and collection of the results.

⑤-⑥ Test Results: The raw results are analyzed to assess verification requirements incrementally. Results for REQ1 let us assess whether idle cores generate noise due to any background activity. REQ2 results enable calibration that the HEMs readings are accurate and a trustworthy building block for MASTECS analyses. Finally evidence for REQ3 and REQ4 provides insight on the impact of contention.

D. Results

In the following we report the results from applying MASTECS technology to fulfill verification requirements REQ1-4. All experiments build on the use of RVS tool to collect timing and relevant hardware events on the final VMC hardware and software configuration. The RVS tool gathers information at the desired granularity whilst the program under analysis executes. In the scope of this case study, we instructed the tool to automatically insert software instrumentation points for all 16 processes. Since each process consists of distinct Read, Computation and Write steps, information is collected at the granularity of each step.

1) REQ1: In order to fulfill REQ1 we prepared an experiment using the same same VM workload configuration used in the final setup and a much simpler experimental setup in which we execute a single micro-benchmark in one of the cpu cores while the remaining ones are left idle. We run several micro-benchmarks:

- RO1B. Micro-benchmark accessing and hitting one bank of the L2 with read operations.
- WO1B. Micro-benchmark accessing and hitting one bank of the L2 with write operations.
- RO4B. Micro-benchmark accessing and hitting in all banks of the L2 with read operations.
- WO4B. Micro-benchmark accessing and hitting in all banks of the L2 with write operations.
- RO. Micro-benchmark accessing and missing in the L2 and going to memory where it generates read operations.
- RW. Micro-benchmark accessing and missing in the L2 and going to memory where it generates read and write operations.

We leverage the per-core (per-thread) counters in the L2 cache. By comparing the per-core, also known as local, HEMs in the L2 with global counters we can assess whether the other cores are generating additional activity. In particular, we read the following global/per-thread hardware event pairs: L2 hits (456-global and 465-local), L2 misses (457-global and 466-local), L2 store allocates (460-global and 468-local), and L2 data accesses (462-global and 470-local). In Table II we present the results of the differences between each pair of global and per-thread HEMs. As it can be seen the local and global activity matches quite well which shows that the only activity generated in the cache is that coming where the micro-benchmark runs. The differences between global and per-thread HEMs are lower than 0.69%, which shows that noise from idle cores is negligible in the tested configuration.

TABLE II: Ratio between global and per-thread HEM pairs.

HEM	L2Hit	L2Miss	L2StAlloc	L2DataAcc
RO1B	0.20%	0.01%	0.00%	0.20%
WO1B	0.45%	0.13%	0.05%	0.56%
RO4B	0.03%	0.00%	0.00%	0.03%
WO4B	0.42%	0.29%	0.15%	0.68%
RO	0.21%	0.20%	0.06%	0.40%
RW	0.42%	0.30%	0.16%	0.69%

2) REQ2: To fulfill REQ2, we instructed RVS to collect execution information on timing, instructions, and memory accesses through local (per thread) and global (per platform) hardware counters. In particular we instrumented a dummy function on which we expect no activity and assessed RVS+PMULib instrumentation overhead against a reference scenario with minimal HEM manipulation (PMU only). At the beginning and end of the function we read 6 on-core HEMs gathered on the T2080 platform during the profiling experiments. These are per-core HEMs Processor cycles (CYC, 001), Instructions completed (INS, 002), SGB promotions (SGBP, 230), DLINK requests (DLINKR, 443) that are per core HEMs; and the L2-related HEMs L2 misses per thread

(L2Mt, 466), L2 store allocates per thread (L2STAt, 468), L2 Data accesses per thread (DL2At), and L2 Data misses per thread (DL2Mt). Those were specifically selected as they provide information on the instruction mix with emphasis on the memory operations, requests to the DL1, L2, and memory.

Table III shows the values observed. With manual instrumentation using PMULib on top of LynxSecure we observe minimal instruction overhead and no memory request. With the automation provided by the RVS infrastructure we observe very low absolute values that become negligible in relative terms as soon as the instrumented function is in the order of hundred of thousands of cycles, translating into micro-seconds.

TABLE III: Probe effect analysis. Instrumentation noise using PMULib on top of different setup layers.

HEM	CYC	INST	SGBP	DLINKR	L2Ht	L2Mt	L2STAt	DL2At	DL2Mt
ID	1	2	230	443	465	466	468	470	472
PMULib	21	5	0	0	0	0	0	0	0
RVS	1129	1562	85	85	71	2	3	57	4

3) REQ3: In order to capture REQ3 we use RapiTime to generate WCET estimates for each processes under both isolation (cycles solo) and contention scenarios with the RO and RW micro-benchmarks (RO sld and RW sld, respectively). This involves executing tasks against tailored, memory-aggressive micro-benchmarks, which are automatically stubbed by RVS. We also report the slowdown captured by RVS when the process is executed in the IFC, that is, without micro-benchmarks and with the other processes running in parallel (Parallel sld). Results are reported in Table IV.

We observe that the slowdown generated by the RO micro-benchmark is generally low. In fact, most of the times, the suffered interference is smaller than that observed in the IFC (see Observed slowdown in Table V). The contention impact of the RW micro-benchmark, instead, is always higher than taht in the IFC, effectively upper bounding it.

In general, the slowdown generated by the micro-benchmark is limited for the processes lasting longer and vice-versa. At the extremes of the spectrum we find PROC6 that lasts millions of cycles (10e6) and suffers a slowdown of only $\sim$1.10 for both core1 an core2; and PROC8 that lasts dozens of thousands of cycles (10e4) and suffers slowdowns around 5.5x when is contended against RW. As PROC8 has high density of access to memory, it suffers high slowdown against the RW micro-benchmark. However, PROC8 seems not to suffer contention from the other running processes in the IFC. Finally, all processes with a duration in the order of hundreds of thousands of cycles (10e5) display slowdowns that range from 1.40 to 3.00x against the RW.

4) REQ4: A TCM tailored to VMC hardware and software configuration has been developed and assessed in MASTECS. In particular, the tool has been integrated together with a scheduling and mapping optimization framework to provide an early assessment of different deployment configurations and identify those schedule scenarios that are not jeopardized by multicore timing interference.

TABLE IV: Core 1 and 2 results under contention scenarios. A single-core is activated with the process under analysis along with RO and RW micro-benchmarks on Core 3.

	Core 1			Core 2		
Process ID	Cycles in isolation	RO slowdown	RW slowdown	Cycles in isolation	RO slowdown	RW slowdown
PROC1(10e5)	733925	1.01	2.63	737283	1.01	2.57
PROC2(10e5)	370678	1.00	1.40	371201	1.00	1.40
PROC3(10e5)	746765	1.00	2.45	796911	1.00	2.41
PROC4(10e5)	160812	1.05	3.00	159425	1.05	2.98
PROC5(10e5)	736484	1.01	2.12	748450	1.01	2.06
PROC6(10e6)	3785988	1.00	1.10	3786251	1.00	1.09
PROC7(10e5)	740612	1.01	2.47	749216	1.00	2.44
PROC8(10e4)	57404	1.16	5.52	56599	1.16	5.51

In the following we evaluate the TCM bounds on an example deployment scenario and schedule. The profiling information on the VMC processes in isolation has been fed to the TCM and the obtained analytical bounds on multicore timing interference are assessed against maximum observed slowdown in real experiments.

TABLE V: TCM bounds against observed values.

	Core 1			Core 2		
Process ID	Time in Isolation	Observed Slowdown	TCM Bound	Time in Isolation	Observed Slowdown	TCM Bound
PROC1	733925	1.06	1.06	737283	1.07	1.06
PROC2	370678	1.01	1.17	371201	1.00	1.17
PROC3	746765	1.00	1.06	796911	1.00	1.06
PROC4	160812	1.06	1.67	159425	1.06	1.66
PROC5	736484	1.06	1.06	748450	1.07	1.06
PROC6	3785988	1.02	1.06	3786251	1.03	1.06
PROC7	740612	1.05	1.06	749216	1.06	1.06
PROC8	57404	1.03	2.62	56599	1.02	2.65

Table V reports the (maximum) observed and computed relative slowdown suffered by each VMC process because of contention. Results show the TCM results are generally upper-bounding the impact of contention, modulo a $\sim$1% tolerance threshold due to unaccounted negligible activity on the VMC Manager. While bounds are generally tight, in few cases the TCM bound seem to be overly pessimistic (see PROC4 and PROC8 in both cores). It should be noted, however, that pessimism is only apparent as it is generally difficult to hit the worst-case contention scenthose cases, namely PROC8 in both cores, correspond to short, memory intensive tasks (with $\sim$10% of instructions being memory accesses) where relative impact of memory accesses is extremely large and so is the maximum impact of contention, which is not easy to trigger with simple observations.

VII. Perspective

The MASTECS project partners have achieved success in bringing the technologies to a good commercial position and technical maturity, providing a foundation for deployment of the MASTECS methodology in support of certification of emerging aircraft systems.

The key to achieving industry-wide benefit from these tools and techniques is to ensure that the technology and commercial models enable the manufacturers and users to build on and

share best practice. For example, using a standardized process that is familiar to certification authorities reduces risk of failing to achieve certification, repeatable automation abstracts from the challenges of low-level testing making it economic, and reusable IP designed and tested/qualified for use in high-integrity systems means a faster time to market.

The automotive and aerospace case studies were highly valuable in providing feedback to the technologies and guiding the process of "productization", helping to steer the technology partners in meeting the needs of real aerospace and automotive projects - this is an example of sharing best practice to benefit the whole industry.

The significant challenges of building safety-critical systems on multicore technology will continue. As new platforms appear with new performance enhancing features (such as multi-level caches, DMA, decentralized interconnects, GPU and other accelerators) the technology required to support them will continue to develop too, building on the baselines in this paper. Expect many more developments in this area.

VIII. Conclusions

The pursuit of increased performance in critical domains is relentless, and the avionics domain is not an exception. Advanced increased-autonomy related features like Advanced Air Mobility and Single Pilot Operation, require unprecedented levels of computing performance. The use of multicore processors is the main path followed to provide the required performance. The other side of the coin is that multicores bring their own challenges including software timing analysis. In this work we have presented the MASTECS Multicore Timing Analysis methodology and tools. We also showed its application to an avionics case study. Both help assessing how MASTECS technology helps achieving CAST32-A/A(M)C20-193 requirements.

Acknowledgments

This work has been supported by the Spanish Ministry of Science and Innovation under grant PID2019-107255GBC21/AEI/10.13039/501100011033 and the European Unions Horizon 2020 Framework Programme under grant agreement No. 878752 (MASTECS).

References

[1] MASTECS: Multicore analysis service and tools for embedded critical systems. https://mastecs-project.eu/.

[2] Jaume Abella, Carles Hernández, Eduardo Quiñones, Francisco J. Cazorla, Philippa Ryan Conmy, Mikel Azkarate-askasua, Jon Pérez, Enrico Mezzetti, and Tullio Vardanega. WCET analysis methods: Pitfalls and challenges on their trustworthiness. In *10th IEEE International Symposium on Industrial Embedded Systems, SIES 2015, Siegen, Germany, June 8-10, 2015*, pages 39–48. IEEE, 2015. doi:10.1109/SIES.2015.7185039.

[3] Javier Barrera, Leonidas Kosmidis, Hamid Tabani, Enrico Mezzetti, Jaume Abella, Mikel Fernández, Guillem Bernat, and Francisco J. Cazorla. On the reliability of hardware event monitors in mpsocs for critical domains. In Chih-Cheng Hung, Tomás Cerný, Dongwan Shin, and Alessio Bechini, editors, *SAC '20: The 35th ACM/SIGAPP Symposium on Applied Computing, online event, [Brno, Czech Republic], March 30 - April 3, 2020*, pages 580–589. ACM, 2020. doi:10.1145/3341105.3373955.

[4] Frederic Boniol, Youcef Bouchebaba, Julien Brunel, Kevin Delmas, and Alfonso Mascarenas Gonzalez. PHYLOG certification methodology: a sane way to embed multi-core processors. In *10th European Congress Embedded Real Time Systems, ERTS 2020, Jan-Feb 2020*, 2020.

[5] European Union Aviation Safety Agency. Notice of Proposed Amendment 2020-09. https://www.easa.europa.eu/sites/default/files/dfu/npa_2020-09_0.pdf, 2020.

[6] Federal Aviation Administration, Certification Authorities Software Team (CAST). *CAST-32A Multi-core Processors*, 2016.

[7] Freescale semicondutor. e6500 Core Reference Manual. https://www.nxp.com/docs/en/reference-manual/E6500RM.pdf, 2014. E6500RM. Rev 0. 06/2014.

[8] Freescale semicondutor. QorIQ T2080 Reference Manual, 2016. Also supports T2081. Document Number: T2080RM. Rev. 3, 11/2016.

[9] Scott Gerhold, Mike Dunham, and Branden Sletteland. Alternative multi-core processor considerations for aviation. In *AHS International 74th Annual Forum & Technology Display, Phoenix, Arizona, USA, May 14-17, 2018*, 2018.

[10] Sylvain Girbal, Jimmy Le Rhun, and Hadi Saoud. METrICS: a Measurement Environment For Multi-Core Time Critical Systems. In *9th European Congress Embedded Real Time Systems, ERTS. Jan-Feb 2018*, 2018.

[11] Laurence H. Mutuel, Xavier Jean, Vincent Brindejonc, Anthony Roger, Thomas Megel, and E. Alepins. Assurance of multicore processors in airborne systems. DOT/FAA/TC-16/51, Federal Aviation Administration, 2017.

[12] Xavier Palomo, Mikel Fernández, Sylvain Girbal, Enrico Mezzetti, Jaume Abella, Francisco J. Cazorla, and Laurent Rioux. Tracing hardware monitors in the GR712RC multicore platform: Challenges and lessons learnt from a space case study. In *32nd Euromicro Conference on Real-Time Systems, ECRTS 2020, July 7-10, 2020, Virtual Conference*, volume 165 of *LIPIcs*, pages 15:1–15:25, 2020.

[13] Xavier Palomo, Enrico Mezzetti, Jaume Abella, Reinder J. Bril, and Francisco J. Cazorla. Accurate ilp-based contention modeling on statically scheduled multicore systems. In *25th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2019, Montreal, QC, Canada, April 16-18, 2019*, 2019.

[14] Roger Pujol, Hamid Tabani, Jaume Abella, Mohamed Hassan, and Francisco J. Cazorla. Empirical evidence for mpsocs in critical systems: The case of NXP's T2080 cache coherence. In *2021 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1162–1165, 2021. doi:10.23919/DATE51398.2021.9474078.

[15] David Radack, Harold G. Tiedeman, and Paul Parkinson. Civil certification of multi-core processing systems in commercial avionics. Technical report, Rockwell Collins, 2018.

[16] NXP Semiconductors. Chip Errata for the i.MX 6SLL. Document Number: IMX6SLLCE, 2019.

[17] Alejandro Serrano-Cases, Juan M. Reina, Jaume Abella, Enrico Mezzetti, and Francisco J. Cazorla. Leveraging hardware qos to control contention in the xilinx zynq ultrascale+ mpsoc. In Björn B. Brandenburg, editor, *33rd Euromicro Conference on Real-Time Systems, ECRTS 2021, July 5-9, 2021, Virtual Conference*, 2021.

[18] Hamid Tabani, Leonidas Kosmidis, Jaume Abella, Francisco J. Cazorla, and Guillem Bernat. Assessing the adherence of an industrial autonomous driving framework to ISO 26262 software guidelines. In *Proceedings of the 56th Annual Design Automation Conference 2019, DAC 2019, Las Vegas, NV, USA, June 02-06, 2019*, page 9. ACM, 2019. doi:10.1145/3316781.3317779.

[19] Reinhard Wilhelm. Mixed feelings about mixed criticality (invited paper). In Florian Brandner, editor, *18th International Workshop on Worst-Case Execution Time Analysis, WCET 2018, July 3, 2018, Barcelona, Spain*, volume 63 of *OASICS*, pages 1:1–1:9. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/OASIcs.WCET.2018.1.

[20] Frank Wolfe. EASA and FAA to issue further guidance on multicore certification this year. *Aviation Today*, February 2020. URL: https://www.aviationtoday.com/2020/02/28/easa-and-faa-to-issue-further-guidance-on-multicore-certification-this-year/.

[21] Xilinx. Zynq UltraScale+ MPSoC, APU - PMU Counter Values Might Be Inaccurate When Monitoring Certain Events. Document Number: AR# 68878, 2017.

Embedded Real Time SYSTEMS

30-31 MARCH 2022, TOULOUSE - FRANCE

Regular Paper

Using AI to estimate Memory Interference Impact on Avionics Software on Multicore Platform.

Florent Chenevier, Florence de Grancey, THALES, Joshua Salort, SII

Abstract: Characterization of memory interferences on multi-core platform is a complex and hot challenge in avionics. Instead of a fine and complex explicit modeling of all the contributors to this phenomenon, we propose an original methodology to perform this characterization, using a data/machine-learning approach. In a first step, we analyze the binary of avionics applications to extract Memory Access Pattern and compute statistical features about the usage of memory by the application. Secondly, we generate a representative dataset of applications using Bandit-based algorithm, which accelerates space exploration and allows limiting the training dataset size. The analysis of our first results reveals interesting trends concerning software behavior understanding and modeling ability.

Keywords: Memory Interferences, Multicore, Bandit & Good UCB algorithm, Space filling, Machine learning,

I. Context

A. Multicore platform for avionics platform, a challenge.

Multicore platforms begin to be used in the avionics domain, for several reasons. First, the promises of performances increase combined with a Size Weight and Power footprint reduction on the other hand. Besides, the need for performances is ever growing to serve new functions. Finally, the foreseen lack of single-core platform available in the future for industrial application. However, when it comes to using these platforms to implement critical avionics applications, difficult challenges remain to be addressed. Among those, we can name their high level of integration reducing both observability and implementation capability of partitioning mechanisms on internal shared resources, and the complexity of their design leading to strong difficulties to build behavior prediction models [1].

One of the main issues is the memory interferences (or contentions) phenomenon, which occurs on the SDRAM access bus shared by all the cores of the processor [2]. The concurrent memory bus solicitations by Software Applications are managed by internal arbitration mechanisms, which can induce a slow-down of all applications. This slow-down, sometimes eliminating the whole performance gain provided by additional cores, is hard to predict, as it results from the independent but concurrent executions of the Software Applications running on each core. It is even quite unpredictable when we cannot make any assumption on other hosted applications. When no assumption can be made, we can at least characterize the Application's contributions to this phenomenon, and the impact this one has on the considered applications.

B. Notion of Software Aggressiveness & Sensibility

As a software application both contributes to the memory contention phenomenon and is a "victim" impacted by it (depending on its own SDRAM bus solicitations and the platform arbitration with those of other applications), we need to quantify these contributions and impact. A common approach [19] for measuring this impact is to use the ratio of execution time T_{iso} measured when running in isolation (a single core used on the platform) and execution time T_{cont} measured "in a contention situation". The overhead is defined as = $(T_{cont} - T_{iso})/T_{iso}$

In order to be able to compare measures done between several applications, a reference measure frame is necessary. For this, we will consider two Reference Software Applications:

- A reference Sensible application (aka “Etalon”) designed to be a reference “victim” of memory contentions, without contributing to the phenomenon. Its Overhead measured when running alongside with A_i will then define *Sensibility* $S(A_i)$.
- A Reference Stress Application (aka “RSA”) designed to be a reference “generator” of memory contentions, without being affected by the phenomenon. The Overhead measured on A_i running alongside with RSA will define the *Aggressivity* $A(A_i)$.

C. Estimation of contentions footprint using machine learning

An Avionics software developer needs to establish the worst-case execution time (WCET) of his Application A_i in operational conditions. In a multicore context, estimating the contention footprint of A_i (i.e.: *Aggressivity* $A(A_i)$ *& Sensibility* $S(A_i)$) is plainly part of this activity.

To avoid a complex modeling that would use explicit hypotheses on both platform and application; we will try, as C. Courtaud *et al.* [19], to predict this contention footprint on our Avionics Platform using machine learning. This approach raises the following three challenges:

- The relevant input features has to be identified. They have to contain appropriate level of information to represent application Sensibility and Aggressivity.
- A representative dataset of the operational distribution of features has to be built.
- A model has to be built to capture the nonlinear behavior of Aggressiveness and Sensibility.

II. Previous work on Memory Contentions

The problem of memory contentions has been a study case since the availability of multicore CPU (therefore since more than a decade). Various approaches have been developed, either to analyze and/or reduce the off-chip (direct SDRAM) accesses, or to improve the performances of those accesses. Many analyses or mechanisms have been proposed regarding memory controller or cache controller ([9], [13], [14], [15]). Some approaches use a global modeling of both software task and platform's memory & cache management mechanism ([9], [10], [11], [18]). Some Linux-based solutions have been proposed implying several mechanisms such as memory bandwidth allocation (memguard) per core ([12], [16], [17]), bank page coloring for DRAM bank allocation per core ([15], [16], [17]) and bank page lockdown mechanism upon their access frequency, to avoid their cache eviction ([16], [17]). Finally, a few approaches focus on the way a software application accesses the memory and its impact on memory contention ([10], [13], [19] & [20]).

C. Courtaud *et al.* Approach ([19] & [20]) appears to be quite adaptable to our situation, as it does not rely on a fine knowledge of the platform (it's behavior is de facto "learned" during machine learning training phase) but only relies on the way the software addresses the memory (with introduction of Memory Address Pattern and associated metrics). It is also evolutionary (if new relevant metrics are discovered) and allows the analysis of existing software (directly from the binary executable file). At last, it is the most adaptable to our context.

A. Memory Access Pattern and statistical features

Courtaud *et al.* introduced in [19] the notion of Memory Access Pattern (MAP) in order to capture the use of memory by the software application. This pattern contains the sequence of all Assembly instructions accessing the memory (either to *write* or to *read* a value), established while going through a software branch. This can be captured either by monitoring the software execution on target, or by symbolic execution, using some dedicated tools such as **`angr`** [21]. The Figure 1 hereunder (from [19]) provide a clear idea of the MAP:

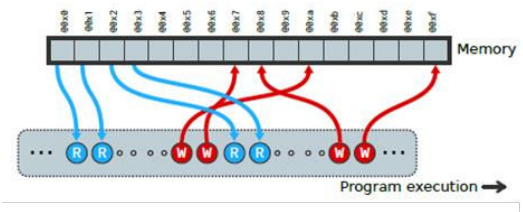


Figure 1: Representation of Memory Access Pattern (MAP)

In this MAP, we also store some context information associated to each instruction:

- The mnemonic and its type (R/W: Read or Write)
- The address in memory where the instruction read or write
- The number of "local" instructions (i.e.: instructions that only use CPU registers and do not use memory) that follow the considered R/W instruction.

From this MAP, we extract some statistical characteristics that Courtaud *et al* identified as relevant to analyze the memory interferences phenomenon:

- Memory intensity: the ratio of Read & Write instructions versus the total number of instructions (including Local instructions) This ratio is in [0;1].
- R/W ratio: the number of Read instructions versus the number of Write and Read instructions. This ratio trends to 0 when the Sw writes much more often than it reads, and to 1 on the contrary.
- Access type interleaving ratio: the number of switching from Read to Write (and reverse) in the MAP. It is a way to quantify the mixing of Read & Write operations. This ratio is in [0;1], and trends to 1 when Sw switch between Read & Write operation each time it accesses memory.
- Access pattern Entropy: an application of Shannon entropy used to quantify randomness of the memory accesses in the MAP:

$$H(P) = \mathbb{E}[I(P)] = -\sum_{j \in P} p(j) log(p(j))$$

where j is a jump in memory (i.e. : the distance between two memory addresses accessed consecutively)

III. Platform & Methodology: Our proposal

Our current work implies the following activities:

1) identify relevant features on application software regarding memory contention issues.
2) Build the pipeline to generate simulated applications A_i and measure $A(A_i)$ and $S(A_i)$.
3) Generate an optimized set of test applications covering the widest range of situations.
4) Perform real measures campaign.
5) Build a tool able to extract the pertinent features from a real Avionics application.
6) Train an estimator to make it able to predict $A(A_i)$ and $S(A_i)$ from the input features.

In this paper, our main contributions cover the first five activities listed above. Training an estimator and its evaluation is left as future work.

A. Defining Memory Access Pattern (MAP) and features for Contentions analysis.

As detailed in [19], the effects of the memory contentions on any application can be determined by analyzing the way the application sequentially accesses the memory during its execution. We will here use the notion of Memory Access Pattern (MAP) introduced in [19]. We extract the MAP using **`angr's`** Control-Flow Graph (CFG) recovery and symbolic execution capabilities. Once built, we extract from the MAP the four features detailed in § II.A *"Memory Access Pattern and statistical features"*.

Another approach would be to skip this feature extraction step, and to consider directly the MAP (or maybe the full sequence of all assembly instructions –not only those which access memory- in their order of execution) as an entry for an NLP-based estimator. In such a case, we would consider the CPU assembly as a language, the sequences of instructions as texts, and the estimation of Aggressivity & Sensibility as NLP sentiment analysis of a text. However, we discarded such a deep-learning/NLP approach until now, as it might be quite heavy: training from scratch a (Transformer-like) NLP model may need a considerable amount of patterns. Furthermore, the analysis of explicit features provides useful keys (to eventually correct or adapt a software to lower its contention footprint), better than a black-box deep learning model.

B. Building the dataset for estimator training

For the training of an estimator (step 6 in figure below), the main challenge is to build a representative training, validation & test dataset which captures both MAP & its features plus Aggressivity & Sensibility (A&S) measures. That means (step 1 and 2), building a set of simulated software applications (each one is a sequence of assembly

instructions). Then, we run each application on target in parallel with the Etalon and RSA applications to get its footprint (A&S) by measuring execution times (step 3), and we also extract the MAP and its associated features (step 4). The capitalized measures, MAPs and features constitute our machine-learning database (step 5).

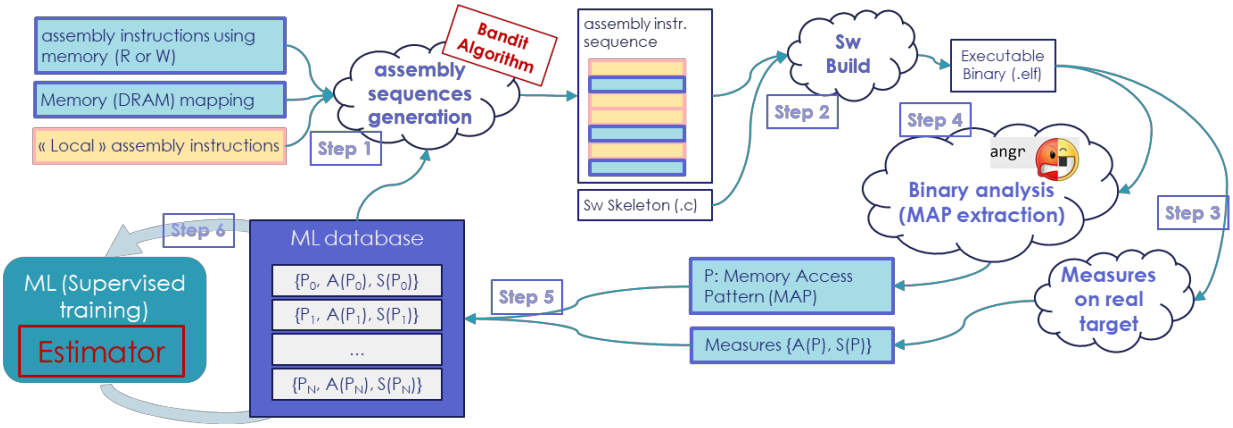


Figure 2: Principle of the dataset generation process for ML training

As we may repeat all these operations several thousands of times, we shall automatize and optimize this pipeline: minimize each step's duration and optimize the number of test applications to reach adequate precision of estimator. This optimization process is described in §IV "Optimizing the data collection".

C. Optimizing the dataset construction

A representative dataset should capture the diversity of both the MAP extracted features and of the measured outputs (Aggressivity and Sensibility). The construction of this dataset raises several challenges: the industrial necessity to limit the measures on target (and so the number of items in ML database), and the need to explore the widest range of instruction sequences to capture a diversity of application behavior.

D. Inference Process Extracting the MAPs from real software and footprint characterization

After training, the next phase is to use the trained estimator to get the Aggressiveness and Sensibility (A/S) footprint for any Avionics Application. For this, given the binary of an Application, we use **`angr's`** symbolic execution to go through the different branches of CFG and extract the corresponding MAPs (step 1 in figure below). Currently, this symbolic execution only supports linear CFG (i.e.: without branch). We will extend it to real cases in a second phase. Each MAP and its features are submitted to estimator (step 2). The collection of Aggressiveness and Sensibility provided by estimator for each MAP constitutes the A&S footprint of the Application (step 3).

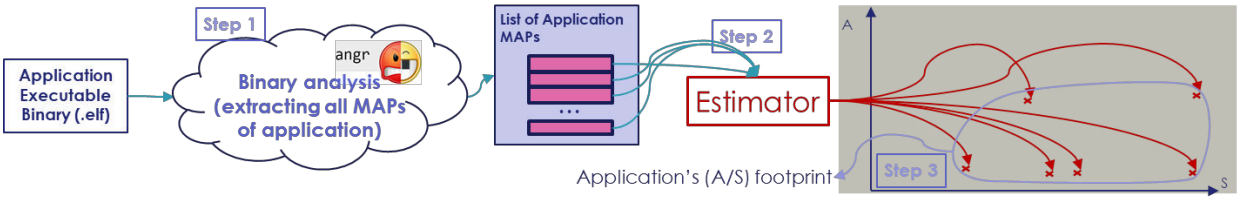


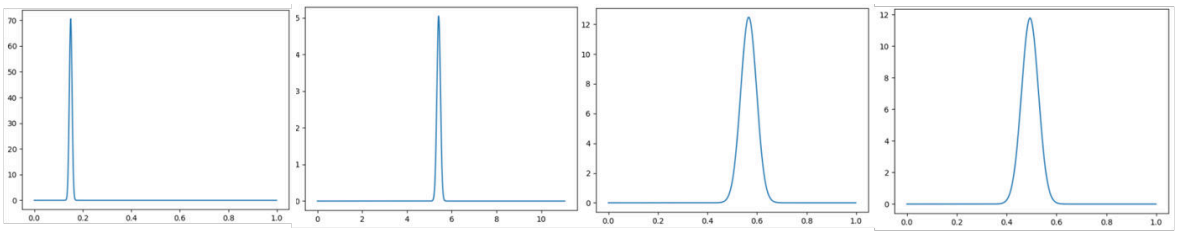
Figure 3: Principle of real software footprint estimation

IV. Optimizing the data collection

A representative dataset should capture both the diversity of the four inputs metrics and the diversity of the measured outputs (A/S). The construction of this dataset raises several challenges: a combinatorial explosion to explore all possible sequences, the feasibility of only a few experiments, and the necessity to introduce software expertise to explore the pertinent sequences.

A. The failure of random Monte Carlo generation

As a first step, to evaluate the complexity of the dataset creation task, we performed naïve data collection using random Monte Carlo exploration. Sequences are built using a random uniform selection of instructions from the CPU dictionary. First experiments (see details hereunder) show that the metrics of these sequences are quite close to those induced by the repartition of Read, Write & Local instructions in the dictionary. We can explain that by the fact that hundreds of random selections preserve the distribution of instructions in the dictionary. On 1400 sequences of 1500 instructions (randomly picked in a 200-instructions dictionary), we get the following metrics distribution:



Intensity : Mean= 0.15, Std dev.= 0.005 | Entropy : Mean= 5.42, Std dev.= 0.08 | R/W ratio : Mean= 0.57, Std dev.= 0.03 | Interleaving : Mean= 0.49, Std dev.= 0.03

Therefore, a more relevant approach should be find using space exploration methods.

B. Previous work on space exploration

The question of collecting enough representative data to model application can be compared with the problematic of space exploration / space filling designs for computer simulation experiments. Several methods exist, based on random sampling, geometrics criteria [3], maximin design [4] or latin hypercubes [22] & [23]. These methods proposed a way to sample the input space to be representative using few points. Few studies in space exploration also present methods to maximize exploration of output space [5] using bi-level objective criteria. After analysis, we move aside those algorithms, as we cannot introduce in them any kind of expertise concerning the design of input sequences.

The question of collecting enough representative data in few experiments is also close to the problematic of multi-armed bandit's algorithms. Such algorithms are designed to maximize a cumulated reward while performing a limited combination of arms. Several state of the art algorithms exist such as multi-armed UCB [6], contextual multi-armed bandits [7]. We found that these algorithms are not suitable to our use case, as we cannot define a unique reward to orient the algorithm search.

A third alternative for space exploration was found in the work of S.Bubeck *et al*, considering a Good UCB algorithm where several experts are used and a good Turing algorithm is used to select the expert which propose a better exploration [8]. This specific kind of multi-armed bandit algorithm appears as the most suitable algorithm to solve our problem compared to space filling methods, as it uses the notion of "experts" (implementing domain knowledge) to guide exploration.

C. Informed sequence Exploration: Good UCB algorithm

1. Principle

The Good UCB algorithm proposed in [8] uses a set of K Experts {I1,… Ik} to explore sequentially space Ω of elements X, in order to find specifics discrete elements A. At each time t, a played experts Ik observes an element X(k,t) which can belong or not to the subset A. The aim of the algorithm is to select at each time step t, the expert Ik which maximizes the number of elements X(k,1..t) belonging to A discovered by the expert and only him.

For that, at each time step t, the algorithm uses for each expert k: {1…K} its "missing mass" Ri, that is the number of elements of A non-discovered by the expert. In practice, this value is not measurable, so a missing mass estimator is built as follows:

$$\hat{R}_{(k,}n_{k,t-1)} = \frac{1}{n_{i,t-1}} \sum_{x \in A} 1 \left\{ \left(\sum_{s=1}^{n_{i,t-1}} 1\{X_{k,s} = x\} \right) = 1 \;\; and \left(\sum_{j=1}^{K} \sum_{s=1}^{n_{j,t-1}} 1\{X_{j,}s \; = x\} \right) = 1 \right\}$$

With $n_{i,t} = \sum_{s<t} 1\{I_k = i\}$ number of times the expert Ik is selected during the time horizon {1…t}. The right part of the bracket represents the number of times the expert Ik observed the element x during the timeline

{1...t-1}. The left part represents the number of times the element x is observed by all experts. The missing mass is incremented when the expert Ik was the only one to observe an element x.

The optimal expert is selected by choosing the expert that maximizes the missing mass. We introduce a penalty factor C to enhance exploration at the first stages of the process.

$$I_t = \arg\max \hat{R}_{(k}, n_{k,t-1)} + C.\sqrt{4\log(t)/n_{k,t-1}}$$

2. Application

To apply the good UCB algorithm to our use case, we made the following assumptions :

- An "expert" is a builder of instruction sequences, which can be a random builder (eg. picking randomly instructions in the instruction dictionary) or a rules-oriented builder using a specific algorithm to select instructions.
- The desired discrete set of elements A is design such way: The p-dimension metrics space is discretized, an element A is a "p-boxes" in this space. A new element of discrete set A will be considered as discovered if the measured metrics fall into one p-box of discrete space.

We also made the following choices:

- We decided to design three instance of Good UCB algorithm, where the missing mass is computing 1) with respect to the aggressivity metrics only (ie. The missing mass is incremented when a new value of aggressivity is discovered), 2) with respect the sensibility metrics only, 3) and the combination of aggressivity and sensibility only (ie. The missing mass is incremented when a new 2D box of a aggressivity/sensibility table is discovered).
- We decided to modify the missing mass estimator by introducing a new hyper parameter ObsMax that is the maximum "allowed" number of (re)discoveries of an element by all experts. Indeed, we considered that the space explored by each expert may not be disjointed.

$$R_k, n_{k,t-1} = \frac{1}{n_{i,t-1}} \sum_{x \in A} 1\left\{ \left(\sum_{s=1}^{n_{i,t-1}} 1\{X_{k,s} = x\} \right) = 1 \;\; and \left(\sum_{j=1}^{K} \sum_{s=1}^{n_{j,t-1}} 1\{X_{j,}s \; = x\} \right) = ObsMax \right\}$$

3. Experts Design

The definition of used "experts" mainly shapes the performance of Good-UCB algorithm. An expert is characterized by an accessible exploration space, which, ideally, should contain some elements of the discrete set of elements A. In our case, ideally, the union of the accessible exploration space of each expert shall recover the complete metric space. So a specific attention is provided on expert design in order to favor exploration.

As experts are instruction-sequences builders, several choices are possible. First, as demonstrated in § IV.A, we do not select random instruction builders, as they build sequences with MAP metrics (RW ratio, interleaving, intensity & entropy) quite close from instruction dictionary mean values. We favor the design of "metrics oriented experts" which aims to build instruction sequences with a target value for one or several features of the MAP (RW ratio, interleaving, intensity & entropy). We design experts with uniform repartition of targeted value to allow complete 4-metrics space exploration.

In addition to this target metric, we perform some checks during the sequence building to verify that the built sequence succeeds in compiling and does not present typical software errors such as divisions by zeros or attempts to access unauthorized memory areas.

V. Preliminary Exploration Results

We performed several generation campaigns of 200 generated sequences, each sequence composed of 1500 instructions picked in a 206 instructions dictionary, and accessing a 12 Mbytes Memory area. We always used the same 27 experts, in order to evaluate the algorithm behavior depending on the hyper-parameter values.

A. Exploration tuning

We first explore the impact of an exploration based on Missing Mass computed on Sensibility or computed on Aggressivity (Figure 4 below):

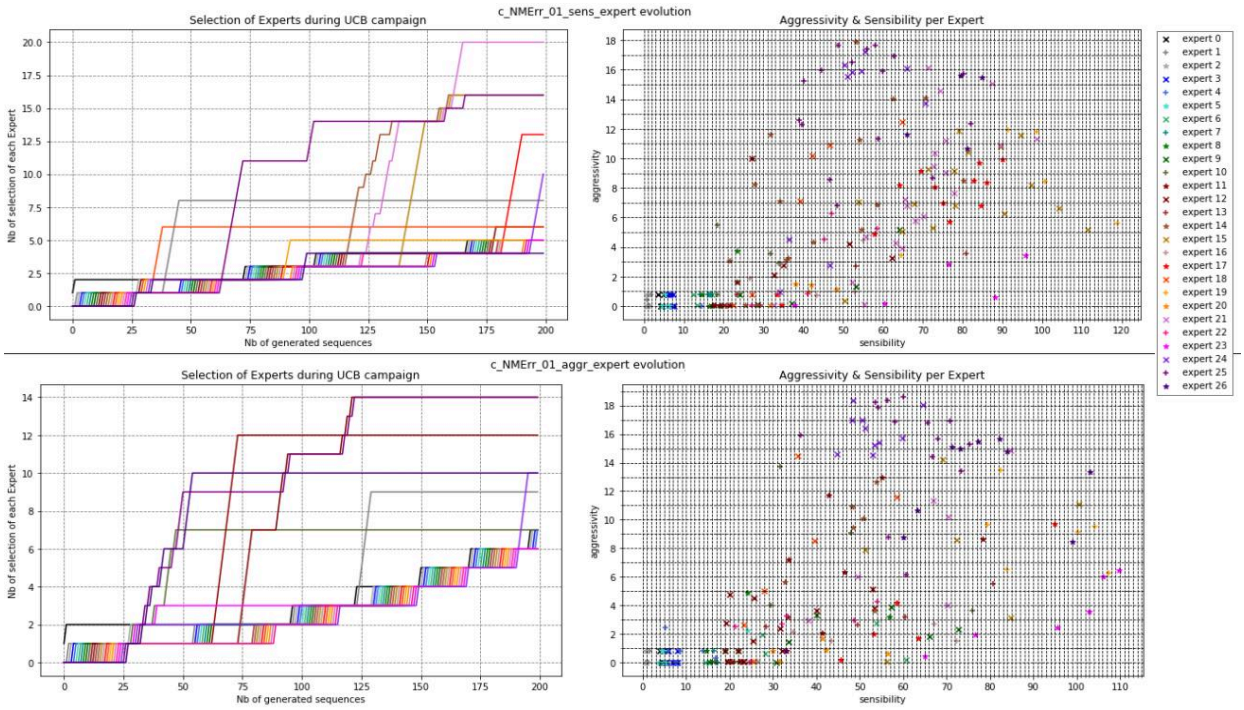


Figure 4: Exploration with Missing mass computed on Sensibility vs Agressivity

The Expert selection trends are different in both cases: for instance, the most selected experts in “sensibility exploration” are not the same as for the “aggressivity exploration” and the distribution of generated sequences in the A & S plan are different. The exploration orientation depending on the measure used for missing mass has an impact on the exploration.

In a second step, we explore the influence of hyper-parameter C: Figure 5 hereunder provide the evolution of missing mass and the number of selection for each expert during campaign (200 sequences generated). A low value of C (C=0.1, left graph) will lead to select more often some Experts, while a higher value of C (C=6.0, right graph) will lead to a more distributed use of experts.

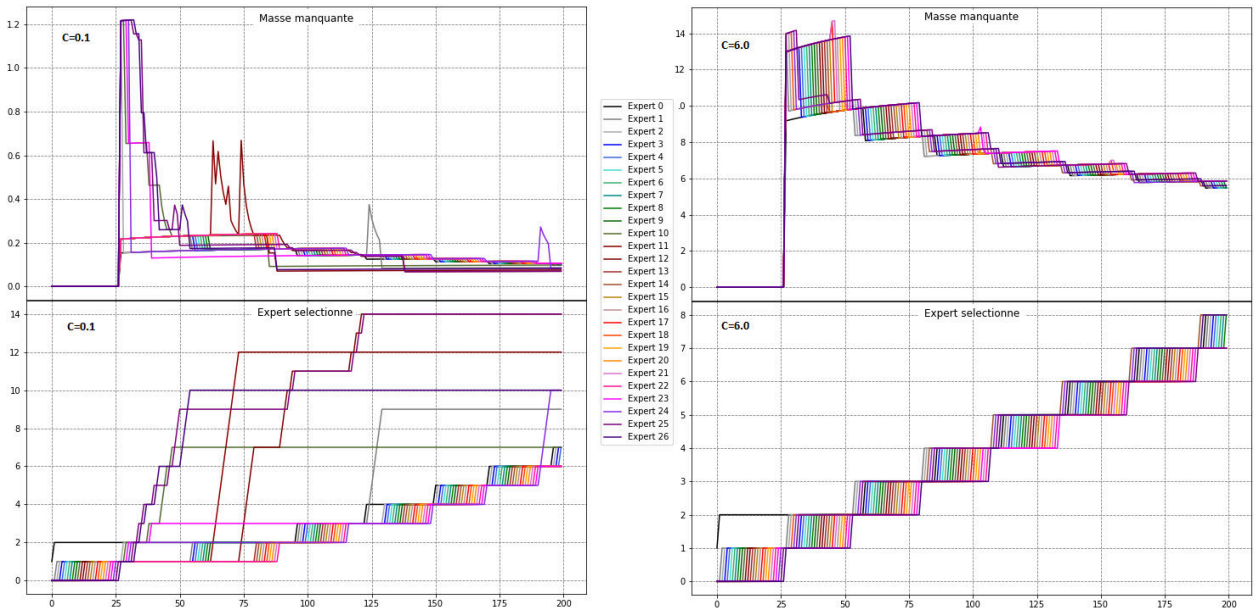


Figure 5: Impact of C Hyper-parameter on Missing Mass & Expert selection

When exploring the impact of ObsMax Hyper-parameter, the Figure 6 hereunder, comparing expert selection for Obsmax value 1, 3 & 5 shows that allowing more observations (e.g.: 5) of the same discrete element of the search space will lead to use a wider subset of experts, while allowing less (e.g.: 1) will lead to overuse one specific expert.

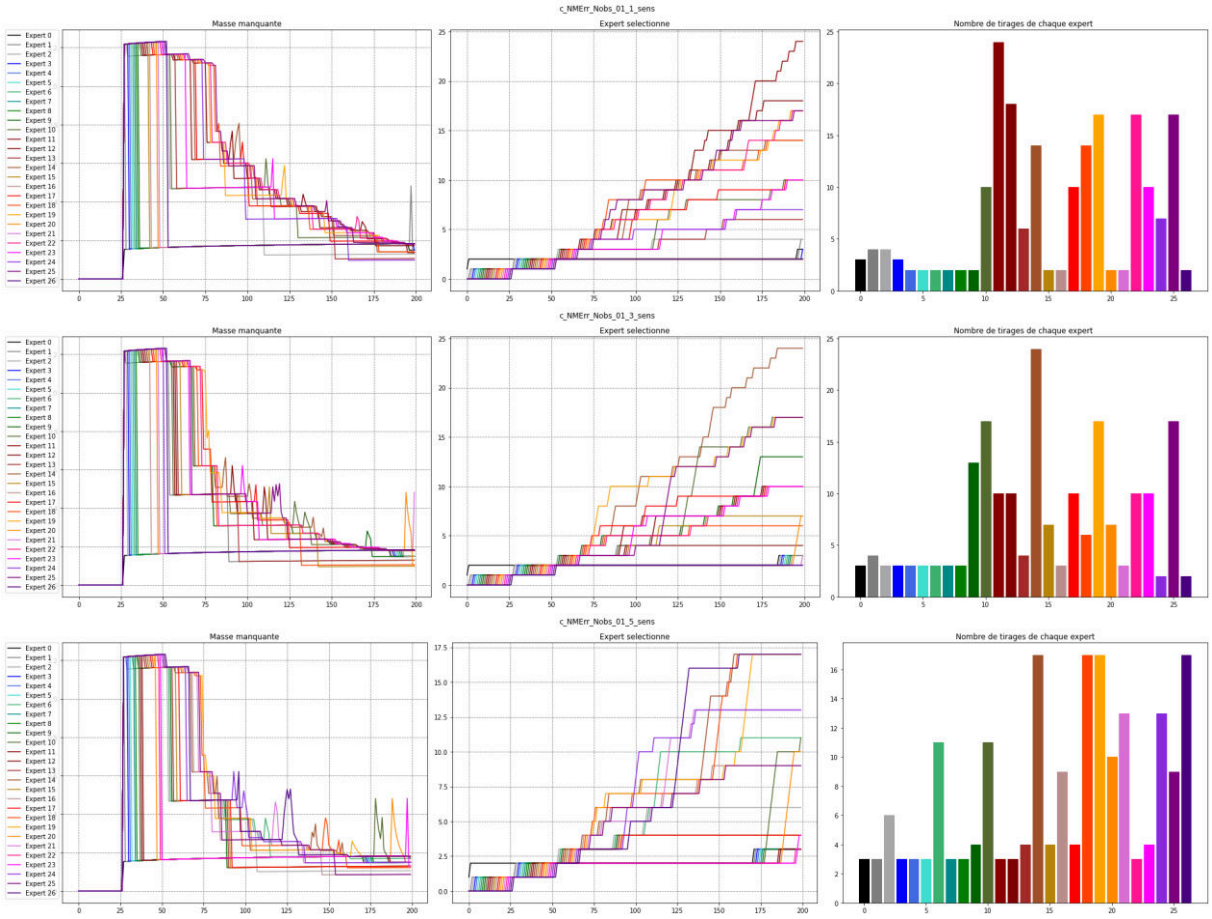


Figure 6: Comparison of influence of ObsMax (1, 3, 5), for C=0.1

B. Analysis & Modeling

The first results, provided in Figure 7 & 8 hereunder, shows that current experts have a quite low "exploration capability" (actually, they generate sequences with metrics quite close to the targets they were given). This limit the global exploration capability of UCB algorithm. Further generation campaigns will be launch in the future with less constrained targets provided to experts.

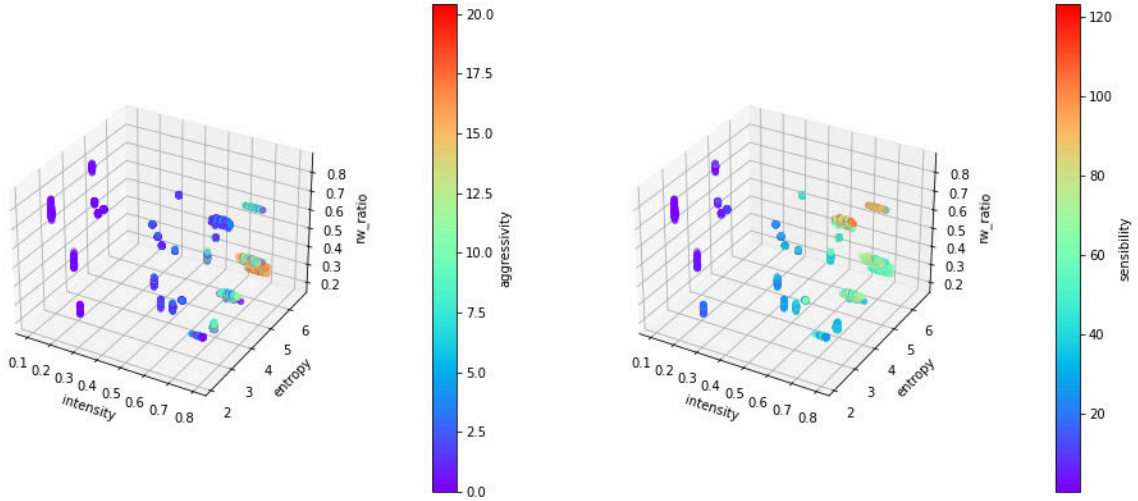


Figure 7: Aggressivity & Sensibility versus intensity, entropy & R/W ratio

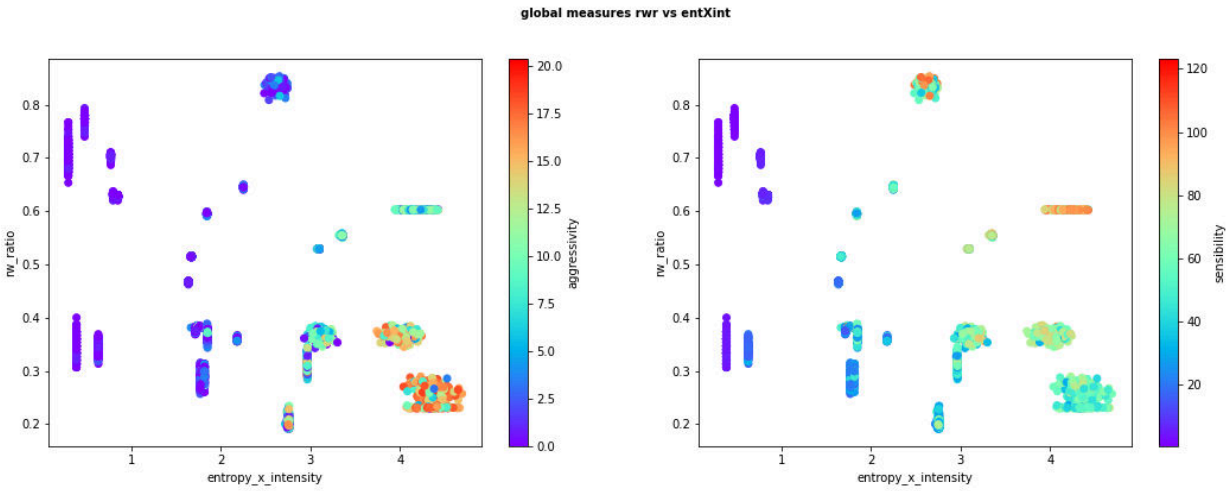


*Figure 8 : projection of Aggressivity & Sensibility on a (rw_ratio, entropy*intensity) plane*

However, the first measurement campaigns already present interesting trends on measure interpretation, shown in Figure 7 & 8 above and Figure 9 below. We first observed that data are split into two "families": one with Aggressivity value below 1% and Sensibility below 30%, and a second one with higher values. The first family contains mainly sequences with few read or write instructions (low intensity), or with high intensity but with very low entropy. The second family contains sequences with higher intensity of memory access or with more entropy in these accesses. A second trend reveals that we reach high A&S values for high intensity *and* entropy values. A third trend shows the influence of R/W ratio: the sequences with a "Write profile" (i.e: R/W ratio close to 0) are more aggressive, when those with a "Read profile" (R/W ratio close to 1) are more sensible. This confirms analysis performed by other software engineering teams.

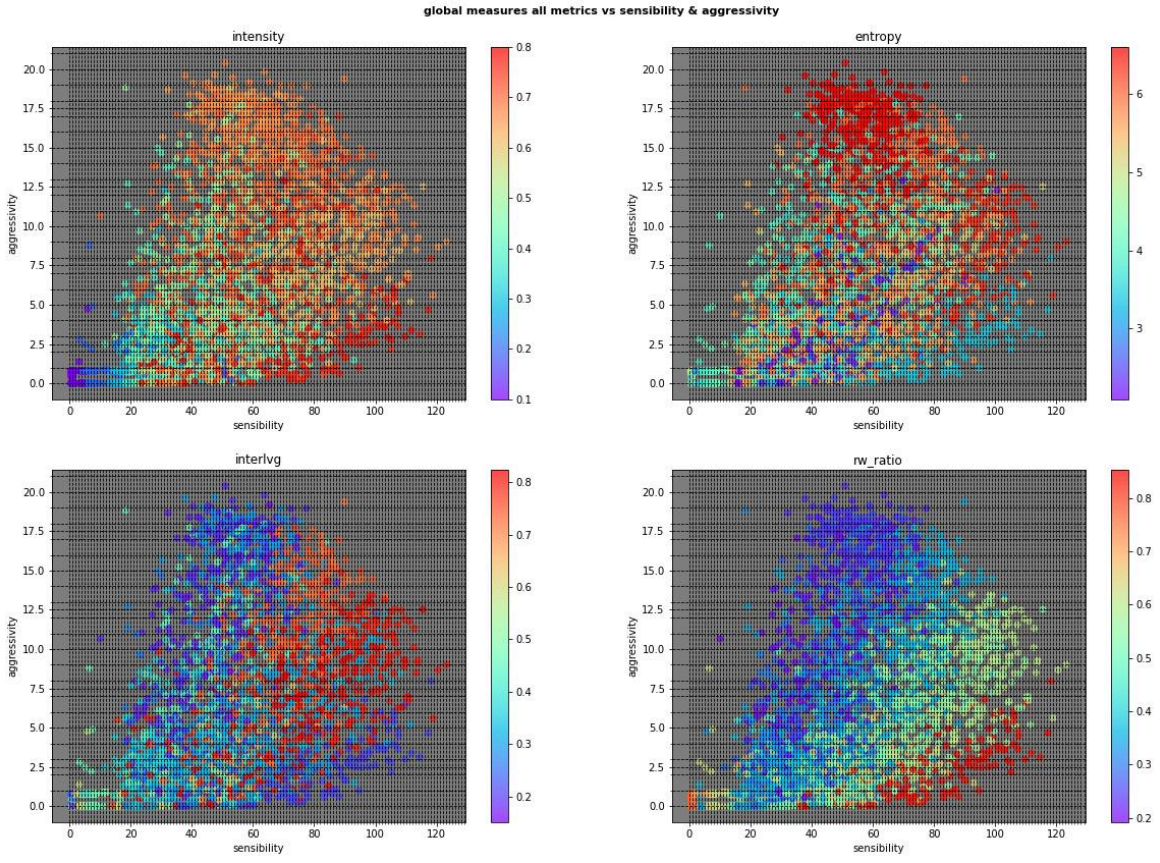


Figure 9: Repartition of each metrics on the Aggressivity/Sensibility plane

We decided to confirm our intuitions by building regression trees for prediction of Aggressivity and Sensibility. The built regression tree effectively shows that intensity and entropy are the more discriminant features. A first

split is performed using a threshold value of 0.5 for intensity. Sequences with intensity below this threshold are mainly “low sensitive and low aggressive” sequences. For high intensity values, a second split concerns entropy with a threshold of 3 or 5, depending from aggressive or sensitive point of view. R/W ratio and interleaving occurs only at third or fourth stages of splits. These regression trees offer simple rules to estimate the behavior trends of sequences.

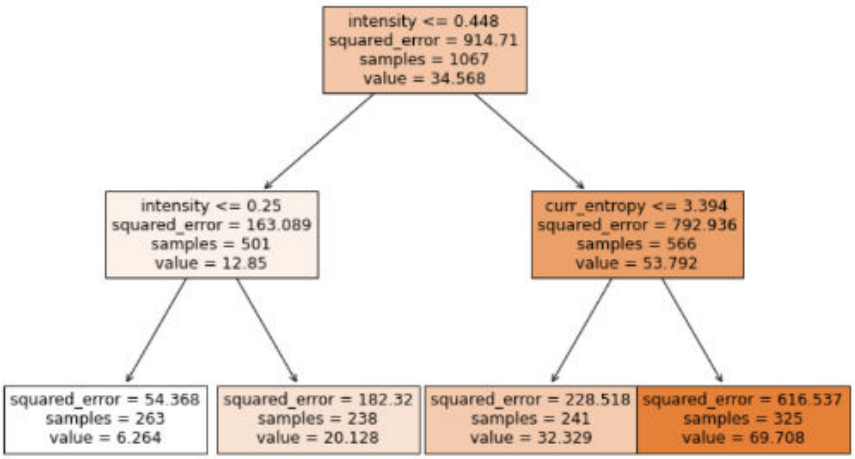


Figure 10: Example of regression tree for sensibility metrics build with around one thousand measurements.

VI. Conclusion and Next steps

We elaborate a framework to easily generate Test Software applications and relevant measurements dedicated to the analysis of memory contention phenomenon. Based on concepts elaborated in previous works (MAP and associated features), we provide a way to efficiently generate a database of software sequences with known characteristics and their associated measured contention footprint (Aggressivity & Sensibility).

This database will allow testing different type of machine-learning estimators to get a better model of software behavior regarding contention phenomenon.

Furthermore, the analysis of our first results reveals interesting trends concerning software behavior understanding, to be completed with complementary analyses.

In the next steps, we will:

- Enhance the generation of sequences by refining the values of UCB hyper-parameters
- Train several type of estimators and compare their performances on generated Test sequences.
- Test the estimators on real avionics software applications

VII. Bibliography

[1] R. Wilhelm, C. Ferdinand, C. Cullmann, D. Grund, J. Reineke, and B. Triquet, ‘Designing predictable multi-core architectures for avionics and automotive systems’, in *Workshop on Reconciling Performance with Predictability (RePP)*, 2009, vol. 10, pp. 2–3. Accessed: Oct. 19, 2016.

[2] H. Kim, D. de Niz, B. Andersson, M. Klein, O. Mutlu, and R. Rajkumar, ‘Bounding and reducing memory interference in COTS-based multi-core systems’, *Real-Time Syst.*, vol. 52, no. 3, pp. 356–395, May 2016

[3] Welch, WJ, ACED: Algorithms for construction of experimental designs, The American Statistician, 1985

[4] Van Dam, E. R., Husslage, B., Den Hertog, D., & Melissen, H. (2007). Maximin Latin hypercube designs in two dimensions. Operations Research, 55(1), 158-169.

[5] Chen, Q. & Paulavičius, R. & Garcia-Munoz, S. & Adjiman, C. (2018). An Optimization Framework to Combine Operable Space Maximization with Design of Experiments. AIChE Journal. 64. 10.1002/aic.16214.

[6] Wei Chen, Yajun Wang , Yang Yuan Combinatorial Multi-Armed Bandit: General Framework, Results and Applications

[7]Langford, John; Zhang, Tong (2008), "The Epoch-Greedy Algorithm for Contextual Multi-armed Bandits", Advances in Neural Information Processing Systems 20, Curran Associates, Inc., pp. 817–824

[8] S. Bubeck, D. Ernst, A. Garivier, Optimal Discovery with Probabilistic Expert Advice: Finite Time Analysis and Macroscopic Optimality, 2013, Journal of Machine Learning Research 14 (2013) 601-623

[9] S. P. Muralidhara, L. Subramanian, O. Mutlu, M. Kandemir and T. Moscibroda, "Reducing memory interference in multicore systems via application-aware memory channel partitioning," 2011 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), 2011, pp. 374-385.

[10] B. M. Tudor, Y. M. Teo and S. See, "Understanding Off-Chip Memory Contention of Parallel Programs in Multicore Systems," 2011 International Conference on Parallel Processing, 2011, pp. 602-611, doi: 10.1109/ICPP.2011.59.

[11] Liya Liu and O. Hasan and S. Tahar, "Formal Analysis of Memory Contention in a Multiprocessor System", SBMF, 2013

[12] H. Yun, G. Yao, R. Pellizzoni, M. Caccamo and L. Sha, "MemGuard: Memory bandwidth reservation system for efficient performance isolation in multi-core platforms," 2013 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS), 2013, pp. 55-64, doi: 10.1109/RTAS.2013.6531079.

[13] J. Marandola, S. Louise, L. Cudennec, J. Acquaviva and D. A. Bader, "Enhancing Cache Coherent Architectures with access patterns for embedded manycore systems," 2012 International Symposium on System on Chip (SoC), 2012, pp. 1-7, doi: 10.1109/ISSoC.2012.6376369.

[14] Dakshina Dasari and Vincent Nelis and B. Akesson, "A framework for memory contention analysis in multi-core platforms", Real-Time Systems, 2015, V.52, pp 272-322

[15] Lei Liu, Z. Cui, Mingjie Xing, Y. Bao, M. Chen and Chengyong Wu, "A software memory partition approach for eliminating bank-level interference in multicore systems," 2012 21st International Conference on Parallel Architectures and Compilation Techniques (PACT), 2012, pp. 367-375.

[16] Muhammad Ali Awan and P. Souto and B. Akesson and K. Bletsas and E. Tovar, "Uneven memory regulation for scheduling IMA applications on multi-core platforms", Real-Time Systems, 2018, V.55, pp 248-292

[17] H. Yun, G. Yao, R. Pellizzoni, M. Caccamo and L. Sha, "MemGuard: Memory bandwidth reservation system for efficient performance isolation in multi-core platforms," 2013 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS), 2013, pp. 55-64, doi: 10.1109/RTAS.2013.6531079.

[18] D. Casini, A. Biondi, G. Nelissen and G. Buttazzo, "A Holistic Memory Contention Analysis for Parallel Real-Time Tasks under Partitioned Scheduling," 2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), 2020, pp. 239-252, doi: 10.1109/RTAS48715.2020.000-3.

[19] C. Courtaud, J. Sopena, G. Muller and D. Gracia Pérez, "Improving Prediction Accuracy of Memory Interferences for Multicore Platforms" 2019 IEEE Real-Time Systems Symposium (RTSS), 2019, pp. 246-259, doi: 10.1109/RTSS46320.2019.00031.

[20] Cédric Courtaud. Caractérisation de la sensibilité aux interférences mémoire dans les systèmes temps réels embarqués sur des plateformes multi-coeurs. Systèmes embarqués. Sorbonne Université, UPMC University of Paris 6, 2020. Français. tel-03022017

[21] Shoshitaishvili Yan, Wang Ruoyu, Salls Christopher, Stephens Nick, Polino Mario, Dutcher Audrey, Grosen John, Feng Siji, Hauser Christophe, Kruegel Christopher,Vigna Giovanni, “SoK: (State of) The Art of War: Offensive Techniques in Binary Analysis”, 2016, IEEE Symposium on Security and Privacy

[22] Tang, B. (1993). Orthogonal array-based Latin hypercubes. Journal of the American statistical association, 88(424), 1392-1397.

[23] Leary, Stephen & Bhaskar, Atul & Keane, Andy. (2003). Optimal orthogonal-array-based Latin hypercubes. Journal of Applied Statistics. 30. 585-598. 10.1080/0266476032000053691.

Modelling and analyzing multi-core COTS processors

Frederic Boniol, Julien Brunel, Kevin Delmas, Claire Pagetti
ONERA, Toulouse, France

Victor Jegu
Airbus, Toulouse, France

***Abstract*—To embed multi-core COTS processors in an avionic product, the platform must be thoroughly analyzed from two perspectives: the worst case real-time behaviours and the safety impact of internal failures. Both activities are very complex and error-prone for large size systems. Moreover, the frameworks for both perspectives (real-time and safety) are completely decoupled, leading to independent and possibly incoherent analyses.**

Our purpose is to unify both worlds and help designers in their certification process. To this end, we have formalized and unified as much as possible the different perspectives of multi-core analysis. We have also proposed a simple description language for the platform, which contains the minimal concepts needed by both perspectives, as well as an automatic translation to the two analysis frameworks.

I. Introduction

Aeronautical safety critical systems are subject to *certification*, meaning that a *certification authority* assesses the compliance of the product with a set of adequate standards.

a) ***Certification of multi-core COTS – CAST 32A****:* The CAST32-A position paper [1] provides a set of guidance for software planning and verification on multi-core-based systems. Indeed, multi-core chips, i.e., chips integrating several cores interconnected by a shared bus, face important challenges for their integration in safety critical environment. There are two main types of analysis to perform: worst case real-time analysis and safety analysis.

Real-time and interference As a matter of fact, it is very difficult to ensure *time predictability* [2], [3] for multi-core COTS, one of the key elements requested by certification. *Time predictability* is the capability to compute a safe and tight upper bound of the number of cycles required to execute a piece of software in the worst case. For multi-core COTS, the problems come from the intensive resource sharing, the lack of documentation and the complex internal behaviour (e.g. cache coherence) to increase the average performance. For mastering the worst case behaviour, the CAST32-A promotes the computation of *interferences* – situation where several applications execute in parallel and encounter a serious timing delay compared to when executing in isolation – and *interference channels* – shared resource of the platform.

Internal failures and safety effect The classical approach was to consider the processor as a whole such that any failure leads to the complete failure of the system. Such an approach is considered as a bit naive and pessimistic for multi-core. Indeed, if a core fails, the rest of the platform can still work correctly and the global system can still be safe. Thus, making a sharper analysis decreases the pessimism. On the other hand, modern processor architectures integrate many components and intelligence such that they can be seen as systems themselves. Identifying the failure modes, their effects and their failure rates is rather challenging. Some works, such as [4], propose to emulate a component failure and observe the reaction of the platform. Others, such as [5], [6], propose to deduce abstract failure modes from the functional services in pragmatic reasoning approach. Some, e.g. [7], try to quantify the failure rate with real platform experiments. For mastering the failure propagation, the CAST32-A promotes the identification of internal failures and their containment within the equipment (integrating the multi-core) not to polute the avionics.

b) ***Objectives and contribution.****:* Practically, the applicant must argue that they have identified the interference and the safety effect for their platform and their specific use. The analyses are applied on the platform which includes the hypervisor or RTOS (real-time operating system) if any. By specific use, the CAST32-A speaks of *configuration settings*, i.e. the way the processor is used. This includes the description of which components are used and how (with which parameters).

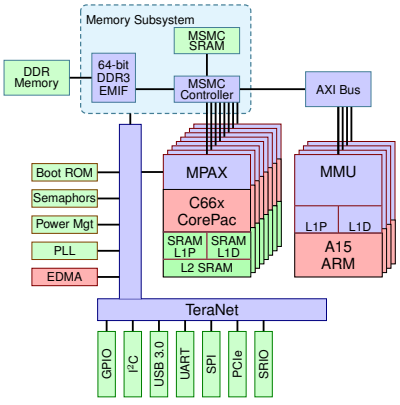


Fig. 1. KEYSTONE platform

Let us consider as an example the KEYSTONE TCI6630K2L [8] from Texas Instruments, which is depicted in Figure 1. The configuration settings include which cores are running and with which frequency; the peripherals that are used, how the memory is configured and so on.

Once the configuration settings have been clearly described, the applicant must then identify the interferences, i.e., compute

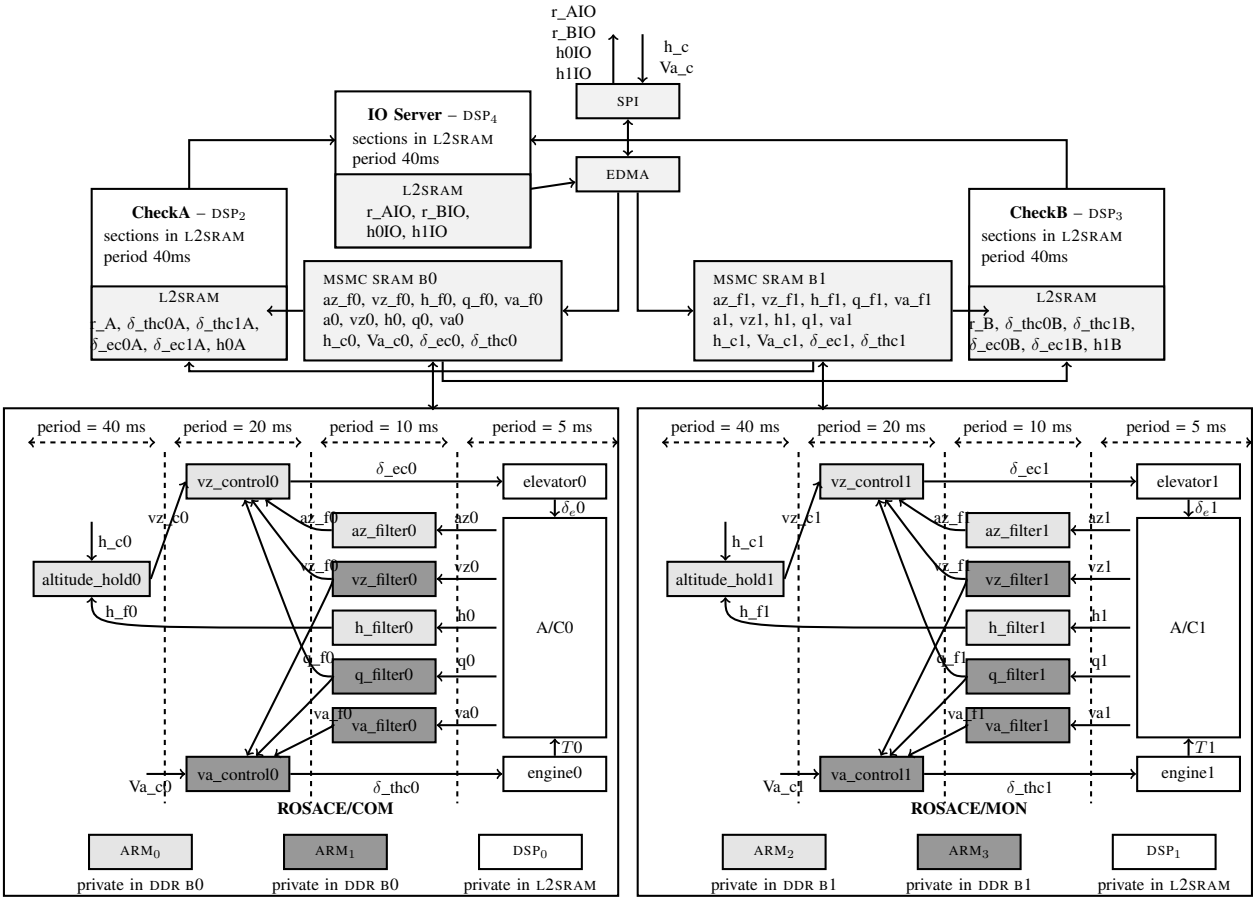


Fig. 2. Adapted RROSACE application

how software could access the different resources in parallel. The basic solution is to compute all *transactions* – accesses from a core to a shared resource – and enumerate all combinations with a solver [6].

In parallel, the applicant must also identify the failure modes of each internal component of the platform and determine how these failure modes would impact the transactions. For instance a non acceptable transaction (outside the configuration settings) can occur in the presence of some failure. The basic solution is to make a dysfunctional model and analyze the safety effect [9] with a safety framework, such as [10].

Both perspectives deal with the notion of transaction and how the platform is solicited. However, they are analyzed via independent tools and techniques. Our objective is to unify them as much as possible to factorize the modelling work and reduce the divergence between the perspectives. To do so, we have formalized the notion of transaction, interference and erroneous transaction. We have defined a multi-core-based system description framework to 1) describe thoroughly the platform with the common concepts needed by the analyses; 2) translate the description to each perspective and in a format that is compliant with the analysis tools. This framework was developed within the project PHYLOG[1].

*c) **Outline of the paper.**:* To help illustrate the concept, we have defined a complete use case based on the KEYSTONE in Section II. Section III provides the formalisation of the multi-core transactions and the common description language named PML. Section IV presents the interference analysis and how such an analysis is possible from PML. In a similar way, Section V presents the safety perspective. We then detail the related works in Section VI before concluding.

II. USE CASE

To help present the contribution, we will rely on a real use case that consists in executing a simplified longitudinal flight control system (see Figure 2) on the KEYSTONE.

a) **KEYSTONE.***:* The platform, shown in Figure 1, runs in *bare-metal* (*i.e.* without any RTOS and hypervisor) and is composed of: 1) an eight C66 DSP pack, in which each core comes with dedicated L1 and L2 caches, and a memory extension and protection unit (MPAX); 2) a four ARM pack, in which each core comes with dedicated L1 caches, and a memory management unit (MMU); 3) a central memory

[1]https://w3.onera.fr/phylog/

system that gives access to the platform's SRAM (MSMC SRAM), and an external DDR. Each of the these two memory systems is composed of 8 Banks, which are denoted B_x in the sequel. The memory access management is performed by the Multicore Shared Memory Controller (MSMC); 4) a set of IO peripherals (e.g. GPIO, UART), and utility peripherals (e.g. Boot, Semaphores); 5) a memory transfer peripheral (EDMA); 6) an ultra speed bus (TERANET) connecting the peripherals, the memory systems, and the cores.

*b) **Applications**:* We consider a COM/MON longitudinal flight controller which is an adaptation of RROSACE (for redundant ROSACE) [11], [12]. The purpose is to execute two parallel ROSACE– an open source longitudinal flight controller – and to perform regular verification that both copies, named COM/MON for COMmand and MONitoring, agree on the computed orders. To do so, the orders are usually compared in the MON duplicate. Our purpose is slightly different from [11], which goal was to offer a safe COM/MON strategy. Instead, we want to implement a representative use case that will stress several hardware components of the multi-core. Moreover, we have embedded the aircraft models to increase the size of the footprint and to be close to the real behaviour.

The overall use case is described in Figure 2. To allow the communication with the cockpit to receive pilots orders (required altitude *h_c* and required *Va_c*), we implemented a communication with the SPI (Serial Port Interface). We use the same medium to display the results. This results in implementing 5 functions:

1) ROSACE COM which has been allocated on several cores. The environment is on DSP_0 which is configured with a L2SRAM such that the execution is contained locally, except for the data that is exchanged with the controller. Those global variables are stored in MSMC SRAM B_0. The controller has been split in two parts: one executing on ARM_0 and the second on ARM_1. All the private sections are stored in DDR B_0. The ARM caches are activated. The controller receives orders (*h_c0*, *Va_c0*) from the pilot and computes the orders δ*_ec0* and δ*_thc0*.

2) ROSACE MON works in the same way except that the aircraft model is on DSP_1, the controller is on ARM_2 and ARM_3, the private sections are on DDR B_1 and the global variables are in MSMC SRAM B_1. The controller receives the same orders as COM and they are stored as *h_c1* and *Va_c1*; and computes the actions δ*_ec1* and δ*_thc1*.

3) CheckA executes on DSP_2. Its L2 is configured as L2SRAM and contains all the sections and data. CheckA reads several data in the MSMC SRAM B0 (δ*_ec0*, δ*_thc0*, *h0*, *Va_c0*) and MSMC SRAM B1 (δ*_ec1*, δ*_thc1*, *h1*, *Va_c1*). It then checks whether the orders computed by COM and MON are close, e.g. by verifying whether $|\delta\_ec0 - \delta\_ec1|$ and $|\delta\_thc0 - \delta\_thc1|$ are small, and CheckA stores the result in the Boolean variable *r_A* (which is true if COM and MON agree, and false otherwise).

4) CheckB works as CheckA and computes *r_B*.

5) IO server executes on DSP_4. Its L2 is configured as L2SRAM and contains all the sections and data. The IO server is in charge of communicating with the outside of the multi-core via the SPI. More precisely, it configures the EDMA to receive the pilot orders from the SPI and copy them on MSMC SRAM B0 and MSMC SRAM B1. It also periodically reads the outputs of CheckA and CheckB directly in their L2SRAM and copy them locally in its L2SRAM. It configures the EDMA to send those to the SPI.

III. MODELLING MULTI-CORE ARCHITECTURE: PML

To prepare the certification documentation required by the CAST32-A, the applicants must analyze the platform from the two perspectives, real-time and safety. Even if they differ in terms of framework, they both rely on an accurate representation of the platform itself. Such a representation is derived from hardware documents and expert knowledge.

A. Components

The software are hosted by hardware components. When a software requests some resource, it initiates a *transaction* within the platform. This transaction consists of a *path* of physically connected components. According to their role in a transaction, components are classified as follows, taking inspiration from the initiator-target model introduced in [13], [14], [15], [6].

Definition 1 (Initiator-target model): A multi-core is composed of three types of components:

Initiator: a component which initiates a transaction (e.g. ARM, DSP and EDMA);

Target: an end-component which is targeted by initiators (e.g. MSMC SRAM and SPI);

Transporter: any intermediate component between initiators and targets (e.g. TERANET and AXI BUS).

Example 1: The components of the KEYSTONE illustrated in Figure 1 are colored according to their type, the color code being: red for Initiators, blue for Transporters and green for Targets.

The mapping of the (software) applications to the platform components defines the configuration of the platform and induces which components / transactions are active.

Example 2: The set of RROSACE software components and their allocation are defined in Figure 2. In particular, DSP 5-7 are turned off, several peripherals are disabled, several DDR and MSMC SRAM Banks are unused. Figure 3 shows the final configuration.

B. Transactions and services

The interaction between software and platform is abstracted away through the notion of service. Indeed, when a software initiates a transaction within the platform, e.g. to retrieve data, each component along this transaction plays its role by providing a service. Components offer many services such as execute or address translation. But in the context of this article, we focus on the minimal services that are needed for the interference and safety perspectives, *i.e.*, LOAD and STORE.

Example 3: Let us consider again the KEYSTONE with the configuration described in II. Application *az_filter_0* may

need to read data stored in the MSMC SRAM B_0 memory. This is expressed as a LOAD service call and consists in a transaction propagated through internal components until the DDR is reached. Besides, each of these components provides a LOAD service. Figure 3 shows an extract of the service-oriented KEYSTONE architecture.

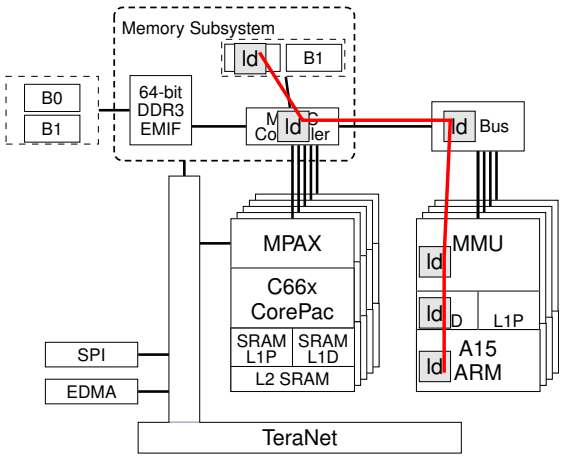


Fig. 3. Example LOAD transaction

Definition 2 (Platform service): A platform offers a set of services that can be called upon by the Initiators and that generate transactions. We have identified the following services:

- LOAD: retrieval of some data from a given target by an initiator.
- STORE: writing of some data to a given target by an initiator.

Definition 3 (Component service): We consider that all components offer both services (LOAD and STORE). In particular for a component c, we will use the notation c_l (resp. c_s) to represent the LOAD (resp. STORE) service offered by the component c. For a component service s, $cp(s)$ is the component that provides s.

Example 4: The component service ARM_0_l is provided by the component $cp(\text{ARM}_0\_l) = \text{ARM}_0$.

Definition 4 (Transaction): A transaction is initiated by an initiator and follows a pre-defined *path* to connect the initiator to the final target. We denote a LOAD (resp. STORE) transaction tr that represents the initiator i reaching the target t as $i \rightarrow^l t$ (resp. $i \rightarrow^s t$).

Remark 1: We make the distinction between LOAD and STORE transactions for two main reasons: firstly, for some platforms (not for the KEYSTONE), LOAD and STORE may use different paths; secondly they induce different effects on both interference and safety analyses. Indeed, LOAD and STORE transactions induce completely different temporal effects. Moreover, the propagation of a failure along a LOAD transaction goes from the target to the initiator whereas it goes the other way around in the case of a STORE transaction.

Remark 2: Note that the path has no specific direction. Indeed, a transaction actually represents a sequence of interactions, which can go a one way or the other. For instance, a LOAD consists in sending a request from a core to the DDR and then the data is sent back from the DDR to the core.

Remark 3: The KEYSTONE verifies the so-called *unique path property*. Indeed, any transaction $tr = i \rightarrow^X t$ with $X \in \{l, s\}$ always follows a unique path (both for resquest and data). For other types of platform, making an X from i to t could lead to several paths. In such a situation, the notation $i \rightarrow^X t$ should be enriched. In order to take this into account, we would need to introduce the concept of *single transaction*, satisfying the unique path property and we would define a transaction as a set of single transactions.

Definition 5 (Copy): DMAs (such as the EDMAS) make copies from one memory area to another one. Thus we denote the copy transaction as $DMA \rightarrow^{copy} [Mem_1, Mem_2]$. We can see such a transaction as the pipeline of the two transactions $DMA \rightarrow^l Mem_1$ and $DMA \rightarrow^s Mem_2$.

Definition 6 (Path): A transaction $tr = i \rightarrow^X t$ with $X \in \{l, s\}$ follows a path of components denoted by $cp\_path(tr)$, which is a chain of *Transporters*, except the last component, which is a *Target*. Let $p = c_1 \leftrightarrow \ldots \leftrightarrow c_k$ be a component path, then two successive components c_j and c_{j+1} in p are physically connected.

Each component along $cp\_path(tr)$ contributes to the transaction tr by providing the service X. The resulting path of services is denoted by $path(tr)$. Thus if $cp\_path(i \rightarrow^X t) = c_1 \leftrightarrow \ldots \leftrightarrow c_k$, then $path(i \rightarrow^X t) = c_1\_X \leftrightarrow \ldots \leftrightarrow c_k\_X$.

Example 5 (Path): For instance, the transaction $tr_1 = \text{ARM}_0 \rightarrow^l$ MSMC SRAM B_0 shown on the Figure 3 follows: $cp\_path(tr_1) = \text{ARM}_0 \leftrightarrow \text{ARM}_0\text{\_L1D} \leftrightarrow \text{MMU}_0 \leftrightarrow$ AXI $\leftrightarrow$ MSMC CTRL $\leftrightarrow$ MSMC SRAM B_0 and $path(tr_1) = \text{ARM}_0\text{\_l} \leftrightarrow \text{ARM}_0\text{\_L1D\_l} \leftrightarrow \text{MMU}_0\text{\_l} \leftrightarrow$ AXI _l $\leftrightarrow$ MSMC CTRL_l $\leftrightarrow$ MSMC SRAM B_0_l.

C. PML *metamodel*

The purpose of PML (for Multi-Core Meta-Model) is to describe the common description of a multi-core needed for both perspectives. Figure 4 provides a graphical representation of PML.

A platform is composed of several physical connected components, each with a *type* (Initiator, Transporter, Target). We represent the link between an initiator and the transactions it initiates through the association *issued by*. Each transaction tr relies on the path of services $path(tr)$ (association *along*) targeting a specific service among them (association *targets*). Each service (which can be of type LOAD or STORE) is provided by a component (association *provides*). Finally, instead of representing the allocation of software to hardware components, we consider an abstraction of it, simply representing the transactions that are made possible by this allocation (association from software to transaction).

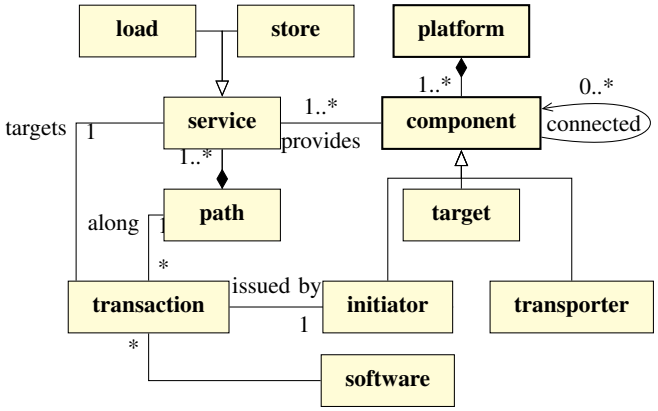


Fig. 4. Overall PML overview

D. Tooling support

The construction of an PML model is supported through a Scala API provided in the PML analyser[2]. The detailed codes of the experiments presented in this paper are provided as examples of the API usage.

The API offers a programmatic way to instantiate the physical components and applications of a platform. The description of the transactions used by the applications can be cumbersome and error-prone. Therefore, thanks to the API, one can simply specify software/hardware and data/hardware allocations from which the transactions can be automatically derived.

Validation strategy: The API also contains a set of graphical exporters that can extract specific views of the model. Such exports can be very useful during the design and the validation of the PML model. Among the possible extracts, there are:

- the physical/service connection graph of the platform;
- the physical/service connection graph restricted to the connections used by at least one transaction;
- the transactions used by a given application.

The API also provides the automatic generation of the interference model and the safety model, as detailed in the next sections.

IV. Interference analysis

One of the analyses required by the CAST32-A is the identification of all interferences and interference channels. An interference happens when two or more transactions occur simultaneously and when they use either a common service or services offered by the same component.

A. Interference calculus overview

Let us first explain what is exactly the interference calculus. The idea is to enumerate all the simultaneous transactions that can lead to a timing alteration on the application execution.

Definition 7 (Simultaneous transactions): $tr_1||\dots||tr_j$ denotes the situation where the transactions tr_i with $i \in 1..j$ can occur simultaneously. This is only possible when the initiators are distinct: $\forall k,l \in 1..j, k \neq l \Rightarrow Initiator(tr_l) \neq Initiator(tr_k)$.

Example 6 (Simultaneous transactions): For instance, let us consider $tr_1 =$ ARM$_0$ $\rightarrow^l$ MSMC SRAM B$_0$ and $tr_2 =$ ARM$_2$ $\rightarrow^s$ MSMC SRAM B$_1$. tr_1 is a LOAD transaction and tr_2 is a STORE one. *Initiator*(tr_1) =ARM$_0$ and *Initiator* (tr_2) = ARM$_2$. Then we can have $tr_1||tr_2$, *i.e.* these two transactions can be initiated simultaneously.

PML captures the minimal concepts that are needed by all analyses. When focusing on a given analysis, the corresponding view may have to be enriched. The default semantics of PML assumes that 1) two services belonging to two different components do not share any resource and thus do not interfere (*i.e.* they can simultaneously serve different transactions); 2) two services offered by the same component cannot execute in parallel since each of them needs all the resources of the component. However, some processors contain components powerful enough to provide several services at the same time. For instance, crossbars allow for parallel communications. To take this knowledge into account, we have to extend the PML model with a new relation specifying the services that can run in parallel without producing an interference.

Definition 8 (Parallel services): Let us introduce the relation *parallel*, which represents the pairs of services s_1 and s_2 that can run in parallel without conflicting on any resource.

$$(s_1, s_2) \in \textit{parallel} \iff s_1 \textit{ and } s_2 \textit{ do not interfere}$$

This input information must be given by the designer. Such a knowledge could come from a deep analysis of the processor datasheet or from precise benchmarks exploring the behaviour of each component of the plateform.

Example 7 (Parallel services): The documentation of the KEYSTONE processor states that the ultra speed bus (i.e., the TERANET component) enables LOAD and STORE transactions in parallel without any interference. For instance, if two DSPs access the DDR Banks simultaneously, one with a LOAD transaction and the other with a STORE, then they should not interfere on the TERANET. This is encoded as

$$\textit{parallel} = \{(\text{TERANET\_}l, \text{TERANET\_}s)\}$$

Note that for the KEYSTONE, the *parallel* relation does not include any other pair of services. This means that all the components, except the TERANET, are only able to provide one service at a time. For instance, the AXI bus cannot be simultaneously crossed by a LOAD transaction and a STORE one.

Definition 9 (Interference channel): An *interference channel* is a component c, more precisely a Transporter or a Target, such that there exist two simultaneous transactions *conflicting* on this component. By conflict, we mean that this will generate an interference and thus a timing effect.

Formally, we say that c is an interference channel iff there are two *distinct* transactions tr_1 and tr_2 such that:

[2]available at https://w3.onera.fr/phylog/

(a) either $\exists s \in path(tr_1) \cap path(tr_2)$ such that $cp(s) = c$, *i.e.* the two transactions use the same service s of component c,

(b) or $\exists s_1 \in path(tr_1)$ and $\exists s_2 \in path(tr_2)$ such that $cp(s_1) = cp(s_2) = c$ and such that $(s_1, s_2) \notin parallel$.

If one of the two conditions above holds, we say that tr_1 and tr_2 interfere and they conflict on c.

Example 8 (Interference channel): Considering $tr_1||tr_2$ of Example 6. As tr_1 is a LOAD and tr_2 is a STORE, they do not use any common service: $path(tr_1) \cap path(tr_2) = \emptyset$. However, they cross two common components: $cp_path(tr_1) \cap cp_path(tr_2) = \{$AXI, MSMC CTRL$\}$. tr_1 uses the LOAD service of AXI, *i.e.* AXI_$l \in path(tr_1)$, while AXI_$s \in path(tr_2)$.

According to Example 7, (AXI_l,AXI_s)$\notin parallel$. Thus, condition (b) of Definition 9 holds. tr_1 and tr_2 conflict on the AXI and similarly on MSMC CTRL.

Definition 10 (Interference): An *interference itf* is a situation where several transactions occur simultaneously and conflict on some interference channel(s), *i.e. itf*$= tr_1||\dots||tr_n$ such that $\forall i, j \in 1..n$, if $i \neq j$ then

1) either tr_i and tr_j interfere,
2) or there exists a subset $\{tr'_1, \dots, tr'_k\} \subseteq \{tr_1, \dots, tr_n\}$ such that tr_i and tr'_1 interfere and $\forall l < k, tr'_l$ and tr'_{l+1} interfere and tr'_k interferes with tr_j.

We moreover denote by *trans(itf)*=$\{tr_1, \dots, tr_n\}$ the set of transactions of *itf*. And we say that $tr_1||\dots||tr_n$ is an *n-ary intererence*, or simply an *n-ary itf*.[3]

The conditions above mean that *trans(itf)* must form a connected graph (where the edges are the pairs of transactions that interfere). In other words, for each pair (tr_1, tr_2) in *trans(itf)* either tr_1 and tr_2 interfere, or there is a "path" of interfering transactions in *trans(itf)* from tr_1 to tr_2.

Example 9 (Interference): Let us consider again the transactions tr_1 and tr_2 of Example 6. $tr_1||tr_2$ is a *2-ary itf* that conflicts on AXI and MSMC CTRL.

Example 10 (Interference): Let us now consider the transactions $tr_3 =$ DSP$_3 \rightarrow^l$ MSMC SRAM B$_1$ and $tr_4 =$EDMA$\rightarrow^l$ SPI. As shown in Figure 5, tr_1, tr_2 interfere; tr_3 interferes with tr_1 and tr_2 on MSMC CTRL; tr_4, tr_3 interfere on TERANET. Thus $\{tr_1, tr_2, tr_3, tr_4\}$ is a connected graph, even if tr_4 does not interfere directly with tr_1 and tr_2. $tr_1||tr_2||tr_3||tr_4$ is then a 4-ary itf.

Definition 11 (Interference channel associated with an itf): For a given *interference itf*$= tr_1||\dots||tr_n$, an interference channel associated with *itf* is an interference channel that appears between at least two transactions of *itf*. The set *chan(itf)* of the interference channels associated with *itf* is defined as follows:

$$chan(itf) = \{ \quad c \in Transporter \cup Target | \exists tr_j \neq tr_k \in trans(itf) \text{ such that } tr_j \text{ and } tr_k \text{ interfere on } c\}$$

Example 11 (Interference channel associated with an itf): As a last example, let us consider again the 4-ary itf depicted

[3]Note that we use *itf* either to denote a specific interference or to abbreviate the term "interference".

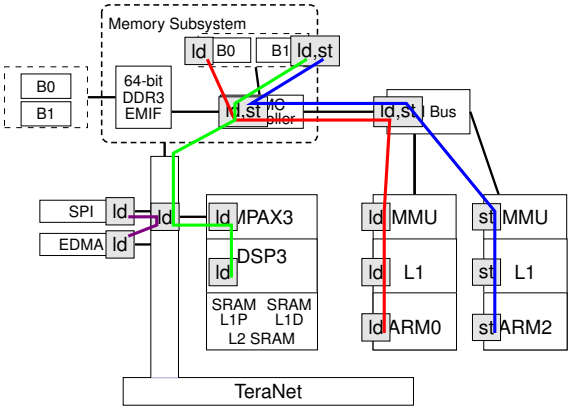


Fig. 5. Example of 4-ary itf: $tr_1||tr_2||tr_3||tr_4$

Figure 5: *itf* $= tr_1||tr_2||tr_3||tr_4$. The associated set of interference channels is *chan(itf)* $= \{$AXI,MSMC CTRL, MSMC SRAM B$_1$,TERANET$\}$.

Complementary to definition 10 which defines *n-ary itf*, i.e., a simultaneous transaction $tr_1||\dots||tr_j$ which interfere by forming a connected graph, we can now define the set of interference-free simultaneous transactions :

Definition 12 (Interference-free): A simultaneous transaction $s = tr_1||\dots||tr_j$ is an *n-ary interference-free* iff $\forall i, j \in 1..n$, if $i \neq j$ then tr_i and tr_j do not interfere.

As shown below (paragraph Experiments), identifying all the simultaneous transactions that are supposed to be interference-free is a way to explicit the hypotheses hidden in the model.

B. What is generated from PML?

Method 1 (Interference identification): The CAST32-A asks for the identification of all interferences. This means that we need first to determine the transactions and their path. Then, with $n = 2^{|\text{initiators}|}$ (|initiators| being the number of initiator components), we enumerate all n-*ary* itf, *i.e.* all the combination of n simultaneous transactions that may interfere. And finaly we enumarate all the associated interference channels. The way to compute the interference is then left to a solver. In our case, we use IDP [16] or MONOSAT [17]. The constraints are hard coded and are independent from the platform model. Thus, from PML, we need to generate automatically the transactions and their paths.

Example 12 (Generation of simple transactions): For the use case, the model is composed of 54 transactions, each of them being defined by its path of components. Transactions tr_i $i = 1 \dots 4$ in Figure 5 are examples of these 54 possible transactions. This model is then enriched with the parallel relation as defined in Example 7.

C. Experiments

The interference analyser generates for each $n \in 2..N$ (where N is the number of initiators)

1) the set IF^n of n-ary interference-free simultaneous transactions;
2) the set I^n of n-ary *itf*.

Validation strategy: IF^n is interesting to check the correctness of the model. Indeed, as any combination $tr_1||\ \ldots||tr_n \in IF^n$ is supposed not to generate any interference, the idea is to measure the behaviour of $tr_1\ \ldots tr_n$ in isolation and to check that it does not change when running them in parallel. The sets I^n provide the answer of the CAST32-A certification objective.

Example 13 (Interference calculus): For the use case, the interference analysis provides the following results:

Type	Size				
	2	3	4	5	6
itf	364	2 580	12 384	40 704	92 768
free	928	7 298	30 067	66 796	75 072

Type	Size				Total
	7	8	9	10	
itf	144 896	148 480	90 112	24 576	556 864
free	33 024	0	0	0	213 185

The next step after generating IF^n and I^n for all $n \in 2..N$ is to associate a benchmark $m_1||\ldots||m_n$ with each $itf = t_1||\ldots||t_n$ of I^n and to run it in order to quantify the interference. In the same way, a similar benchmark should be associated with each element of IF^n in order to check that there is no interference. The total number of *itf* and of interference-free simultaneous transactions seems to be too large to be tractable. However, let us note that the transactions tr_i we are considering are micro-transactions performing only one type of action (*load* or *store*). The corresponding micro-benchmarks m_i are then very small codes only repeating a same instruction a finit number of times. The typical execution time of such micro-benchmarks is about $10ms$. Running about 600 000 benchmarks would take less than 2 hours.

V. SAFETY ANALYSIS

The purpose of the safety analysis is to identify the effect of physical failures on the applications, that is the behaviour of the transactions in the presence of failures. The way to analyze the CAST32-A safety objectives to multi-core was presented at [18]. We simply sketched here the main ideas and detail the link with PML.

A. PHYLOG *safety analysis reminder*

The formal concepts of the the safety analysis are presented in details in [18]. Let us here detail and illustrate how those steps are applied in the context of multi-core-based systems. Note that to the best of our knowledge, there is no contribution to this question in the literature.

*a) **Identification of failure modes**:* As a preliminary approach, we consider two kinds of failure modes:

Erroneous The component does not properly process a transaction, which results in its corruption (data or address).

Lost The component does not process incoming transactions, which results in a deny of service.

The safety effect of these failure modes are described in the table below. Note that, the user could easily define more failure modes.

Type	FM	Comments
STORE	*err*	erroneous value or wrong destination is stored
	lost	no value is stored
LOAD	*err*	erroneous data is loaded
	lost	no data is loaded

*b) **Failure propagation model**:* The dysfunctional model describes the interconnection of physical components (e.g. cores, MMU) and the transactions that ensure the system's functions. The idea is to abstract the transactions to determine 1) whether a transaction was correctly handled, 2) what are the effects of a failure on a given transaction or 3) what are the effects of an erroneous transaction on the other transactions.

In the safety view, the path associated with a transaction tr is directed. This is due to the propagation of failures, which follows a direction along a transaction (either from the initiator to the target in the case of a STORE transaction, or the other way around, in the case of a LOAD transaction). This direction is ensured by the use of input and output ports in each component. The idea is to represent the data propagation and how their loss or corruption would affect the applications. To do that, each component is modeled as a mode automaton [19] and the whole system is the connection of all components. The behaviour of a the platform is partially illustrated in Figure 6.

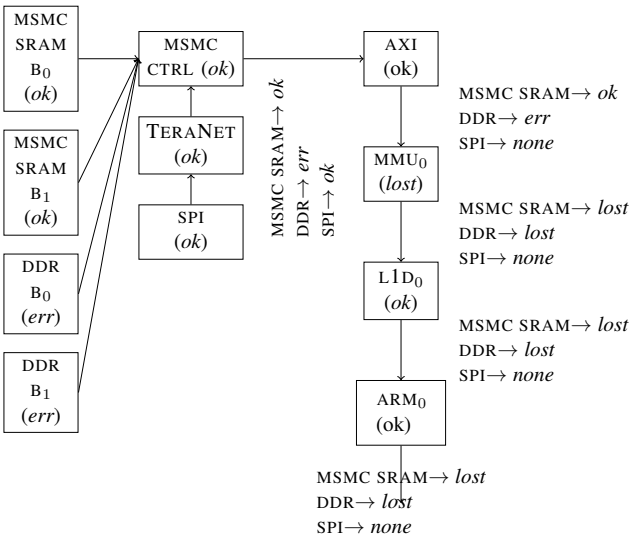


Fig. 6. LOAD transaction with some failures

Let us consider the transaction tr_1 of Figure 3, translated in a failure propagation view as shown in Figure 6. We observe the local effect of each component failure mode. In

this scenario, all components are *ok* except the MMU_0 which is in the *lost* mode and the DDR banks that are *err*.

The MSMC CTRL has to merge the inputs of several components (MSMC SRAM banks, DDR banks, TERANET). This merging necessitates a more complex behaviour in the mode automaton that we will not detail here. In effect, it would take, as input for Target t, the output value provided by t.

The outputs of MSMC CTRL are the inputs of AXI. The AXI BUS only considers the transactions of the ARMs.

The outputs of AXI are the inputs of MMU_0. Here, because of the internal failure of the component, values are not transmitted anymore to the ARM, taking then a *lost* value.

STORE works similarly with more failure propagation. First, as for the interference view, the model can be enriched with a special type of Transporters, called Virtualizers (such as the MMU or MPAX), which define the authorization for accessing transactions. To better model the effect of Virtualizer failures, we consider that an erroneous Virtualizer may access any target and pollutes these targets with *err* values. Second, a Target that receives *err* values with a STORE transaction is considered itself as corrupted from now on. This behaviour is taken into account within the hard-coded library of ALTARICA components. The interested reader can find more information on the formal encoding in [18].

c) ***Safety objective****:* In the context of our case study, the safety objective is to ensure that RROSACE controls correctly the aircraft in the following sense: if the orders sent to the actuators are *err*, this situation must be detected. The detection is done by CheckA and CheckB, thus the violation of the safety objective arises when ROSACE MON, ROSACE COM, CheckA and CheckB are *err* at the same time. Thus, the situation that we want to avoid, which is called a *failure condition* is defined by *CheckA*=err *and CheckB*=err *and COM*=err *and MON*=err.

We consider that *CheckA*=err if its L2SRAM is *err* or DSP_2 is *err*. Same for CheckB. *COM*=err if at least one of its LOAD is *err*. Same for MON.

B. What is generated from PML?

As for the interference view, some parts are hard coded in ALTARICA. We have developed a generic library for each type of components (Initiator, Target, Transporter). What is specific is the number of Target and which Target are visible in the set T_c (coming directly from the transactions). The global system, *i.e.* the interconnection of components, is also completely generated. Finally, the user must add its failure conditions. Those are then translated as an observer, which is the typical way to observe the output of the system. An external analyzer (CECILIA WORKSHOP [10]) is then used to perform the safety assessment out of the comprehensive ALTARICA model extracted from PML.

Validation strategy: Since the exported ALTARICA model can be imported in the CECILIA WORKSHOP, the user can benefit from:

Step-wise simulator that unfolds a failure scenario graphically to observe the error propagation encoded by the model. Such a simulator can be used to validate some well-chosen test-cases through expert consultation or fault injection if the application and the platform (or a detailed model) are available.

Sequence generator that computes all the failure scenarios up to a given size. This tool can be used to build a validation test base by computing all the scenarios containing one single failure where the feared event is observed (positive test) or where the feared event is not observed (negative test). One can then conduct a fault injection campaign based on these tests to validate both the vulnerability (with positive test) and tolerance (negative tests) of the architecture.

C. Experiments

Part of the results when assessing the failure condition is given in the Table I. Instead of directly looking at the safety objective, we just show two sub-failure conditions.

failure conditions	cut set
COM.*err*	$\{\text{ARM}_x.err\ (x \in \{0,1\})$, $\text{MMU}_x.err\ (x \in \{0-4\})$, MSMC SRAM $\text{B}_0.err$, AXI.*err*, DDR $\text{B}_0.err$, $\text{DSP}_0.err$, $\text{MPAX}_X.err\ (x \in \{0-4\})$, EDMA.*err* $\}$
CheckA.*err*	$\{\text{DSP}_2.err$, $\text{L2SRAM}_2.err$, EDMA.*err*, $\text{MPAX}_X.err\ (x \in \{0-4\})\ \}$

TABLE I
SAFETY ASSESSMENT RESULTS FOR ROSACE

We observe that ROSACE COM.*err* is reached when one of the transaction has failed ($\text{ARM}_{0,1} \rightarrow^{l}$ DDR B_0, MSMC SRAM B_0) or one of the virtualizer has failed (MMU or MPAX) or the EDMA has failed (as it could write in any target, thus in particular DDR B_0 and MSMC SRAM B_0). CheckA has failed if CheckA execution resources have failed ($\text{DSP}_2.err$ and $\text{L2SRAM}_2.err$) or one of the virtualizers has failed (MPAX) or the EDMA has failed because the latter can write erroneous values in the L2SRAM_2.

Table II shows the timing of the framework applied on the RROSACE use case. Even if applied on a unique use case, with a quite realistic size, it shows promising scalability.

Task	Interference		ALTARICA
	IDP	MonoSat	
Model generation	4s	$< 1s$	11s
Analysis	4h45	$127s$	14s

TABLE II
TIMING PERFORMANCE OF THE FRAMEWORK

VI. RELATED WORK

Abstracting components by the services they offered is not new to analyze platform behaviour. For instance, CPA (Compositional Performance Analysis) has been widely used to compute worst case traversal time on embedded networks (such as AFDX or TSN) and the methodology considers that abstract resources provide network services [20] such as Qbv.

More recent work [21] computes memory access timing on multi-core processor with PYCPA. The multi-core itself is abstracted with its event arrival curves as a sequence of LOAD or STORE transactions (not the combination of both) and only the interaction with the memory is considered.

a) Support to design: Some works follow a different approach, without modelling the platform. This is the case for instance, of the timing analyses proposed in [22], [23], which take into account possible faults of hardware components.

For automotive system engineering, the authors of [24] have modelled the concepts that are important to the whole design process, from system engineering to software engineering. The obtained metamodel is used to ease the interaction between the different tools that are used during the development of automotive software. Although the hardware architectures are multi-core, interaction between cores is not the focus of this study.

AMALTHEA[4] is another framework proposed in the automotive domain for multi-core software development. This framework, based on the Eclipse technology, provides a metamodel for multi-core software and hardware modelling. The objective of this model-driven approach is to centralize all the information necessary for the complete development process. From this central model, it is possible to call different tools for partitioning, mapping, code generation, and trace analysis. AMALTHEA focuses on the development process and aims at reducing data exchanges between the tools involved in the process. Our approach is different since we only focus on the data necessary to the certification issues, allowing us to use a simpler metamodel for multi-core processors.

In the avionics and space fields, the DREAMS project [25] followed a similar approach by generalizing it to embedded distributed platforms, including multi-core processors. The aim of this project was to define a framework and a methodology for designing mixed-criticality systems (MCS). This framework is based on a metamodel of MCS capturing all the relevant design, implementation and configuration artefacts, and on a model-driven engineering process supported by tools focusing on design-space exploration, real-time scheduling, and reconfiguration synthesis. Thus the DREAMS framework focuses on the left branch of the V-cycle, and ranges from design model to derivation of platform configuration. Our contribution is different since we focus on the right branch of the V-cycle, and particularly on the certification activities in this branch. Our objective is not to support the design process, but to ease the generation of certification artefacts compliant with the MCP-CRI standard.

In the avionics fields, some work tried to adapt the MCP-CRI standard to COST multi-core architectures [26]. To ease design and certification stages, they propose to group the MCP-CRI objectives into three high level principles: (1) determining the final configuration, (2) managing interference channels, and (3) verifying the use of shared resources. However, they showed that predicting interference on a COTS multi-core architecture is a very challenging task because of the amount of possible scenarios. A way to overcome this difficulty is to use a formal model of the architecture and a formal analysis method to explore the set of interference channels. Such is the aim of our contribution.

Some other works have studied how to support the design of multi-core systems with the language AADL [27], [28]. The purpose it to take into account the shared resources and the software-to-hardware allocation for the analyses that come with the AADL toolset, in particular timing analyses. Comparing to our work, again, there are more details about the architecture than in an PML model, which is certification-oriented. Thus, it would be worth studying the generation of an PML model from such an AADL model. However, the information related to the load/store services that are needed by the software would need to be added to get an PML model.

b) Support to code generation: In [29], [30] the authors propose a metamodel of GPU architectures with the aim of supporting the development of application on such hardware platforms for non specialists in parallel programming. They extend the MARTE UML profile with a description of the allocation of data to memory elements. In [31], the authors extended an existing development framework dedicated to space application with the ability to handle multi-core platforms and time and space partitioning systems. This framework eases the development process by generating part of the code.

Focusing on multi- and manycore architectures, SHIM[5] (for Software-Hardware Interface for Multi-Many Core) is another framework dedicated to software design for multi- and many-core processors [32]. Its objective is to standardize the interface between the multi-core hardware and the software tools. It supports a precise description of the hardware components of the processor and its internal topology, including the processor cores, the inter-core communication channels, the routing protocols, the memory sub-system, hardware virtualization features, etc. The aim of SHIM is to provide a common metamodel enabling the use of many types of tools, including performance analysis, system configuration, auto-parallelizing compilers, and code generation. The approach of SHIM and AMALTHEA are very close in the sense they both provide a centralized model to support software design and code generation. They mainly differ by their respective application domain. AMALTHEA is promoted by automotive manufacturers for automotive systems. SHIM is developed by a consortium of multi- and manycore manufacturer for more general purpose software.

The approach of PML is similar in the way that the idea is to define a central model to be exploited in external views. However contrary to SHIM and AMALTHEA, which provide a detailed view of the architecture components, our model concentrates on an abstract definition only considering three types of component: *initiator*, *transporter* and *target*. Our claim is that such an abstraction is sufficient for interference and safety analyses.

[4]http://www.amalthea-project.org/

[5]https://www.multicore-association.org/workgroup/shim.php

VII. Conclusion and future works

We have defined a unified framework to analyze multi-core platform and partially answer the CAST32-A position paper. More specifically, we have abstracted a platform as the services it offers and modelled the interactions between software and complex hardware components via this service-based approach. The purpose was to propose a common model that covers the minimal concepts needed for both interference and safety analyses, which are required by the position paper. Thanks to this formalization, we have defined PML a metal-model dedicated to the description of any multi-core.

From such a description, we have implemented a tool that automatically generates the inputs needed for the analysis tools. For both perspectives, the approach consisted in hard-coded generic parts that can be reused for any platform and an automatic generation from the specific description of a platform. As PML encodes the minimal concepts, it is possible in each view to add more information (such as complex failure modes or failure propagation). We have run the framework on a realistic case study that was also used along the paper to illustrate the contributions.

In the future, we would like to extend PML (and the associated analyses) to consider cache coherence related behaviours. In our framework, a transaction is quite simple: it is represented by a simple sequence of connected components. With cache coherence, things become more complicated: requests can be broadcast, some transporters can initiate a transaction to send a data to another cache, etc. We also would like to better model the notion of *parallel* transactions: indeed, some components have some *capacities*, *i.e.* the ability to deal with several transactions in parallel to some extent.

Among the analyses required by the CAST32-A, there is a need to quantify the effects of interference. Thus, the applicant should define intensive benchmarking strategies [33], [34] in adequation with the interference. Thus, PML tooling should also propose an automatic translator to stressing benchmark for a given platform.

References

[1] Certification Authorities Software Team, "Multi-core Processors - Position Paper," Tech. Rep. CAST 32-A, Nov. 2016.

[2] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström, "The worst-case execution-time problem - overview of methods and survey of tools," *ACM Transactions Embedded Computing Systems*, vol. 7, no. 3, pp. 36:1–36:53, May 2008.

[3] R. Wilhelm and J. Reineke, "Embedded systems: Many cores - many problems," in *7th IEEE International Symposium on Industrial Embedded Systems (SIES'12)*, 2012, pp. 176–180.

[4] I. Villalta, U. Bidarte, J. Gómez-Cornejo, J. Jiménez, and J. Lázaro, "Seu emulation in industrial socs combining microprocessor and fpga," *Reliability Engineering & System Safety*, vol. 170, pp. 53–63, 2018.

[5] V.-A. Paun, B. Monsuez, and P. Baufreton, "On the determinism of multi-core processors," in *French Singaporean Workshop on Formal Methods and Applications*, 2013.

[6] L. Mutuel, X. Jean, V. Brindejonc, A. Roger, T. Megel, and E. Alepins, "Assurance of Multicore Processors in Airborne Systems," 2017.

[7] S. Houssany, N. Guibbaud, A. Bougerol, R. Leveugle, F. Miller, and N. Buard, "Microprocessor soft error rate prediction based on cache memory analysis," in *12th European Conference on Radiation Effects on Components and Systems (RADECS'11)*, 2011, pp. 412–419.

[8] Texas Instruments, "TCI6630K2L Multicore DSP+ARM KeyStone II System-on-Chip," Texas Instruments Incorporated, Tech. Rep. SPRS893E, 2013.

[9] M. Bozzano, A. Villafiorita, O. Åkerlund, P. Bieber, C. Bougnol, E. Böde, M. Bretschneider, A. Cavallo, C. Castel, M. Cifaldi *et al.*, "Esacs: an integrated methodology for design and safety analysis of complex systems," in *Proc. ESREL*, 2003, pp. 237–245.

[10] *Cecilia Workshop framework*, Dassault, 2014.

[11] H. Deschamps, G. Cappello, J. Cardoso, and P. Siron, "Coincidence Problem in CPS Simulations: the R-ROSACE Case Study," in *9th European Congress Embedded Real Time Software and Systems ERTS 2018*, Jan. 2018.

[12] H. Deschamps, "Scheduling of a Cyber-Physical System Simulation," Ph.D. dissertation, Institut Supérieur de l'Aéronautique et de l'Espace, 2019.

[13] V. Brindejonc and A. Roger, "Avoidance of dysfunctional behaviour of complex cots used in an aeronautical context," in *19eme Congrès de Maîtrise des Risques et Sûreté de Fonctionnement*, 2014.

[14] X. Jean, L. Mutuel, and V. Brindejonc, "Assurance methods for cots multi-cores in avionics," in *35th Digital Avionics Systems Conference (DASC'16)*, 2016.

[15] L. Mutuel, X. Jean, and V. Brindejonc, "Investigation of error types associated with failures in multicore processors," in *20eme Congrès de Maîtrise des Risques et Sûreté de Fonctionnement*, 2016.

[16] B. de Cat, B. Bogaerts, M. Bruynooghe, and M. Denecker, "Predicate logic as a modelling language: The IDP system," *CoRR*, vol. abs/1401.6312, 2014.

[17] S. Bayless, N. Bayless, H. H. Hoos, and A. J. Hu, "Sat modulo monotonic theories," *arXiv preprint arXiv:1406.0043*, 2014.

[18] P. Cuenot, K. Delmas, and C. Pagetti, "Multi-core processor: Stepping inside the box," in *Proceedings of 31st European Safety and Reliability Conference ESREL 2021*, 2021.

[19] A. Rauzy, "Mode automata and their compilation into fault trees," *Rel. Eng. & Sys. Safety*, vol. 78, no. 1, pp. 1–12, 2002.

[20] D. Thiele and R. Ernst, "Formal worst-case timing analysis of ethernet tsn's burst-limiting shaper," in *2016 Design, Automation & Test in Europe Conference & Exhibition, (DATE'16)*, 2016, pp. 187–192.

[21] S. Saidi and A. Syring, "Exploiting Locality for the Performance Analysis of Shared Memory Systems in MPSoCs," in *2018 IEEE Real-Time Systems Symposium, RTSS 2018, Nashville, TN, USA, December 11-14, 2018*, 2018, pp. 350–360.

[22] J. Abella, E. Quiñones, F. J. Cazorla, M. Valero, and Y. Sazeides, "Rvc-based time-predictable faulty caches for safety-critical systems," in *2011 IEEE 17th International On-Line Testing Symposium*, 2011, pp. 25–30.

[23] D. Hardy, I. Puaut, and Y. Sazeides, "Probabilistic wcet estimation in presence of hardware for mitigating the impact of permanent faults," in *Proceedings of the 2016 Conference on Design, Automation & Test in Europe*, ser. DATE '16. San Jose, CA, USA: EDA Consortium, 2016, p. 91–96.

[24] G. Macher, E. Armengaud, E. Brenner, and C. Kreiner, "A Lightweight Meta-Model to Support Automotive Systems and Software Engineering," in *9th European Congress on Embedded Real Time Software and Systems (ERTS 2018)*, 2018.

[25] S. Barner, A. Diewald, J. Migge, A. Syed, G. Fohler, M. Faugère, and D. G. Pérez, "Dreams toolchain: Model-driven engineering of mixed-criticality systems," in *Proceedings of the ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems*, ser. MODELS '17, 2017, p. 259–269.

[26] I. Agirre, J. Abella, M. Azkarate, and F. Cazorla, "On the Tailoring of CAST-32A Certification Guidance to Real COTS Multicore Architectures," in *12th IEEE International Symposium on Industrial Embedded Systems (SIES'17)*, 2017.

[27] J. Delange and P. H. Feiler, "Design and Analysis of Multi-Core Architecture for Cyber-Physical Systems," in *Embedded Real Time Software and Systems (ERTS2014)*, Feb. 2014.

[28] S. Rubini, P. Dissaux, and F. Singhoff, "Modeling shared-memory multiprocessor systems with AADL," in *Proceedings of the First International Workshop on Architecture Centric Virtual Integration co-located with the 17th International Conference on Model Driven Engineering Languages*

and Systems, ACVI@MoDELS 2014, Valencia, Spain, September 29, 2014, J. Delange and P. H. Feiler, Eds., 2014.

[29] A. W. De Oliveira Rodrigues, F. Guyomarc'H, and J.-L. Dekeyser, "An mde approach for automatic code generation from uml/marte to opencl," *Computing in Science Engineering*, vol. 15, no. 1, pp. 46–55, 2013.

[30] ——, "A Modeling Approach based on UML/MARTE for GPU Architecture," in *Symposium en Architectures nouvelles de machines (SympA'14)*, 2011.

[31] C. Honvault, J. Hugues, and C. Pagetti, "Model-Based Design, Analysis and Synthesis for TSP Multi-Core Space systems," in *9th European Congress on Embedded Real Time Software and Systems (ERTS 2018)*, 2018.

[32] M. Kondo, F. Arakawa, and M. Edahiro, "Establishing a standard interface between multi-manycore and software tools - shim," in *2014 IEEE COOL Chips XVII*, 2014, pp. 1–3.

[33] J. Bin, S. Girbal, D. Gracia Perez, A. Grasset, and A. Merigot, "Studying co-running avionic real-time applications on multi-core cots architectures," in *Embedded Real Time Software and System Conference (ERTS'14)*, 2014.

[34] S. Girbal, J. le Rhun, and H. Saoud, "METrICS: a measurement environment for multi-core time critical systems," in *9th European Congress on Embedded Real Time Software and Systems (ERTS'18)*, 2018.

Session Th.4.C

Assurance & Certification

Thursday 2nd June

14:00

–

Room Pastel

Toward the certification of safety-related systems using ML techniques: the ACAS-Xu experience

Christophe Gabreau*†, Adrien Gauffriau†, Florence de Grancey*‡, Jean-Brice Ginestet§, Claire Pagetti¶
* IRT Saint Exupéry, † Airbus, ‡ Thales, § DGA, ¶ ONERA

***Abstract*—In the context of the use of Machine Learning (ML) techniques in the development of safety-critical applications for both airborne and ground aeronautical products, this paper proposes elements of reasoning for a conformity to the future industrial standard. Indeed, this contribution is based on the EUROCAE WG-114/SAE G-34 ongoing standardization work that will produce the guidance to support the future certification/approval objectives. The proposed argumentation is structured using assurance case patterns that will support the demonstration of compliance with assurance objectives of the new standard. At last, these patterns are applied to the ACAS-Xu use case to contribute to a future conformity demonstration using evidences from ML development process outputs.**

***Disclaimer*: This paper is based on the EUROCAE WG-114/SAE G-34 standardization results at the time of the writing. Though some of the authors are active members of the working group, it is a free interpretation of the current draft work and only reflects the authors' view. As the working group has not published any released outcomes yet, some parts of the described argumentation may have to be modified in the future to conform to the final standard objectives.**

I. Introduction

A. Context

In the avionics context, the certification of aircraft systems is ruled by the regulation authorities, e.g. EASA for Europe and FAA for the United States. EASA developed Certification Specifications (CS 2x.1301/1309) defining the requirements that rule systems airworthiness. In addition to this, Authorities published AMC/AC (Acceptable Means of Compliance/Advisory Circular) to recognize that developing systems using industrial standards (ED-79A/ARP4754A for complex systems, ED-12C/DO-178C for software item and ED-80/DO-254 for hardware item) are acceptable means to show evidence that a system behavior, operating functions implemented by software and/or hardware items, is compliant with the regulation requirements.

Among the methodologies used for certification purposes, the assurance case concept is not new. The safety domain was one of the first to elaborate the safety cases concept. Safety cases were originally theorized by Tim Kelly [KBMB97] and then generalized by John Rushby [Rus15]. In particular, in [Rus15], Rushby claims that the introduction of this kind of methodology in the industries are *a significant contribution to system and software assurance and certification*.

B. Objectives of the paper

The first objective of the paper is to present the guidelines drafted by the EUROCAE WG-114/SAE G-34 joint working group (WG-114 subsequently) on the certification of ML-based systems. The draft guideline [EUR21] is called AS6983 in the rest of the document. Those guidelines cover 3 levels of engineering:

1) System/Subsystem: Classical system and safety assessment processes are used to capture the requirements allocated to the ML-based function and identify the items that will be used to implement the system. Among these requirements is the DAL (Design Assurance Level) that modulates the fulfillment of the assurance objectives. The modulation is not used in the paper because the levelling of objectives has not been discussed yet in WG-114.
2) ML Constituent: This level has been introduced by the WG-114 between subsystem and item in order to support the design of one ML-based subsystem function that can be deployed on several items. This level encompasses the data management process (design the datasets that will be used to train, test and validate the ML part of the ML constituent) and the ML model design process (train/validate the model to fit the intended function and verify that its properties are compliant with its ML model requirements).
3) Item: Both classical and ML specific process will be used to transform the designed model into an implementation model that will be hosted in a SW or and HW item.

These 3 levels of engineering are described in an end-to-end ML-based system development workflow (see Fig. 1). The semantics of the arrows is as follows: plain thick arrows mean *is an input*, plain arrows mean *produces* and dashed arrows mean *uses*.

The second objective is to structure those guidelines with a set of assurance cases patterns that will reflect the learning assurance objectives developed by WG-114 and support a possibly future demonstration of conformity for certification purposes.

The third objective is to apply the assurance cases patterns on a detailed use case. Indeed, we will apply them on the hybrid architecture promoted by [DDGG+21] to implement an Airborne Collision Avoidance System (ACAS-Xu) for drones. The purpose of this architecture is to embed several neural

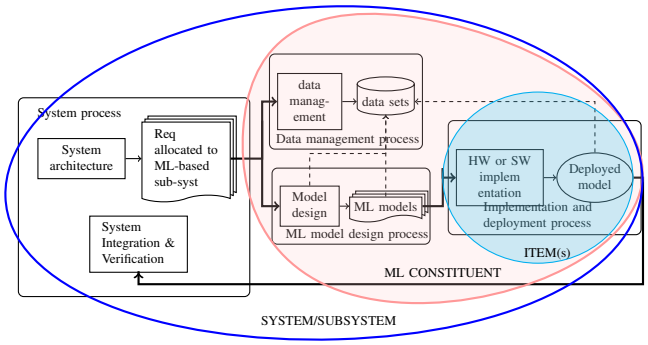


Fig. 1. End-to-end ML-based system development workflow

networks (NNs) to replace the ACAS-Xu standardized Look-Up Tables (LUT) along with a safety net based on extracts of the LUT. In that former work, we proposed an initial end-to-end certification strategy already inspired from the WG-114 works and which was reflecting the progress status of the standardization approach at this point. The WG-114 approach has evolved since and we will present an updated version in line with the current AS6983 guidelines.

II. Standardization approach

Today the WG-114 is a worldwide group of more than 500 engineers that draws its expertise from a large variety of industry fields such as air-framers, Unmanned Aircraft systems (UAS), Urban Air Mobility (UAM), electric Vertical Take-Off and Landing (eVTOL) manufacturers, engine manufacturers, airborne and ground equipment manufacturers, regulators or air navigation service providers.

A. Overview of the guidelines

As mentioned in their "Statement Of Concerns (SOC)" [EUR20a], the WG-114 anticipated *a growing commercial pressure for Artificial Intelligence (AI) solutions within the aerospace industry over the coming few years, there is an urgent call for regulation and the emergence of norms around acceptable usage*. The role of this joint group will be to produce a new standard for the development and the certification of aeronautical products using artificial intelligence (once recognized as an acceptable means of compliance by the adhoc authorities). The standard will be multi-domains, the joint working group will evaluate key applications for AI usage within aeronautical systems, with a scope encompassing ground-based equipment, airborne vehicles and Air Traffic Management (ATM) systems. In terms of processes, the full life cycle will be under consideration, from design and manufacture, to operation and through-life maintenance. In its first issue, the standard will focus on *offline trained Machine Learning (ML) based systems* meaning that the embedded ML algorithms are only trained on ground and does not keep on learning during operation.

The SOC document has allowed the whole industry to align on the main concerns related to the use of AI in the safety-related systems that make the aeronautical ecosystem. One of the main challenges raised by the document was to determine how the future standard was going to interfere with the other existing standards that currently rule the development and the certification of the aeronautical systems. This paper intentionally focuses on avionic systems processes and their applicable standards: safety assessment (ARP4761 [SAE96]), system development (ARP4754A/ED-79 [SAE10]), software development (DO-178C/ED-12C [RTC11]) and hardware development (DO-254/ED-80 [RTC00]). In order to propose an end-to end certification approach, the WG-114 considered all these processes and has identified the gaps with the existing standards in the chapter 4 of the SOC document. At this point of time, there is a collegial consensus around a *process-oriented standardization approach* introducing new assurance objectives (a.k.a. learning assurance objectives) to cover the development processes and fill the identified gaps. Therefore this paper considers the ongoing work of the working group and zooms on the levels of engineering that are impacted so far.

B. Application to the use case

The airborne anti-collision system ACAS X [KHC12] was developed to overcome some limitations of TCAS systems and as a response to flights traffic increase. In the ACAS Xu standard, the design relies on a set of offline computed lookup tables to make avoidance decisions. Some works [KBD+17] proposed to replace those LUT with some surrogate neural networks: the purpose was to reduce the memory footprint and thus to improve the execution time. The former work [DDGG+21] relies on neural networks as proposed by [KBD+17] but also on a safety net based on an extract of the LUT to ensure the compliance with the reference behavior given by the LUT. This ACAS Xu hybrid architecture allows the use of ML algorithms while guaranteeing the safety of the system in all operational domain. At operation (see Figure 2), when the check module meets a pre-identified zone where the ML controller predictions are incorrect, the system switches to the safety net. In the other cases, the ML controller is operated.

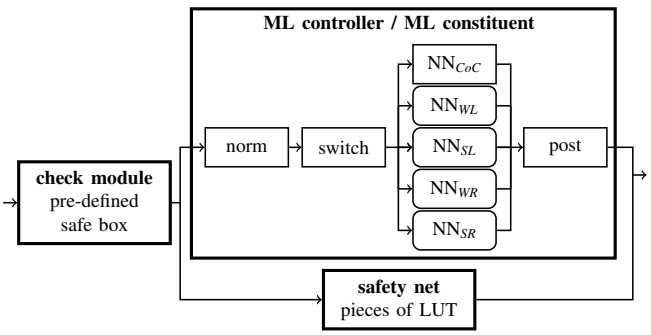


Fig. 2. Hybrid Architecture Overview

We will apply the assurance case patterns on the ACAS-Xu use case and demonstrate the compliance to learning

assurance objectives with supportive evidences. Indeed we will use evidence artefacts that have been developed during the design (ML algorithm development, performance measures, generalization demonstration using formal methods), the implementation artefacts demonstrating that the inference model is preserving the properties of the design model and the tests of the hybrid controller performed with simulation means.

C. Overview of the assurance case concept

Assurance Cases are a method to structure an argumentation in order to make a demonstration of conformity. It has been used for a while for safety demonstration in the whole industry. This method has been theorized by John Rushby in [Rus15] with this definition: *Assurance cases are a method for providing assurance for a system by giving an argument to justify a claim about the system, based on evidence about its design, development, and tested behavior*. Indeed, John Rushby emphasizes that *the argument provides a context in which to justify and assess the quality and relevance of the evidence submitted, and the soundness of the reasoning that relates this to the hierarchy of subclaims that leads to the top-level claim*. When building the argumentation related to a novel technique of development (such as Machine Learning), this may help to identify the lacks, weaknesses of the argumentation tree and thus the places where some additional research is needed.

In the updated edition of "The uses of argument" [Tou03], Toulmin states that the argumentation is mixing logic and epistemology. Considering the latter, he states that the claim of knowledge leading to the argument soundness should not be questionable, *a man who puts forward some proposition, with a claim to know that it is true, implies that the grounds which he could produce in support of the proposition are of the highest relevance and cogency: without the assurance of such grounds, he has no right to make any claim to knowledge*.

This being said, Rushby in [Rus15], clearly separates 2 kinds of argumentation strategies:

- Reasoning steps *are interpreted logically: we must determine if the conjunction of subclaims in a step deductively entails its claim*. So it may lead to a problem in logic: *do the subclaims truly imply the claim?*.

- Evidential steps *are interpreted epistemically: they are the bridge between our concepts (expressed as subclaims) and our knowledge of the world (recorded as evidence)*. Therefore it may lead to a problem in epistemology: *does the evidence amount to knowledge that the claim is true?*

Both questions are justified and emphasized in [Lev11] which highlights the possible misuse of an assurance case due to the confirmation bias (experts may try to build assurance cases enforcing the compliance of their own system) and illustrated in the Nimrod accident report [HC09]. However as underlined by [Lev11]: *the type of evidence required and assurance arguments used are straightforward with prescriptive regulation*. As mentioned earlier in the paper, this is actually the approach of the WG-114 standardization group. Indeed the working group is a college of experts challenging themselves and working on a consensus base to provide the best process-oriented assurance activities to ensure the certifiably of aeronautical systems.

Most of the recent papers on assurance cases rely on the GSN notation developed by Tim Kelly from York university [KW04] and standardized by [ACW18]. The graphical notation is really a good format to ease the analysis, favour the use of patterns, ease the concurrent working and allow for a global view of the work. However we do think that this is not enough to fully describe the assurance case content, permit a fine configuration control and express the changes between versions that are necessary to control the credit for the final goal of the process assurance: the demonstration of conformity. Therefore we prone to mix graphical and textual notation to provide the interesting aspects of the both notations as it is done in the certification argumentation of the SAFEGUARD system in [MSG21]. At last, and to echo the previous paragraph about the confirmation bias, one can challenge the reasoning or the fact that evidences may not be sufficient to fully achieve the demonstration. To help solving the legitimate interrogation inherent to an assurance case in [Rus15]: *doubting that the subclaims to a reasoning step really do entail its claim, or that the evidence cited in an evidential step has adequate weight*, Rushby was proposing the useful concept of defeater. The GSN standard V3 [ACW21], by introducing dialectic principles now allows for challenging the reasoning or the evidential steps.

D. Notation

The assurance case extracts are patterns containing AS6983 guidance objectives as proposed by [HG18] for DO-178 and [DPP20] for the multi-core guidelines formerly detailed in the CAST-32A. From such patterns, it is up to the applicant to instantiate them for a given product. In this paper, we illustrate the instantiation on the ACAS Xu use case. The assurance case patterns were designed with the GSN V3 [ACW21] (note that they are standard assurance cases for GSN as we did not use the GSN pattern notations). The instantiations are mainly described textually. In addition, we use the following colour codes all along the paper: the grey boxes indicate that the goal is supported by classical well-known guidance, the white boxes are used to structure the argumentation and yellow boxes contain objectives from the AS6983 draft guidance. Any other colour is used to link the assurance case extracts to one another in order to ease the navigation.

III. ML-BASED SUBSYSTEM DEVELOPMENT

The future ML standard will propose an end-to-end guidance to develop and certify a system containing a ML-based function. However the WG-114 approach will try to stick on the existing guideline as much as possible. From an airborne perspective, this means using the ARP4754A [SAE10] guidance whenever possible to integrate the ML-based function at subsystem level. Most of the processes at system/subsystem level of engineering (safety assessment, system requirements capture, architecture, integration, validation and verification

processes) are reused from the ARP4754A. The AS6983 overlaps a bit with the system/subsystem ARP guidance for the beginning of the *ML constituent* development.

A. Assurance case pattern

The argumentation is based on the application of the ARP4754A objectives [SAE10] from table A-1//2.0 Aircraft and System Development Process and Requirements Capture and //5.0 Implementation verification process. As this is a pure ARP4754A pattern, this argumentation is not detailed it in the paper. It leads to the identification of the ML constituent development goal: *The ML constituent performs its intended function at acceptable level of safety for allocated DAL.*

B. Application to ACAS Xu

The ACAS Xu use case subsystem is composed of:

1) ML controller (or ML constituent): contains all the ML models approximating the LUTs in every points of the input space. The models are hosted in the *SW item 3* whereas the traditional functions (normalisation, selection of the NNs and post processing to compute the advisory maneuver) are hosted in *SW item 2*.
2) SW item 1: contains the *safety net* and the *check module*.
3) HW item: the Texas Instrument keystone platform hosts the 3 SW items defined above.

The process flow chart to produce the ACAS-Xu system (see figure 3) is an application of the ML-based system/subsystem development process from WG-114.

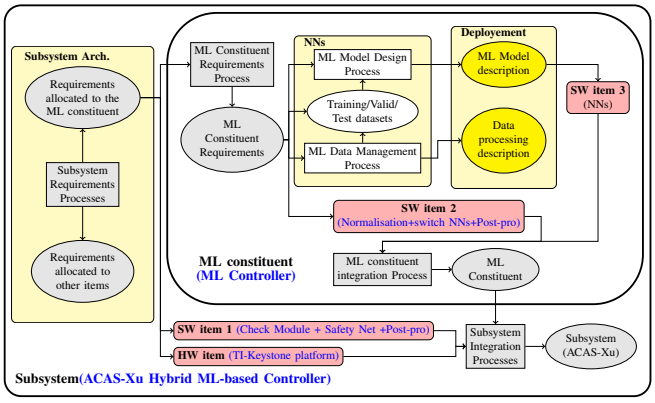


Fig. 3. ACAS-Xu system development workflow

The ARP4754A objectives from table A-1//2.0 Aircraft and System Development Process and Requirements Capture are supported by the following evidences:

- ARP4754A-2.3 (System requirements): The hybrid ML-based controller requirements are defined. The ACAS Xu function is specified using the RTCA SC-147 /EUROCAE 75 standardized tables.
- ARP4754A-2.4 (Derived requirements): The hybrid ML-based controller derived requirements are defined and rationale explained. The subsystem Operational Design Domain (ODD in the rest of the document) is partitioned between the ML controller and the safety net.
- ARP4754A-2.5 (System architecture): The strategy for the architecture definition has been developed earlier in the section.
- ARP4754A-2.6 (Item allocation): The hybrid ML-based controller requirements are allocated to items. The strategy to satisfy the objectives is to develop the entities (actually 2 items and a ML constituent) defined by the subsystem architecture. The traditional SW items Check-Module, Safety-Net and Post-pro are developed to DO-178C guidance and the HW item (the NVIDIA Jetson Xavier platform) to DO-254 guidance.

The ARP4754A objectives from table A-1//5.0 Implementation verification process are supported by the following evidences:

- ARP4754A-5.3 (Item implementation): The hybrid ML-based implementation complies with the subsystem requirements. The ML constituent development and verification processes are further elaborated to detail the demonstration of compliance to the future ML standard.
- ARP4754A-5.2 (System verification): The subsystem verification demonstrates the intended function and the confidence of no unintended function impacts to safety. The Hybrid ML based controller is tested against its functional, safety and operational requirements in a simulated environment (including a LUT simulator for comparison).

IV. ML CONSTITUENT DEVELOPMENT

The *ML constituent* contains a ML model and possibly traditional items. Its current definition is: *a defined and bounded set of either hardware item(s) and/or software item(s) that implement ML model(s) and associated ML data processing which are grouped for integration purpose to support(s) one subsystem function.*

A. Assurance case pattern

The certification argumentation is described through the fulfillment of the goal previously described at subsystem level: *The ML constituent performs its intended function at acceptable level of safety for allocated DAL.* Considering the ML constituent may contain both ML model and traditional items, the argumentation strategy is based on both new and classical guidance. The upper level goals of the ML constituent are described in the figure 4.

The learning assurance objectives are divided into 4 goals. The first *Goal req* concerns the ML constituent requirements capture. This goal is refined as an assurance case shown figure 5). *The ML constituent requirements are a satisfactory refinement of the allocated system requirements for the selected DAL.* The strategy to fulfill this goal is supported by 2 levels of requirements: the ML functional requirements and their breakdown into ML model requirements and traditional items identified by the ML constituent architecture:

- ML functional requirements: The ML functional requirements are a satisfactory refinement of the allocated subsystem requirements (functional intent, ODD specifica-

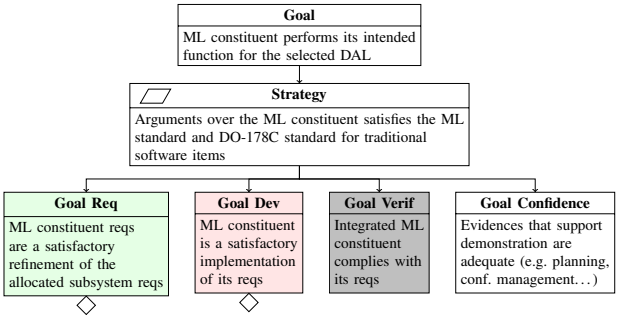


Fig. 4. ML constituent pattern

tion, robustness, DAL, system performance and resources constraints).

- Item requirements: Traditional item requirements are a satisfactory refinement of the allocated ML functional requirements for the selected DAL. The items are specified using the DO-178C guidance.
- ML model/data requirements: The ML model and data requirements are a satisfactory refinement of the allocated ML functional requirements for the selected DAL.

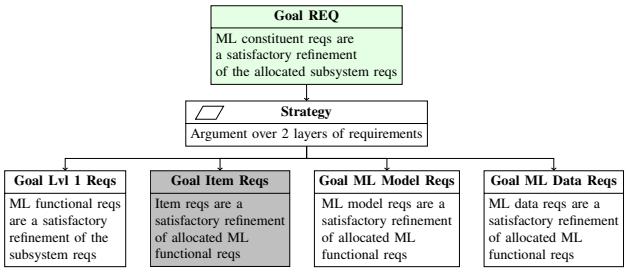


Fig. 5. ML constituent requirement capture pattern

The second goal of figure 4 is *Goal Dev* and concerns the ML constituent development. It is further developed as the assurance case of figure 6. We will detail it later, at the end of this section, as it is central in the WG114 guideline.

The third goal of figure 4 is *Goal Verif* and concerns the ML constituent integration and verification. The ML inference model and the traditional items are integrated to make the ML constituent. The goal is then to show evidence that *The integrated ML constituent (ML inference model and traditional items) complies with the ML constituent requirements*. This goal is not further detailed as it will be evidenced by traditional process artifacts.

The fourth goal of figure 4 is *Goal Confidence*. We have to show confidence that the evidences supporting this argumentation are adequate to the ML constituent development process. As a consequence, the argumentation will rely on the fulfillment of transverse objectives (from AS6983) regarding planning documentation, configuration, change management, quality assurance process and certification liaison. This latter part will not be further detailed.

Let us go back to *Goal Dev* detailed in figure 6. The related strategy is developed in the context of the ML constituent architecture (breakdown into the ML model and the traditional items).

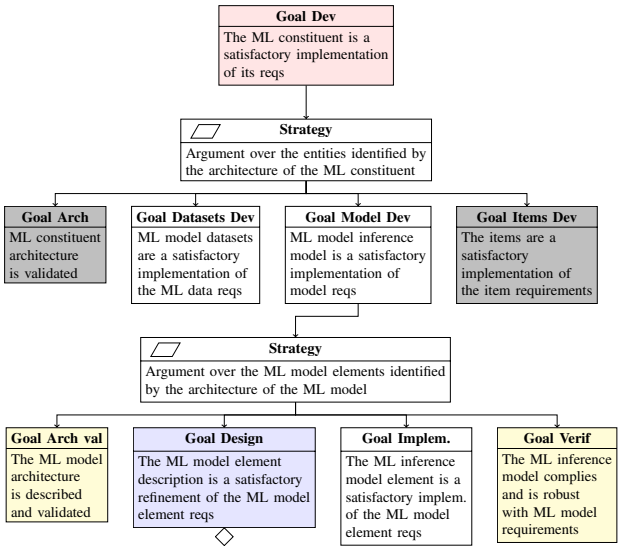


Fig. 6. ML constituent development pattern

It leads to a first level of 4 sub-goals: the *Goal Arch* validates the ML constituent architecture. The *Goal Items Dev* covers traditional items development to classical guidance. The *Goal Datasets Dev* addresses the datasets development, it will be further developed in section V. Eventually the *Goal Model Dev* (*"The ML inference model is a satisfactory implementation of the ML model requirements"*) is elaborated taking into account that the ML model may be decomposed in several ML model elements. This architecture is identified as an element of context, so that the argumentation can be based on each model element identified by the architecture. From this point, the ML model is designed, implemented and verified:

- *Goal Arch Val*: The ML model is breakdown into model elements and the architecture is validated.
- *Goal Design*: Each ML model element is trained, validated and verified using the datasets. Then the main goal is to demonstrate that *"the ML model element description is a satisfactory refinement of the ML model requirements"*. This goal is detailed in section VI.
- *Goal Implem.*: When the ML model design is terminated and frozen, each of the ML model element is implemented to make a ML inference model element. This means that the main goal becomes *"the ML inference model element is a satisfactory implementation of the ML model element description"*. This goal is detailed in the section VII.
- *Goal Verif*: When all the ML model elements are implemented, they are integrated to make the ML inference

model. Then the verification goal becomes: "*the ML inference model complies and is robust with ML model requirements*".

B. Application to ACAS Xu

The *ML constituent* (ML controller) logical architecture is composed of 4 parts (see figure 2) among which 3 traditional parts and 1 ML model:

- the 3 traditional parts are:
  - the *norm* normalizes the values of $\rho, \theta, \psi, v_{own}, v_{int}$ in the interval [-1, 1];
  - the *switch* is in charge to select the $\text{NN}_{pa,\tau}$ that will compute the advisory. The selection is done depending on the inputs pa and τ. In effect, the value of τ is decomposed in 9 possible ranges (e.g. $\tau = 0$ is the first range);
  - the *post*-processing instructions;
- the ML model is composed of 45 elements more precisely, 45 NNs named $\text{NN}_{pa,r}$ with $pa \in \{\text{CoC, SR, SL, WR, WL}\}$ and $r \in [1, 9]$.

Provided that the ML controller architecture is an element of context, the assurance case pattern of figure 4 is applied to the ACAS-Xu use case.

1) *ML controller requirements capture* (figure 5 pattern):
   - The ML controller requirements are refined from the requirements of the Hybrid ML-based controller subsystem (ODD, real-time constraints, anti collision performance, memory size constraints, DAL).
   - The SW item 2 (Normalisation+switchNNs+Post-pro) is specified using the DO-178C guidance.
   - The capture of the NNs requirements is described in section VI.
   - The NNs data requirements are specified by the Operational Design Domain (ODD), defined through the input points of the LUTs from the RTCA SC-147 Minimum Operational Performance Standards (MOPS) For ACAS-Xu. The ODD is divided into sub-ODDs to fit the 45 ML model elements of the ML model architecture.
   - The NNs data requirements are checked for traceability against hybrid ML-based Controller requirements, consistency and compatibility with NNs requirements.
2) *ML controller development* (figure 6 pattern): The ML controller architecture document is created and validated. The SW item 2 is developed according to DO-178C guidance. The datasets are developed and verified against the ML model data requirements defined previously. The ML model (NNs) is developed:
   - Goal design instantiation: The ML model is breakdown into 45 ML model elements (45 NNs). Specifically, it is verified that the union of the 45 ODDs makes the ML Controller ODD. The ML Model Description (MLMD) document is created with the NNs architecture and validated. Each NN is trained, validated and verified using the related datasets. Each NN description is added to the MLMD.
   - Goal implem instantiation: The 45 NNs are implemented from their ML model element description. See section VI for details.
   - Goal verif instantiation: The in-sample generalization capability of the NNs has been formally proven in the design phase. In addition, it has been demonstrated that the implementation process does not alter the semantics of the NNs. Therefore there is no need to demonstrate the robustness/stability in the inference environment. The Requirement-based verification is covered by the verification activities performed at ML controller level.
3) *ML controller test* (figure 4 pattern): The ML controller is verified against its requirements in a simulated environment (including the LUT simulation for functional/safety testing). These activities are covered by classical guidance.

V. Data Management

The objective of the data management process is to deliver trustworthy training, validation and test datasets which will be used to design, implement and integrate the ML model, in order to achieve the delivery of the ML inference model that meets the functional and operational requirements. The figure 7 is an overview of the data management process as per WG-114 current work.

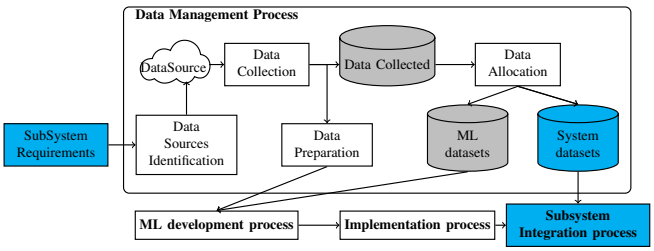


Fig. 7. Data Management Process

A. Assurance case pattern

The data management process refined the *Goal Datasets Dev* of figure 6 and is detailed in the full assurance case (but not in this paper due to lack of space). This process has to be covered by specific learning assurance objectives in order to guarantee the training of ML algorithms in the context of safety-related functions. The argumentation is split in 2 strategies applicable to ML model data: data management (sources identification, data collection, preparation and allocation) and verification.

B. Application to ACAS Xu

The ACAS Xu use case has only few activities in terms of data management as the ML algorithms are trained with the standardized LUTs from the MOPS [EUR20b]. Thus

dataset = LUTs

VI. ML MODEL DEVELOPMENT

The *ML constituent development process* has been fully described in the section IV-A. In particular, the ML constituent has first been decomposed into ML model and traditional items. This section describes the AS6983 assurance objectives that cover the *ML model (only, that is part of the ML constituent) development process* and its output: the ML model description. This artefact is essential, it shall contain all the necessary information for the implementation of the ML model.

We remind that the *ML model* itself has been broken down into *ML model elements* that have to be trained, validated and verified. The development of the *ML model element* is based on 2 main goals identified in the previous sections:

- *Goal Model Reqs* (Figure 5): *The ML model requirements are a satisfactory refinement of the allocated ML functional requirements*. There is a second level of refinement to define the ML model element requirements. The strategy is double: capture the requirements that are necessary for the model element development (e.g. specification, performances, generalization, robustness, stability) and validate them. Due to the lack of space, the assurance case about the ML model element requirements capture are not detailed in this paper.
- *Goal Design* (Figure 6): For each model element of the ML model architecture, *the ML model element description is a satisfactory refinement of the ML model element requirements*. This process is detailed in the next paragraph.

A. Assurance case pattern

This section focuses on the design assurance process of the ML model element. As described in figure 8, the argumentation to fulfill the *Goal design* is based on 2 strategies:

1) The ML model element is designed.
2) The ML model element is verified.

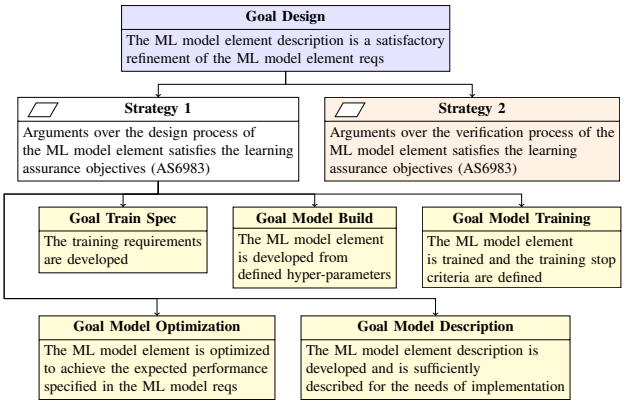


Fig. 8. ML Model design pattern

Sub-assurance case rooted from Strategy 1 (cf figure 8): The design assurance process of the ML model element is based on the following goals:

- *Goal Train Spec*: develops the training requirements. ML training activity could introduce the use of randomization that may alter the determinism and the repeatability of the design process of the ML model design process. In such cases, additional data (such as seeds values used to generate random numbers) should be defined as derived ML training requirements and managed through configuration management.
- *Goal Model Build*: selects/optimizes the hyper-parameters of the ML model element from the ML model requirements and training requirements. The ML model element is developed from these hyper-parameters.
- *Goal Model Training*: determines the ML model element parameters using the appropriate ML training algorithm and the training/validation datasets to meet the applicable requirements of the ML model element and the ML training. The initial values of the ML model element parameters, the loss function, the evaluation metrics and the training stop criteria are defined from the ML training requirements.
- *Goal Model Optimisation*: consists in performing changes in the ML model element after the training phase to achieve the expected performance specified in the ML model element requirements. In case the performance is deemed not satisfactory, derived requirements are developed to specify the required changes.
- *Goal Model Description*: develops the sufficient documentation of the ML model element design to permit the implementation of the ML model element (into a ML inference model element) including the pre/post processing instructions. The ML model element description is a part of the ML model description. There are 2 types of implementation of a ML model element, either an exact or an approximated replication of the ML model element semantics. Each ML model element description contains the design characteristics of the model element ensuring its exact (or approximated) replication in the execution environment: hyper parameters and parameters, analytical/ algorithmic syntax and semantics, replication criteria, execution environment.

Sub-assurance case rooted from Strategy 2 (from figure 8 and refined in figure 9): The verification assurance process of the ML model element is based on the following goals:

- *Goal Validation*: checks the training requirements for correctness and completeness against the ML model element requirements. ML training requirements should conform to the ML design standards and be traceable or justifiable, verifiable and consistent.
- *Goal Performance*: ensures that the performance requirements, including functional and non-functional aspects are met.
- *Goal Robustness*: ensures that the ML model element can

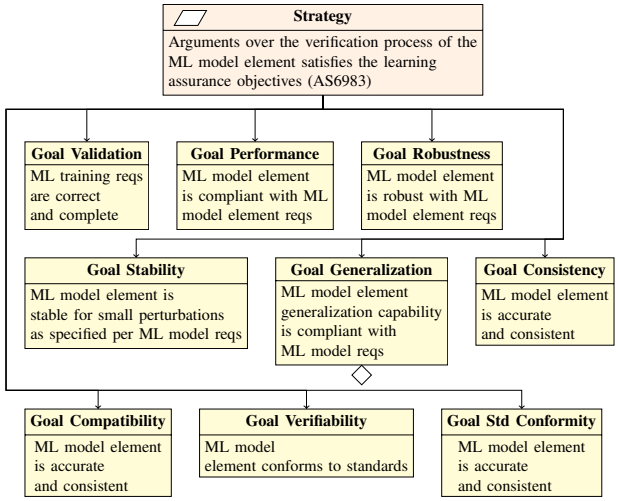


Fig. 9. ML Model verification pattern

continue to operate correctly despite abnormal inputs and conditions.

- *Goal Stability*: ensures stability of the ML model element, i.e. that small perturbations in the inputs do not activate unintended behavior. The expected level of perturbation that the ML model element should sustain has been specified in the ML model element requirements.
- *Goal Generalization*: The verification of the ML model element generalization capability is to ensure that the model element will show the same performance with unknown inputs as the one measured during the ML model element training. This argumentation (cf figure 10) is further detailed in the next paragraph.
- *Goal Accuracy and consistency*: determines the correctness and consistency of the ML model element (e.g. stack usage, memory usage, fixed point arithmetic overflow).
- *Goal Compatibility*: ensures that no conflict exist between the ML model element and the hardware/software features of the target platform, especially for the system response time and the input/output hardware.
- *Goal Verifiability*: ensures that the ML model element does not contain elements or structures that cannot be verified, for example neurons which can never be activated during ML Model element testing or random functions which cannot be reproduced during testing.
- *Goal Std Conformity*: ensures that the ML design standards are followed during the design process of the ML model element and that deviations from these standards are justified.

B. Application to ACAS Xu

Requirement considerations - each of the 45 ML model elements (NN) is specified and validated.

- Behaviour: each NN shall replicate the LUT prediction in its allocated ODD (LUT property).
- Performances: memory footprint, timing and accuracy
- Generalization capability: the LUT property shall be preserved whatever the position of the ownship and the intruder in the input space of the ODD allocated for the ML model element.

Design considerations - the strategy 1 is detailed:

- Goal train spec instantiation: Training is defined to perform a regression task. The training metrics is defined as the mean square error between truth costs and predicted costs.
- Goal model build instantiation: The identified hyperparameters consist in a set of model architecture parameters (numbers of neurons, number or layers, activation function) and training parameters (size of batches, learning rate). Hyperparameters are defined after an optimization search phase performed with Bayesian optimization.
- Goal model training instantiation: The NN is trained until a fixed and large number of epochs is attained. An early stopping criteria is added to avoid overfitting behaviour.
- Goal model optimization instantiation: Pruning methods are used to optimize the NNs memory footprint. Performances are measured and the best model element is retained.
- Goal model description instantiation: The ML controller contains the architecture description of 45 ML model elements (NN). Each NN description contains the design characteristics of the network: hyperparameters and parameters, analytical/algorithmic syntax and semantics, replication criteria, execution environment. The exact replication is selected so that the inference model can inherit from the model performances attained during the design phase if the model semantics is preserved during implementation.

The strategy 2 is substantiated regarding the verification of the performance (each NN accuracy and memory footprint are verified against the NN requirements). There are no verification needs for the robustness and the stability because there are no such requirements.

The generalization aspect is fundamental to the certification demonstration. The NN generalization property is demonstrated by proving that the LUT property is preserved for each NN whatever the ownship/intruder situation in the input space (cf figure 10). The argumentation is split in 2 parts:

1) Identification - All the input situations where the NN and the LUT predictions are different, are considered as incorrect (the NN does not preserve the LUT property).
2) Mitigation - This part is already addressed per the subsystem architecture design: the ACAS-Xu hybrid ML-based controller switches from the ML model (NNs) to the LUTs (Safety Net) when incorrect situations are detected (this is already described in the ACAS-Xu subsystem architecture document).

Therefore only the identification is at stake. Indeed, all the inputs where the NN predictions are incorrect should be identified, i.e. wherever the LUT property is not true in the

input space. The method is to partition the input space (ODD) into into p-boxes (where corners are the points of the LUTs). Then the LUT property is checked in all p-boxes: for each p-box, it is verified that the NN prediction is the same as the true prediction of one of the p-box corner points. As per the paper [DDGG+21], the verification is performed using 3D boxes. Formal methods are used to make the demonstration. The argumentation is decomposed into 3 goals:

- The LUT property is correctly defined. Actually this proof is already available in the ML model requirements validation report.
- The input space (ODD) is correctly decomposed into 3D-boxes (proof: Generalization analysis report).
- The LUT property is formally checked in each 3D-box of the input space (proof: Generalization analysis report)

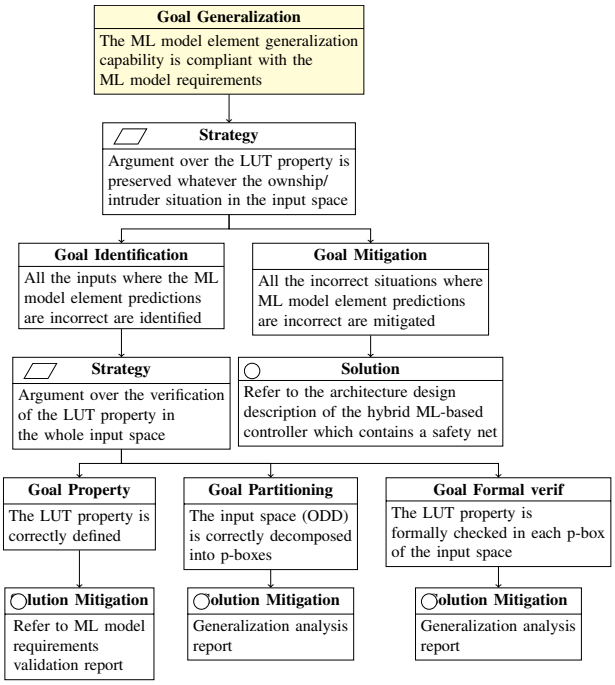


Fig. 10. ML Model Generalization demonstration for ACAS-Xu

VII. Implementation and deployment

The input for the implementation phase should be a trained ML model element that has a complete ML model element description. The implementation process (see figure 11) produces a ML inference model element that is capable to infer on a HW or SW item. It may also optimize the ML model element (without possible retraining) in order to increase the computation performance or better fit the targeted hardware resources, however it must ensure that any optimization preserves the semantics of the input ML model element or at least leads to an acceptable deviation (e.g. merging of convolution/batchnorm/ReLu layers or Winograd algorithms).

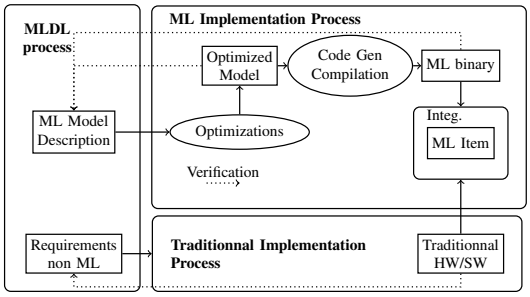


Fig. 11. Implementation process

A. Assurance case pattern

The main goals for the implementation process are the following (the assurance case extract is not provided due to the lack of space):

- *The description of the ML inference model element is a satisfactory refinement of the ML model element description* - The sub-goals are:
  - *Semantics preservation*: Any modification of the model element semantics due to the transformations (optimizations, conversion to the target environment) should be analysed for their impact on the model element performance and behavior with respect to the model element requirements.
  - *Training and target environment differences*: Differences between the 2 environments are identified to assess the impact on behavior and performance of the ML model element, and to evaluate to what extent the activities performed in the design environment can be used as verification credit for the demonstration that the inference model element complies with the model element requirements.
- *The SW or HW item is a satisfactory implementation of the ML model element description allocated to the item and meets the target constraints*: According to the AS6983 current guidance, this objective can be fulfilled using the current standard practices.
- *The integrated HW/SW items (host of the ML inference model element) comply with the ML model element description* - The sub-goals are:
  - *Compatibility with the target*: Timing, memory, latency, throughput and other non-functional requirements should be satisfied by the ML inference model element running on the target environment.
  - *Unintended behavior detection*: The ML inference model element does not introduce unintended behavior relative to the ML Model element.
  - *Design performances preservation*: the performance of the HW/SW item(s) implementing the ML inference Model on the test data set should be verified and documented. Any deviation should be measured and reported to the safety assessment process.

- *The verification of the verification is achieved*: The verification procedures are correct and complete against the ML model requirements. Functional and structural coverage are verified.

B. Application to ACAS Xu

The ML controller is implemented on a TI Keystone platform. The 45 NNs are hosted in a SW item (SW item 3 on a ARM unit). In this context, the ML element description document is updated and validated.

Hereunder the substantiated goals:

- *ML inference model element*:
  - *Semantics preservation*: No post-training optimization is performed. The 45 models are converted to a format that is compatible with the inference platform. The models semantics is described in the ML model description and is preserved during the implementation. This should be detailed in a dedicated dossier for certification purposes.
  - *Training and target environment differences*: The inference environment is different from the development environment, however the execution on the ARM processor is expected to be identical provided that the numerical representation and resolution are the same on both platforms. Then, credit that can be sought for the formal verification of the generalization capability performed during the design phase. A representativeness dossier should be provided for certification purposes.
- *SW or HW item implementation*: The NNs are coded in C language using specific libraries for target integration.
- *Integrated HW/SW items*:
  - *Compatibility with the target*: OTAWA tool is used to compute the memory footprint and consolidate the theoretical memory footprint evaluated during the design phase.
  - *unintended behavior*: As the exact replication of the ML model element is demonstrated then a verification credit can be sought for the formal verification activities of the design phase. This covers the objective.
  - *Design performances preservation*: Test dataset should be used to verify that the 100% accuracy objective is met in the target environment and that timing requirements is attained. This is not yet done.
- *Verification of the verification*: Not performed.

VIII. Related work

In parallel to WG-114, research field groups work on determining and solving the challenges to certify AI-based systems. For instance, the ANITI[1]/DEEL[2] research project released a comprehensive list of certification issues in their white paper "Machine Learning in certified systems" [DCW21]. The main challenges and the way the WG-114 is tackling them are synthesized in the article [FK21].

There are a lot of recent works fostering the use of assurance cases. In [Rus15], John Rushby explains the fundamentals of the theory for the use and the evaluation of the assurance cases. Michael Holloway [HG18], on behalf of FAA, translates the EUROCAE/RTCA ED-12C/DO-178C standard [RTC11] in an assurance case and expresses the underlying arguments which justify the assumption that the document meets its stated purpose of providing guidelines for avionic embedded software. [CPH20] presents patterns that can be used to develop assurance arguments for demonstrating the safety of the ML components. The argument patterns provide reusable templates for the types of claims that must be made in a compelling argument. [MSG21] is the first complete and published demonstration that a system (SAFEGUARD system enforces geofencing restrictions on unmanned aerial vehicles) possesses the overarching properties for certification approval purposes. At last, the work [DPP20] is proposing a domain-agnostic method to design and evaluate patterns (*the design pattern approach is a way of describing a recurring problem and its associated solution based on best practices*) of assurance cases. This will be very useful when time comes for constructing and releasing patterns from the collection of assurance cases available on the field. At last the safety group of university of York, with their "Guidance on the Assurance of Machine Learning in Autonomous Systems (AMLAS)" [RHH21], *defines a safety argument pattern that can be used to explain how and the extent to which the generated evidence supports the relevant ML safety claims, explicitly highlighting key assumptions, tradeoffs and uncertainties*. They suggest an end-to-end process, from system safety requirements to ML component safety case, providing guidance for both the applicant and the certification authority.

IX. Conclusions

This paper has proposed an interpretation of the current WG-114 work (future standard for offline machine learning development) through the development of assurance case patterns. These patterns have been applied to the ACAS-Xu use case, in order to structure a possible argumentation for demonstrating the conformity to the objectives of the standard. The demonstration is obviously not complete, however the main learning assurance objectives have been tackled to show evidence that the proposed process for ACAS-Xu development is certifiable. Beyond this use case, this is paving the way towards the certification of the safety-critical aeronautical products based on surrogate models. The way forward will be to complete the assurance activities on the implementation process and adjust the argumentation to the final guidance when the WG-114 standard (AS6983) is released.

[1]https://aniti.univ-toulouse.fr/en/

[2]https://www.deel.ai

References

[ACW18] Assurance Case Working Group ACWG. The goal structuring notation community standard version 2, 2018. Safety-Critical Systems Club, York, UK.

[ACW21] Assurance Case Working Group ACWG. The goal structuring notation community standard version 3, 2021. Safety-Critical Systems Club, York, UK.

[CPH20] Richard Hawkins Radu Calinescu Chiara Picardi, Colin Paterson and Ibrahim Habli. Argument patterns and processes for machine learning in safety-related systems. 2020. University of York, York, U.K.

[DCW21] IRT StExupery DEEL Certification Workgroup. White paper - machine learning in certified systems, 2021.

[DDGG+21] Mathieu Damour, Florence De Grancey, Christophe Gabreau, Adrien Gauffriau, Jean-Brice Ginestet, Alexandre Hervieu, Thomas Huraux, Claire Pagetti, Ludovic Ponsolle, and Arthur Clavière. How to certify a reduced footprint acas-xu system: A hybrid ml-based solution. In *International Conference on Computer Safety, Reliability, and Security (SAFECOMP)*, 2021.

[DPP20] Kevin Delmas, Claire Pagetti, and Thomas Polacsek. Patterns for certification standards. In *Proceedings of the 32nd International Conference on Advanced Information Systems Engineering (CAiSE'20)*, pages 417–432, 2020.

[EUR20a] EUROCAE / SAE. ER-022/AIR6988 - AI in Aeronautical Safety-Related Systems: Statement of Concerns, 2020.

[EUR20b] EUROCAE WG 75.1/RTCA SC-147. Minimum Operational Performance Standards For Airborne Collision Avoidance System Xu (ACAS Xu), 2020.

[EUR21] EUROCAE WG-114/SAE joint group. Certification/approval of aeronautical systems based on AI, 2021. on going standardization.

[FK21] Christophe Gabreau Baptiste Lefevre Fateh Kaakai, Béatrice Pesquet-Popescu. Ai for future skies: On-going standardization activities to build the next certification/approval framework for airborne and ground aeronautical products, 2021. AISafety 2021.

[HC09] C. Haddon-Cave. The nimrod review: an independent review into the broader issues surrounding the loss of the raf nimrod mr2 aircraft xv230 in afghanistan in 2006, 2009. report, vol. 1025. DERECHO INTERNACIONAL.

[HG18] Holloway and Graydon. Explicate '78: Assurance case applicability to digital systems, 2018. FAA report DOT/FAA/TC-17/67.

[KBD+17] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*, pages 97–117. Springer, 2017.

[KBMB97] Tim Kelly, Iain Bate, John McDermid, and Alan Burns. Building a preliminary safety case: An example from aerospace. In *1997 Australian Workshop on Industrial Experience with Safety Critical Systems and Software, Australian Computer Society, Sydney, Australia*, 1997.

[KHC12] Mykel Kochenderfer, Jessica Holland, and James Chryssanthacopoulos. Next generation airborne collision avoidance system. *Lincoln Laboratory Journal*, 19:17–33, 2012.

[KW04] Tim Kelly and Rob Weaver. The goal structuring notation /- a safety argument notation. In *Workshop on Assurance Cases*, 2004.

[Lev11] Nancy Leveson. The use of safety cases in certification and regulation, 2011. Aeronautics and Astronautics/Engineering Systems MIT.

[MSG21] Hampton Virginia Mallory S. Graydon, Jared D. Cronin Langley Research Center. Retrospectively documenting satisfaction of the overarching properties: An exploratory prototype, 2021.

[RHH21] Chiara Picardi Radu Calinescu Richard Hawkins, Colin Paterson and Ibrahim Habli. Guidance on the assurance of machine learning in autonomous systems - amlas. 2021. University of York, York, U.K.

[RTC00] RTCA/EUROCAE. DO-254/ED-80 - Design Assurance Guidance For Airborne Electronic Hardware, 2000.

[RTC11] RTCA/EUROCAE. DO-178C/ED-12C - Software Considerations in Airborne Systems and Equipment Certification, 2011.

[Rus15] John Rushby. The interpretation and evaluation of assurance cases. Technical report, 2015. Technical Report SRI-CSL-15-01 Computer Science Laboratory, SRI International, Menlo Park, CA, July 2015. URL="http://www.csl.sri.com/users/rushby/papers/sri-csl-15-1-assurancecases.pdf".

[SAE96] SAE. Aerospace Recommended Practices ARP4761- guidelines and methods for conducting the safety assessment process on civil airborne systems and equipment is an aerospace, 1996.

[SAE10] SAE/EUROCAE. Aerospace Recommended Practices ARP4754a/ed-79a- development of civil aircraft and systems, 2010.

[Tou03] Stephen Toulmin. The uses of argument, updated edition, 2003. Original edition 1958.

Do assurance standards need radical changes?

Emmanuel Ledinot(1) Bertrand Ricque(2) Franck Serratrice(3), Jean Gassino(4),
Rémy Astier(5), Philippe Baufreton(6), Jean-Louis Boulanger(7),
Cyrille Comar(8), Jean Claude Derrien(9) Joseph Machrouh(10), Philippe Quéré(11),

(1, 2, 3, 4, 5, 6, 7, 8, 9,10)
Working Group "Safety Standards" – Embedded France
(1): Contact author, THALES, emmanuel.ledinot@thalesgroup.com
(2) Safran; (3) : Renault (4): Institut de Radioprotection et de Sûreté Nucléaire;
(5): Framatome; (6) Safran; (7) CERTIFER;
(8): AdaCore; (9): SAFRAN (émérite), (10) : THALES, (11) : Stellantis.

Abstract

As Embedded France's cross-domain group dedicated to analysis and evolution of safety standards, we observe the emergence of new types of systems. It is noticeable in the automotive, railway, manufacturing, defence, and to lesser extent in aeronautics, space and nuclear domains. Some of these changes are likely to influence development assurance up to the point where an update of part of their principles becomes worth considering. Such an evolution has already started with the surge of new standards dedicated to cybersecurity for safety, machine learning, or advanced automation and autonomy. This paper reviews some aspects of these ongoing evolutions and argues that new standards or revisions of former ones might occur in the near future. The major assurance shifts we anticipate are the need to address new scales of complexity, the need to introduce on-line risk mitigation policies, and the need to revisit qualification of COTS and tools. We regard the automotive domain as emblematic of the confluence all the trends so we illustrate our claims and prospects primarily with examples from this industrial sector.

Keywords: complexity, technological trends, safety, risk analysis, assurance, assurance costs.

1. Introduction

Convolutional and Deep Neural Networks (CNNs, DNNs) have nearly revolutionized automated perception. For sure, there is still a bunch of difficulties regarding robustness and trustworthiness demonstrations in this field, but the performance quantum leap was such that in most industrial domains embedded systems have initiated a significant move. Perception, situational awareness, and complex real-time multi-criteria decision making that were formerly exclusively devoted to humans are now planned to be either shared between man and machine (man-machine teaming) or fully transferred to machine. This transfer of functions from human to machine is under way under stringent cost constraints whatever criticality level is at stake. This situation puts safety assurance under stress. Assurance costs depend in the first place on *complexity* and *criticality*.

Moreover, the extension of the Internet of people and IT services to the Internet of objects and OT[1] services has pervasive consequences on industrial systems. Wireless connectivity (WiFi, 4G/5G/6G), scalable computing and storage (cloud, edge, MEC, fog computing), sensor networks, and big data analytics have motivated the emergence of new *mass*-markets of IoT safety-sensitive products and services [5]. The "IT-OT continuum", also named "phygital continuum" where "phy" refers to the *phy*sical and to the *phy*siological, enables new classes of systems we consider in this paper besides the classical embedded systems mentioned previously.

Are current assurance standards, the foundations of which originate from the 60s, still appropriate to address this twofold shift: 1) higher complexity at *constant* cost? 2) *low-cost* safety-sensitive mass-markets?

Section 2 proposes some insights on how systems are evolving, with emphasis on the automotive domain that one can view now[2] somehow as a safety critical *IoT mass-market*. It features the two trends at the same time: new embedded system complexity and connectivity to the IT-OT continuum. The drone market is similar.

Section 3 explains why these evolutions are leading to engineering bottlenecks that have impact on assurance cost up to the point where, in our opinion, it will no longer scale. Section 4 gives some orientations to overcome that situation.

Our paper continues a series of publications from the Working Group, through which its members disseminate and encourage feedback about their

[1] OT Operational Technologies, IT Information Technologies

[2] Since 2018 connectivity of cars to wireless Internet is becoming mandatory

work [24-33]. All papers are available on Embedded France's website.

Note1: we consider *assurance* as some process involving planned and systematic actions that together provide confidence that errors or omissions in requirements or design have been identified and corrected to the degree that the system, as implemented, satisfies applicable certification requirements"[3]. We also use *development assurance* as assurance applied to requirement capture, specification, design, architecture, implementation, verification and validation[4].

Note2: safety is our main concern. However, many assurance issues we discuss apply as well to cybersecurity and mission reliability.

2. Evolution of safety critical systems

We start by reviewing system evolution in car industry to substantiate why development assurance is caught on one side by skyrocketing of costs, and on the other side by cost containment, if not reduction. Then we review some other evolutions applicable to all industrial domains.

2.1. The automotive case

Today in automotive market, regulation (GSR II) and final customers request new and more complex functions. Tough competition leads to frequent addition of new functionalities and constant evolution of software configurations on:

- Autonomous driving functions (at different levels),
- Phone Inductive chargers,
- Entertainment functions,
- Connectivity within vehicle (to update GPS map, Apps…, it is estimated that 100% of the new cars will be connected to the Internet by 2025).
- Possibility to download Apps…

Regulation authorities request more and more:

- Cybersecurity protection,
- Pedestrian protection,
- Engine emission control,
- New crash requirements,
- Scene recorders,
- Emergency call management.

As a result, vehicle definition becomes more complex every year and generates systematic development cost increase. SW development costs

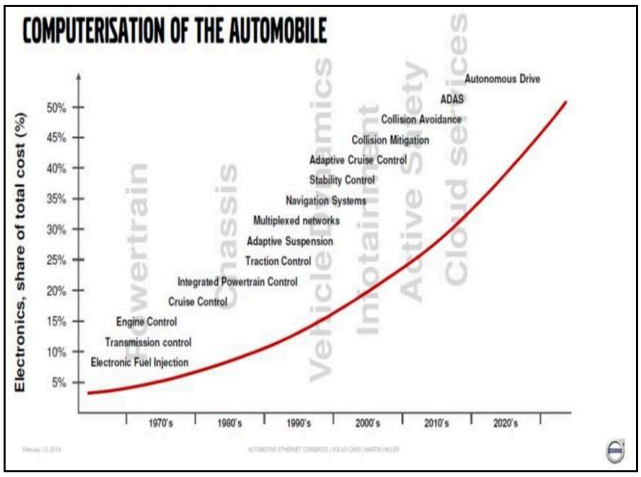


Figure 1: Vetronics relative value 1960-2020

are estimated at 40% of the vehicle cost. Customers regard safety and security as due, in other words they are not willing to pay extra-cost for it.

The safety and software quality assurance standards in practice in the field are:

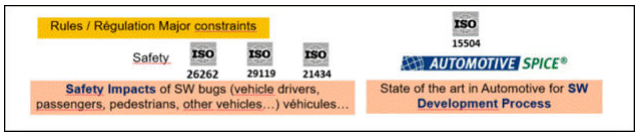


Figure 2: Applicable standards

Besides digital transformation and future self-driving cars, present hybrid and electrical vehicles are other domains of system innovation with safety engineering challenges (e.g. battery management systems, electrical power control, power electronics, etc.).

2.2. Safety-critical low-cost

We deem cars are the pivotal example that lies at the confluence of all trends. Cars are mass-market products featuring all criticality levels and intense competition between players. The system enhancements listed above must be compatible with strict cost-killing policies, development assurance inclusive. Assurance consists in careful description and oversight of all unit development activities and of their mutual relationships, the number of which grows *more than linearly* with project complexity. How to ensure assurance cost containment *with* complexity growth?

The situation is not specific to automotive even if ADAS and self-driving cars exacerbate this tension. It is also the case for drones and more generally for robotics. Manufacturing robots and co-bots of industry 4.0, for instance, are now capable of changing their location in workshops. They come close to, if not in contact with, humans. They are becoming more safety critical while indulging no

[3] This definition is borrowed from "Aircraft system safety – Assessments for Initial Airworthiness Certification.
[4] These terms being defined by ISO15288

extra assurance costs. A major risk management shift is underway in this field: the move from external protection functions to *integrated safety* control.

Emergence of safety-sensitive IoT mass-market creates new classes of incidents for which no regulation, as of writing this paper, defines their severity and societal acceptability. Consider for instance Orange's [17] and Facebook's [18] recent IT service outage events. In case of [17], the prosecutors claim six fatalities have been partly caused by the failure of about 15,000 emergency calls during the outage period. Do we know the true safety impact in Facebook's case [18]? Let us assume that during the 6-hour blackout only 1 in 10,000 subscribers needed access to their account to get some information *required* by some urgency. Since there are 2 billion Facebook subscribers, the IT glitch potentially exposed *200,000 people* to safety-sensitive conditions. How many in the end led to major, hazardous, or even fatal outcomes? Is Facebook's liability engaged?

Digital transformation of society creates new dependency networks, new responsibility chains possibly leading to a significant number of incidents and accidents. Who is in charge of analyzing these new hazards? What level of development rigor is appropriate for safety-sensitive IT and IT-OT services? To our knowledge there is no answer yet, nor work in progress though definite answers are needed.

2.3. Cost-killing policies

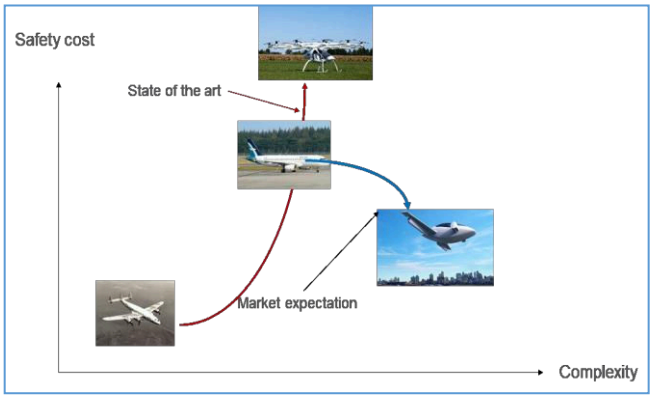


Figure 2 – Cost-Complexity-Market tension

Emergence of *mass*-markets of low-cost moderately critical products or services is new. The association of low-cost and safety-criticality challenges the applicability of present assurance standards. They have ever been elaborated with affordability in mind, but "no price for safety" was also in the mindset. Low-cost and "no price for safety" are harder to conciliate.

Even on traditional safety critical products where high costs have always been considered as *inevitable* and *acceptable*, value-for-money of development assurance is challenged, at least in US aeronautics. It started by 2012 for general aviation i.e. for 1 to 4-seat aircraft. Regulation and safety assurance standards[5] were drastically alleviated to reduce assurance costs. Extending this reformation to transportation aircraft[6] was researched from 2015 to 2018, without any enforced outcome. Figure 2 is notional and suggests what dilemma assurance faces today: ever more complex systems, *not* at ever-higher prices. Safety is due *without* premium.

Hype projects like autonomous flying cabs are not necessary to illustrate the risks of downscaling development assurance. Consider iterative updates of an old aircraft, with on-purpose successive applications of a component-wise approach to safety assessment: "no need to revisit such and such components or subsystems since they were certified in the past and are not concerned by the change of interest". Consider engine change to reduce fuel consumption, with some longitudinal instability side-effect over parts of the flight domain. Consider compensating the pitch-up effect by a new automatic trim function. Consider undue schedule and cost pressure leading to silenced safety issues and breaches of basic architectural rules and conformity to assurance standards. These very standards that have led civilian aviation from 2008 to 2018 to the best ever safety level of 12.2 fatalities per billion passengers. Consider Authority delegation used by the applicant beyond reasonable limits to meet schedules and reduce development costs. Outcome? 346 casualties, $20-billion loss for the aircraft manufacturer; [7] and [19] give a documented 2016-2020 industrial case that illustrates where low-cost policies and compromised safety assessment can head for.

2.4. Unforeseeable operating conditions

We address now a more technical issue with potential deep impact on assurance principles. All standards assume that risk analysis is performed at *design-time* and has to be *complete*. Completeness of top level feared events identification and causal modelling are twofold needed. First, to prevent from quantitative underestimation of the risks. Second, to design the mitigation policies i.e. the monitoring logics, the degraded modes and the FDIR mechanisms. Anticipating *all* the hazardous external conditions was (nearly) possible for integrated-safety systems because the foreseeable adverse conditions were anticipated at design-time, and primarily managed by humans at operation-time. Crisis management was *outside* of the system. If

[5] Accepted Means of Compliance, i.e. the best engineering practices recognized as state of the art by applicants and Authorities

[6] 100+ passengers

inside as it turns out to become the case now, new monitors and decision logics that operate at cognitive level should be demonstrated “complete”.

The members of our group involved in automotive, drone and defense sectors face this new situation. They argue that wherever AI-based machine perception and mission supervision are concerned, the off-line predetermined reactive approach will no longer be effective. One needs to resort to *on-line* and *proactive* mechanisms that do not rely on a predefined classification of situations, with pre-assessed gravities and pre-defined mitigation policies. One needs (a lot of) additional embedded software to manage any unanticipated conditions.

There are no standards to assure *dynamic* hazard analysis and on-line mitigation decision-making. There are no standards to assess such on-line approaches. This is the reason why in the automotive domain, in addition to ISO 26262 and SOTIF, four new standardization committees [12], [13], [14], [15] have been created *this year* to address emergency calls, safety and Artificial Intelligence, preexisting SW product reuse for safety related applications, and intelligent transport systems.

There is an analogous situation for drones in urban conditions w.r.t. SORA and JARUS [20]. Current standards ban the urban drone operations that *assurance does not know how to address* (e.g. vision-based urban emergency landing in GPS-denied conditions).

2.5. Virtualized architectures

There is another dynamicity case with deep impact on assurance: *software-defined architectures*. It started a few decades ago with virtual machines and operating system emulation. Since 2010 IT-OT convergence promotes the use of cloud-native technologies for the embedded (e.g. Pods, containers, K3S, etc.). It introduces virtualized computing, networking, and storage into medium criticality soft real-time information-dominant systems (e.g. supervision, SCADA).

Service Level Agreement (SLA) and best effort policy appear besides hard real-time worst-case policy in *mixed-criticality* infrastructures. Defense systems, transportation and energy infrastructures, as well as industry 4.0, are evolving fast in that direction. Time-Sensitive Networking (TSN) and Software Defined Networking (SDN) are enablers. Past physical segregations are shifting to *software-enforced segregations*, which augment dependencies and couplings.

Software-defined architectures (infrastructure as program) have been motivated by mass-deployment of IoT end-points, ad-hoc network wireless connectivity, and resource load balancing in cloud data centers (e.g. “green IT”). How to justify safety of such *dynamic* architectures? Orchestrators periodically update their logical to physical mappings... In some sense, this mapping to physical resources becomes *implicit*. From the traditional safety assessment perspective, a consequence is that FTA and MBSA[7] will have to model architectures that compute their own structure.

Such self-configuring architectures, akin to self-organizing systems in biology, support *formation control*, a type of distributed control needed for *collaborative mission* systems (e.g. swarms, satellite constellations, platoons of cars or trucks, collaborative combat systems, etc.). Behavioral analysis of such systems is quite hard, at the forefront of research, and beyond present assurance foundational concepts.

2.6. Cyber-security

A consequence of open-world conditions, generalized connectivity and software-defined architectures is the absolute necessity of *pervasive* cyber-protection. Zero trust policies are being enforced and orchestrated with the virtualized resources (DevSecOps). Behavioural complexity and assurance costs are highly impacted by these additional protection functions.

2.7. Ever rising Open Source Software (OSS)

Open source’s role keeps growing in complex embedded systems and CPS/CPSoS development. It is a major affordability enabler for these new complexity classes. Originally, it appeared for software. The emblematic examples are Unix and Linux. Linux has now spread in 5G infrastructures and in IoT (e.g. Yocto). Open Source has also reached hardware (e.g. ARM, Open Hardware group, RISC-V, etc.), AI and Big Data analytics (e.g. TensorFlow, Scikit Learn, PyTorch etc.), system layers (e.g. K8S, OpenStack, OpenFlow, OpenvSwitch, etc.), and all domains of digital engineering.

So what w.r.t. development assurance? Assuring software or hardware COTS is uneasy. Safety is context-*sensitive*; COTS pre-qualification is context-*independent*. Assurance is a fault prevention process at *development*-time, which by definition does not apply to *pre-existing* components**.** As in-house development is receding and integration-based development is becoming dominant, finding efficient means to assure open source components is of critical importance. The trend towards smarter mission supervision (cf. §1, §2.1, §2.4) pushes many open source *tools* into embedded applicative layers. COTS pre-qualification formerly limited to OS, RTOS and libraries is soaring in all domains.

[7] Model-Based Safety Assessment, failure propagation modelling.

2.8. AI-enabled functions

There is currently a worldwide effort to define machine-learning assurance standards. We do not address this important issue in our paper but we cannot afford skipping over it silently since it is one major concern that impacts assurance standards. Example-based specifications, sampling-dependent deterministic behaviours, black-box requirement-to-code, adversarial counterexamples in high dimensional approximation spaces (video/image classifiers, audio and natural language processing, etc.), are but a few of the assurance challenges posed by this specific engineering domain the importance of which has been stressed in §1, §2.1, and §2.4.

2.9. New types of responsibilities

Autonomy and enhanced automation are game changers for the legal. Scapegoating rationales will evolve. Assurance requirements are getting *higher* for autonomy, as already noticed through the growing use formal methods. More rigor is called for unmanned vehicles than for manned ones.

2.10. Wrap-up

Depending on whether you are an optimistic or pessimistic person, system safety engineering and assurance are facing a situation that one might consider as exciting, or panicking.

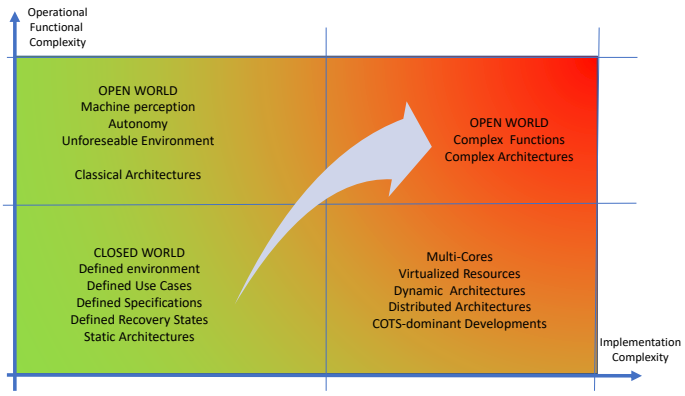


Figure 3: Soaring of assurance with twofold complexity increase

Except for the most safety-critical sanctuaries (e.g. nuclear protection control command, airplane flight control systems, car ABS or ESP, etc.), most of the safety best practices are being dismissed. Examples are “keep it small and simple”, “keep it statically defined”, “keep it closed”, “keep it deterministic”, “keep it worst-case guaranteed”, “minimize attack surface”, and last but not least: “no price for safety”.

Breaching conformity to the safety assurance standards for a product that featured *none* of the new complexity factors we have surveyed in this section led to the major crisis accounted in [7] and [19]. Caution might be needed when considering development assurance as an impediment to financial performance.

3. Some engineering and assurance limits

We proceed in reviewing the factors that put assurance under stress and motivate its evolution. Section 2 has addressed the product side; we now turn to the process side: the weaker engineering processes, the more compensations by assurance processes. First, we point to some engineering capabilities that are missing and that, if available, would enable lower assurance effort. Then, still on the process side, we review some assurance weaknesses that lead also to compensatory redundancy in assurance activities.

Regulation defines the acceptable risk levels that condition the definition of assurance goals and accepted means of compliance. Without regulatory severity matrices one does not know how to scale the risk mitigation policies: fault tolerance architectures and development assurance levels. As indicated in §2.2 and §2.9, some new domains of system engineering are facing this situation. This is a second example of mutual influence between assurance requirements and engineering practices.

They are mutually dependent indeed. The lower trust on engineering quality the more compensatory assurance activities. We now hint at a few technical issues that would help reducing assurance were engineering stronger. More information is available in [33] and [3].

3.1. Some limits of specification and verification technologies

In this section, we exclude engineering aspects related to validation (*Intent* in the Overarching Properties setting [21]). This choice is by no means a matter of priority. For space limitation reasons we concentrate on *syntactic* engineering (text, model, program, data). Validity assurance relies mainly on *semantic* skills: human knowledge of operational conditions and “business logic”. We also limit ourselves to *high-criticality* developments resorting to *model-based* engineering, i.e. to situations where applying assurance standards is the most expensive (e.g. DAL A 100% overhead incurred w.r.t no DAL), requires use of the most advanced engineering techniques.

3.1.1. Contract-based development

In essence, contract-based development could be the backbone of correctness assurance, especially when using assurance-cases [21]. Formal contract-based development is an industrial reality in hardware engineering. To a lesser extent it is also an industrial reality in safety critical software engineering. At system level however, where one

needs it as well if not more, it is unfortunately absent of industrial practice. It is still at (early) research stage [23].

3.1.2. « Verify-while-develop » methods

We prefer this name to the more familiar "correct-by-construction". "verify-while-develop" leaves open the possibility of *flawed specifications*. For laypersons, "correct" implicitly encompasses verification *and validation*. B-method, refinement calculus, or qualified automatic code generators are examples of correctness preserving refinement. Sensitivity of assurance costs to engineering quality resides mainly at specification, implementation, and verification stages.

Formal refinement would be an advanced way of ensuring that implementation is *derived from* and *compliant with* the requirements. These two properties are among the main and most expensive assurance goals. If model-based formal refinement were industrially mature, it could be an enabler of lighter assurance compared to the many redundancies in compliance verification (e.g. compliance of EOC with HLR[8] in DO-178C DAL A). Unfortunately, apart from the noticeable B-method exception, there is no industrial support of specification, model, and code refinement. Successive steps of text copy-paste with incremental additions verified by reviews is the most common low-tech practice.

3.1.3. Verification coverage analysis

Sufficiency of verification coverage, i.e. IVVQ termination criteria, is one of the most critical issues of verification of verification (i.e. assurance on verification).

For software and hardware engineering, nearly all standards have chosen structural coverage analysis as measurement of behavioral space exploration. These measures enable definition of precise IVVQ termination criteria. What termination criteria for *system* verification? In particular, when the new high-level cognitive supervision or perception functions mentioned in §2.1 and §2.4 come into play?

System engineering stumbles on finding means to measure exploration of complex behavioral spaces and to justify exploration *sufficiency*. This generates *mistrust* feelings, which in turn generates costly "overlays" of compensatory measures at engineering and assurance levels.

The situation is analogous for qualification of complex Open Source Software (OSS) components, and for qualification of sophisticated engineering tools. For a COTS for instance, how to justify sufficiency of behavioral exploration to demonstrate fit for purpose in the new context?

3.1.4. Synopsis

More powerful engineering would favor reduction of assurance overhead. Ultimately, this overhead could reduce to process accountability[9], supplemented with quality oversight focused on appropriateness of application of powerful methods supported by qualified tools. Unfortunately, engineering state of the art is incompatible with such lightweight sufficient assurance. Alleviation of assurance overhead and progress of engineering are intermingled as witnessed in current standards by *verification credits*. When development or verification steps are trustworthy[10], some upstream or downstream redundancies of assurance activities may be suppressed.

3.2. Assurance limits

Assurance is criticized because too often it boils down to hundreds of descriptive documents whose value for money and impact on product quality is matter of debate. There is some legitimacy in questioning the cost-benefit ratio of some specific aspects of development assurance[11]. We point to two potential sources of "paper work" pain points.

3.2.1. Reductionism to « error-free» simplicity

Since development-fault detection is possibly incomplete, the risk mitigation policy is to *add* fault prevention assurance activities. Explicit engineering risk analysis[12] should drive the definition of the assurance activities crafted to mitigate the identified risks. However, as we have noticed in our group, standards are not explicitly *engineering-risk*-based. They prescribe process constraints supposed to be solutions to problems that are most often left implicit (if not silenced).

Part of the engineering risks are addressed by *company*-dependent engineering guidelines, not by assurance standards. Fault prevention is mostly ensured by engineering rules (e.g. design patterns, complexity limits, etc.) that are audited but kept confidential as competitive advantages.

Fault prevention is also ensured by decomposing the engineering processes into *elementary* activities that are *simple* enough to be verifiably error-free. Error prevention is enforced by detailed documentation of these elementary activities (a priori development

[8] EOC: Executable Object Code, HLR: High Level Requirements. Eight assurance goals and corresponding verification activities contribute the compliance demonstration.

[9] Quality assurance plans describing the engineering processes

[10] A typical example is tool qualification with high TQLs (Tool Qualification Levels).

[11] Up to the limits formulated in §2.3.

[12] Some sort of "EHA" for Engineering Hazard Analysis that would be the counterpart at process-level of FHA (Functional Hazard Analysis) at product-level.

plans), and by verification of work conformity to the plans (a posteriori audits).

This rationale apportions assurance effort to engineering effort, complexity and criticality. It is the main reason for assurance cost blow-up on the new system complexity classes described in section 2. It is the origin of our feeling of non-scalability, our motivation to address the issue of finding orientations for "radical changes". Beyond the incompressible process accountability baseline, we deem systematic documented work reductionism not scalable nor safety-value for money. In addition, it intrinsically does not fit COTS-intensive integration-based development, which is becoming dominant.

3.2.2. Verification redundancy

Redundancy of activities addressing the same assurance goal (e.g. compliance of executable object code with high-level requirements) is rightfully used to mitigate the risks of partial achievement. The redundancy degree depends on the Development Assurance Level (DAL). Do multiple weak verifications constitute a strong verification? Current approach of Authorities is dominantly to favor verification innovations (e.g. formal methods) *if* they are *added* to current redundancy schemes. To *substitute* a verification step by another one is much harder to get consensus on. Most often, it leads to tool qualification requirements that constitute a barrier because of the high cost of current tool qualification methods [16].

4. What to evolve in assurance standards?

Section 2 has addressed a few new system complexity classes, safety impacts and engineering challenges that in turn challenge current assurance. Section 3 has advocated that verification technologies and core assurance principles are presently too weak to limit assurance to a scalable minimum.

Process-based assurance enforces engineering-fault prevention and engineering-fault detection principles that we have named *reductionism to simplicity* and *verification redundancy*[13]. This section proposes a few orientations to revise these principles. We have grouped these revision proposals in two subsections: the former is dedicated to foundational and disruptive (i.e. "radical") ones, while the latter sticks to present principles but looks after revised balances.

4.1. Matter of principle

4.1.1. Paper work assurance

We challenge systematic fine-grained detailed paper work[14] as effective fault-prevention means for the technical domains reviewed in section 2. It does not scale nor fit with the new AI-based applicative layers and the COTS-dominant virtualized architectures. Assurance overhead grows more than linearly with project complexity because oversight activities look after individual *and mutual* artefact consistency. Moreover, on past projects the high costs of baseline assurance documentation have been occasionally detrimental to the introduction of new powerful development technologies.

Excess of documented reductionism does not apply only to authoring activities (requirements, specifications, models or codes), it may also apply to verification, especially to unit testing. There is an issue about cost-effectiveness and safety-effectiveness of the assurance goals dedicated to verification of testing, and more specifically to fine-grained testing coverage analysis and termination criteria.

4.1.2. Assurance of foreseeably unforeseeable conditions

Present safety standards assume feasibility of "exhaustive" anticipation of the product's future adverse operating conditions. They assume applicability of conservative *worst-case* mitigation rationales that shape fault-tolerance design and safety assessment processes.

Complexity of machine perception, interpretation and decision-making in open environments (e.g. self-driving cars, drones, or any type of autonomous robots) necessitates the introduction of a new risk mitigation paradigm *and* associated assurance: *adaptable proactivity* at supervision and resource levels. This paradigm would resort to complex *on-line* scenery analysis, to anticipation based on model-predictive control, and possibly to collaboration with the *environment* of the system to mitigate the identified risk. It would introduce estimation of potential incident and accident *severities at run-time*. It would require on-line assessment of the available risk mitigation capabilities, and on-line selection of a mitigation strategy that possibly would rely on new *benefit/risk* logics.

[13] We are aware of the exceedingly simplistic character of this statement, up to inappropriateness regarding the invaluable benefits of the assurance requirements related to change control and configuration management.

[14] We exclude from this point the company-owned guidelines that are part of assurance documentation (engineering rules and design best practices). We point to meticulous extensive descriptions of the engineering activities regarded as fault prevention means, especially in the specification and implementation refinement processes.

All these aspects are disruptive w.r.t. present safety standards. Best-effort on-line risk management does no longer claim feasibility of complete identification of the failure cases. Introducing such a paradigm would not be compatible with current risk quantification methods since they rely on dysfunctional modeling under *completeness* assumption w.r.t. to the set of failure cases[15]. In other words, the *inverse* causality dysfunctional modelling that we questioned in [33] and [3] would no longer operate *at all* with this dynamic approach because the causal scenarios leading to occurrence of the feared events would remain *implicit,* i.e. estimated at run-time.

In addition, such a dynamic approach would require onboarding a significant number of engineering models and *tools*, with the consequent tool qualification issues at high TQL[16].

4.1.3. Assurance of virtualized architectures

IT-OT convergence influences embedded system development. IT is used to benefit OT when massive cyber-secure deployment (DevSecOps), data analytics (AI, sensor networks), 4G/5G connectivity or interaction with infrastructures are at stake. IT optimized availability and scalability KPIs. IT dismissed peak guarantees in favor of *best-effort* policies. This design rationale is incompatible with OT safety motivated *worst-case* guarantees. There is a need for future *assured* mixed criticality over the IT-OT continuum. This implies adapting the assurance goals to mixed guarantee regimes.

4.1.4. Assurance of machine learning

The process is under way (e.g. [22]). AI-ML implies disruption in assurance for a bunch of reasons. As an example, statistical estimation of a program introduces *randomly conditioned* deterministic software in a field where predictability, correctness and development rigor were the only eligible concepts for assurance [35].

4.1.5. Assurance of continuous change

Continuous Integration and Continuous Deployment (CICD) is an IT technological trend that is pervading OT very quickly. Certification today is synonymous of “frozen configurations”, or low pace evolution. Cost of ensuring conformity to assurance standards constrains to sparse releases. Conversely, cybersecurity, open learning and mass deployment of IoT updates call for CICD. There is a foundational assurance issue to investigate there: how to conciliate high integrity cost-effective assured deployment with nearly continuous deployment?

[15] Minimal cut sets of fault-trees or minimal sequences of Markov chains, stochastic Petri nets, etc.

[16] Tool Qualification Level

4.1.6. High profile assurance

Assurance standards cannot assign goals and activities that would be intractable because of shortage of supporting tools and trained professionals. However, some recent concepts that have emerged from computer science and control theory (e.g. contracts, abstraction/refinement, invariants, influence cones, etc.) would be beneficial to precise formulation of assurance requirements, even in absence of mature tooled support. They could be beneficial as assurance ontologies, as reference concepts. Some industrial domains (e.g. aeronautic, space, automotive) have banned means prescriptiveness in standards, and concept prescriptiveness altogether[17]. We have advocated in §3 the existence of a link between assurance overhead and engineering strength. Distinguishing between means and concept prescriptiveness, and accepting to prescribe *concepts* fundamental for writing assurance plans would benefit the perceived value of assurance and facilitate the introduction in processes of more advanced development technologies.

4.1.7. Assurance for the new classes of severities

Section. 2.9 explained why the phygital continuum may lead to mass incidents or accidents. New assurance standards must ensure that accountability and responsibility identification remain feasible for these new classes of potential mass-damages

4.2. Matter of balance

4.2.1. Manual .vs. Automated

For a long time, man has been the only actor capable of deciding correctness. Today, most of the engineering artefacts and associated verifications are so complex that manual verification is no longer trustworthy. Suspicion of bugged tools and fear of flawed mechanized verification are legitimate. However, most standards tend to be more suspicious of tool errors than of human errors, even on activities that are far beyond tractability by man intellectual power.

Tool qualification is a necessity. However, more often than not its implementation is so expensive that tool qualification is abandoned in favor of manual verification. Overestimated trustworthiness of man w.r.t. machine is preferred e.g. code review .vs. abstract interpretation for detection of run-time errors in source code.

[17] The few ones commonly used are accuracy, consistency, validity, correctness, compliance, structural coverage and completeness. Glossary definitions of standards are often vague. 50 years of research has come out with many other concepts like abstraction/refinement, simulation relations, observational inclusions, observational abstractions, behavioral equivalence, reachable state space, etc. that would help writing higher quality assurance standards.

It happens more often than not that powerful tools are not used by applicants, or used but kept hidden because declaring them in the plans would lead to excessive assurance complications. Some tuning is needed to avoid these situations that are counterproductive for innovation and for safety.

4.2.2. In-house .vs. Out-sourced

As development projects are getting more and more complex and integrative, specification-based development is receding w.r.t. integration-based development. COTS reuse in a new context, framework instantiation, system and software product-line engineering are becoming the new mainstream approaches to development. They poorly fit with current assurance rationales. They were conceived for developments in which COTS reuse was more the exception than the rule. Adaptation of assurance is needed to go beyond service history and tool qualification approaches.

5. Conclusion

We have reviewed some trends of system markets and system safety engineering that we deem will have mid-term impact on assurance standards: from automation to autonomy, form specification-based dominance to integration-based dominance, from closed-world to open-world, and from static architectures to dynamic architectures. All these trends lead to far higher complexity that mechanically in turn lead to far higher assurance costs while competition would require to keep them constant, if not decreasing.

Then we have reviewed some aspects of assurance, whose cost/benefit ratio are perceived as falling below threshold. We have suggested some high-priority issues to address to remedy this situation. System engineering state of the art has to improve a lot to enable lightweight assurance. The short-term view is mere old engineering wisdom: scaling complexity with mastery.

6. References

[1] A. Benveniste B. Caillaud & al., “Contracts for System Design” Foundations and Trends in Electronic Design Automation Now (2018).

[2] Nancy G. Leveson, *Engineering a Safer World: Systems Thinking Applied to Safety* (2011). MIT Press. Leveson, N.G., Thomas, J.P. STPA Handbook (2018). https://psas.scripts.mit.edu/home/get_file.php?name=STPA_handbook.pdf.

[3] Emmanuel Ledinot, *CPS Engineering: Gap Analysis and Perspectives* (2021). CoRR abs/2104.13210.

[4] Gabriel, N., Holz, E. *SOTIF and FuSa STPA for a Highway Pilot Function of a Passenger Car*. European STAMP Workshop and Conference (ESWC, 2020).

[5] Marc Duranton, Michael Malms, Marcin Ostasz *The continuum of computing,* in HiPEAC Vision 2021 High Performance Embedded Architecture and Compilation.

[6] *EASA Concept Paper : first usable guidance for level 1 machine learning applications*. Issue 01 April 2021. https://www.easa_concept_paper_first_usable_guidance_for_level_1_machine_learning_applications_-_proposed_issue_01_1.pdf

[7] Maria Cantwell, *Aviation Safety Whistleblower Report – US Senate Committee on Commerce, Science and Transportion*. December 2021.

[8] ERDF, *Smart Grids la nécessaire mutation du réseau électrique*, https://youtu.be/Qbt7H3S_tlE juillet 2015 (consulté déc. 2021)

[9] ISO 26262-6: 2018 : *Road vehicles — Functional safety — Part 6: Product development at the software level.* https://www.iso.org/fr/standard/68388.html

[10] ISO/PAS 21448: 2019 : *Road vehicles — Safety of the intended functionality* https://www.iso.org/fr/standard/70939.html

[11] ISO/AWI TS 4654 *Road vehicles — Advanced Automatic Collision Notification (AACN) systems — Algorithm and parameters for injury level prediction* https://www.iso.org/standard/80215.html

[12] ISO/AWI TS 5083: *Road vehicles — Safety for automated driving systems — Design, verification and validation* https://www.iso.org/standard/81920.html

[13] ISO/AWI PAS 8800: *Road Vehicles — Safety and artificial intelligence* https://www.iso.org/standard/83303.html

[14] SO/AWI PAS 8926 : *Road vehicles — Functional safety — Qualification of pre-existing software products for safety-related applications*.
https://www.iso.org/standard/83346.html

[15] ISO/TC 204 : *Intelligent transport systems*. https://www.iso.org/committee/54706.html

[16] Frédéric Pothon, *DO-330/ED-215 Benefits of the new Tool Qualification*, October 2012

[17] La panne des numéros d’urgence causée par un « bug » logiciel, selon l’enquête interne d’Orange (lemonde.fr), juin 2021

[18] Panne géante : Facebook, Messenger, Instagram et WhatsApp à l'arrêt pendant plusieurs heures | LCI, octobre 2021

[19] *Final Committee Report – The design, development & certification of the Boeing 737 Max*, September 2020

[20] J. Guerin, K. Delmas, J. Guiochet *Certifying Emergency Landing for Safe Urban UAV* arXiv2104.14928v1 30 avril 2021.

[21] J. Chelini &al. *Avionics Certification: Back to Fundamentals with Overarching Properties*. HAL Id: hal-02156109. 14 juin 2019.

[22] EASA *Concept Paper: First usable guidance for Level 1 machine learning applications*. April 2021

[23] A. Benveniste B. Caillaud & al., *Contracts for System Design* Foundations and Trends in Electronic Design Automation Now (2018).

[24] P. Baufreton, JP. Blanquart, JL. Boulanger, H. Delseny, JC. Derrien, J. Gassino, G. Ladier, E. Ledinot, M. Leeman, J. Machrouh, P. Quéré, B. Ricque, "Multi-domain comparison of safety standards", ERTS-2010, Toulouse, France, May 19-21 2010.

[25] JP. Blanquart, JM. Astruc, P. Baufreton, JL. Boulanger, H. Delseny, J. Gassino, G. Ladier, E. Ledinot, M. Leeman, J. Machrouh, P. Quéré, B. Ricque, "Criticality categories across safety standards in different domains", ERTS-2012, Toulouse, France, 1-3 February 1-3 2012.

[26] E. Ledinot, J. Gassino, JP. Blanquart(, JL. Boulanger, P. Quéré, B. Ricque "A cross-domain comparison of software development assurance", ERTS-2012, Toulouse, France, February 1-3 2012.

[27] J. Machrouh, JP. Blanquart, P. Baufreton, JL. Boulanger, H. Delseny, J. Gassino, G. Ladier, E. Ledinot, M. Leeman, JM. Astruc, P. Quéré, B. Ricque, "Cross domain comparison of System Assurance", ERTS-2012, Toulouse, France, February 1-3 2012.

[28] E. Ledinot, JP. Blanquart, Ph. Baufreton, C. Comar, J. Gassino, H. Delseny, "Joint use of static and dynamic software verification techniques: a cross-domain view in safety critical system industries", ERTS-2014, Toulouse, France, February 5-7 2014.

[29] E. Ledinot, J. Gassino, JP. Blanquart "Perspectives on Probabilistic Assessment of Systems and Software", ERTS-2016, Toulouse, France, January 27-29 2016.

[30] JP. Blanquart E. Ledinot, J. Gassino, "Software Safety Assessment and Probabilities", DSN 2016, Toulouse, France, June 28 – July 1 2016.

[31] JP. Blanquart E. Ledinot, J. Gassino, "Software Safety: A Journey across domains and safety standards", ERTS-2018, Toulouse, France, January 31 – February 2, 2018.

[32] B. Ricque, J.P. Blanquart, J. Gassino, "A cross-domain comparison of systematic errors control strategies", Lambda-Mu-2018, Reims, France, October 16-18 2018.

[33] E. Ledinot, J.P. Blanquart, J. Gassino; "Towards Rebalancing Safety Design, Assessment and Assurance" ERTS-2020, Toulouse, France, January 29-31 2020. Hal-02442445f.

[34] "Guidelines for Development of Civil Aircraft and Systems", EUROCAE ED-79A and SAE Aerospace Recommended Practice ARP 4754A, 21/12/2010.

[35] "Guidelines and methods for conducting the safety assessment process on civil airborne systems and equipment", EUROCAE ED135 and SAE Aerospace Recommended Practice ARP 4761, 12/1996.

[36] Wilkinson C. "Integration of complex digitally intensive systems" – FAA Streamlining Assurance Processes Workshop – Dallas, September 13-15, 2016.

Session Th.5.A

Monitoring

Thursday 2nd June

17:00

–

Amphithéâtre

Multilayer Monitoring for Real-Time Applications

Etienne Hamelin*, Mihail Asavoae*, Selma Azaiez*, Alexandre Berne*, Cyril Faure*, Kods Trabelsi*
*Université Paris-Saclay, CEA, List, F-91120, Palaiseau, France (email: firstname.lastname@cea.fr)

***Abstract*—Validation of timing requirements of multicore, heterogenous and distributed systems is difficult problem because of a large number of situations that introduce temporal variability and/or interference. A possible solution is to augment the system with monitors and to rely on runtime monitoring techniques. In this paper we propose such a runtime monitoring which spans all the semantics layers of a model-based design, from high-level specification to executable code. We showcase our runtime monitoring on a safety-critical application for driving assistance.**

I. Introduction

Runtime monitoring is a lightweight verification technique for detecting property violations in embedded safety-critical systems. A typical runtime verification workflow consists of the definition, the synthesis and the execution of a collection of monitors which observe concrete executions of the system and check their conformance with respect to a set of stipulated properties. The observation aspect requires either a form of instrumentation (e.g. the code is made available) or external annotations (e.g. only black-box / COTS components are provided). The conformance checking aspect usually refers to property violations, or stated differently, to deviations from expected correct runtime behaviors. The design and implementation of a runtime monitoring environment needs to address these aspects as well as the characteristics of the embedded safety-critical system under consideration. In the following we propose a versatile runtime monitoring, which we denote as *multilayer runtime monitoring* and which combines high-level design details with low-level implementation particularities (of both application and execution platform).

We consider a model-based design (MBD) methodology for embedded software, adapted to address complex designs and to respond to complex and precise requirements. A prime example of such a methodology is based on the synchronous dataflow principle, with well-established workflows from Lustre/SCADE [8] or Simulink [9]. A MBD exposes several programming languages: the (just enumerated) high-level languages, intermediate languages like C and finally, low-level languages (e.g. assembly or binary code). In this case a multilayer runtime monitoring mimics the MBD languages, with monitors being placed at each level and an existing simulation/execution environment, likewise available. We also consider a distributed execution platform, which is suitable to map mixed-critical applications.

This research was partially supported by the ECSEL-JU under the program ECSEL-Innovation Actions-2018 (ECSEL-IA) for research project CPS4EU (ID-826276) in the area Cyber-Physical Systems.

In this paper we exemplify our multilayer runtime monitoring, driven by the non-functional requirements of a safety-critical application from the automotive domain. More specifically, in this paper we address timing properties. The MBD workflow starts with a high-level Polygraph design [11], a specialized dataflow-like language, which is further compiled and deployed on a distributed architecture. In this setting, the design of a multilayer runtime monitoring needs to address a certain number of challenges, presented in Section II. Moreover, details of the underlying MBD are in Section III, a high-level presentation of the multilayer runtime monitoring, in Section IV and the proposed case study, in Section V.

II. Challenges

One of the main challenges in designing and analyzing safety-critical applications running on complex computational systems (e.g. heterogeneous, distributed or even virtualized [15]) lies in the computation and communication timing variability of the application's components. This variability is due to various factors, among which we enumerate:

- task-specific behavior:
  - data-dependent (e.g. a task may chose a different behavior mode)
- inter-task interference due to shared software resources:
  - operating system-managed resources (e.g. scheduling, peripherals etc.),
  - exclusive services (e.g. drivers, communication protocols stacks, synchronization primitives, etc.)
- inter-task interference due to shared hardware resources:
  - shared CPU cores, shared caches etc.,
  - shared communication buses, shared peripherals, etc.

Static timing analyzers like aiT [1], or Otawa [5] compute safe bounds on the worst-case execution time of an application (i.e. behaviour-related timing variability). Another source of timing variability is due to interference between tasks; different techniques (e.g. static, statistical etc.) are developed to compute the necessary bounds on the timing behavior. To address interferences, static timing analyzers are extended with specialized shared cache analyses or cache-related preemption delay analyses.

Static and dynamic analyses are used to accurately capture the timing variability, whereas online monitoring is how the system identifies and addresses the deviations from specified/-validated timing values.

- static analysis is performed on the control-flow graph of the source or binary code,

- dynamic analysis is performed on traces extracted from concrete executions on the actual architecture, on an emulated target etc.,
- online monitoring is performed, at runtime, on the embedded target

The list of challenges, while non-exhaustive, is sufficient to expose the potential complexity arising when the timing variability is analyzed. One of the objectives of our multilayer runtime monitoring is to provide the necessary support, in the form of monitors, at various levels in this system representation (e.g. from high-level design to hardware).

III. Description of the Approach

An overview of our approach is presented in Figure 1, adhering to a MBD workflow for dataflow designs. A use case defines the application and a dataflow model, in Polygraph, defines the correct behavior of the system. From the Polygraph model, through a dataflow compilation, we generate elements of the source code, in C (i.e. a scheduler) which are then integrated with other elements of the source code (e.g. user-written code, libraries, RTOS APIs etc.), and finally compiled into a binary. When deploying and running such binaries on target boards, we obtain the execution traces which are then analysed by a processing engine. The goal is to check their conformance with respect to the correct timing, as defined in the high-level specification [17].

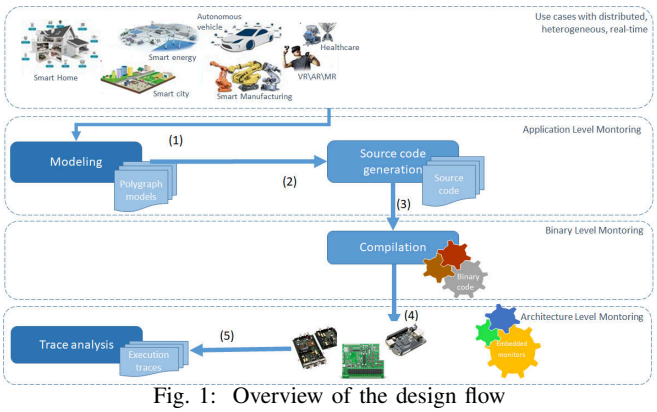


Fig. 1: Overview of the design flow

The first required step for analyzing the timing behaviour of our distributed application is to define accurately what is a "correct" behaviour for this application. To this end, we use the Polygraph model of computation and communication, presented in [11]. Polygraph is a real-time dataflow paradigm that extends the multi-rate synchronous dataflow approach.

The Polygraph model defines a complex set of causal relationships between actor/task firings, message tokens, and time instants. From a Polygraph model verified as both consistent and live, we generate elements of source code, which are linked with other user-provided code files and the instrumentation, and finally compiled into a set of binary firmware. When running these firmware programs on a heterogeneous platform made of single- or multicore targets, we monitor and analyze the execution traces generated from the instrumented programs.

In a MBD workflow, as in Figure 1, the semantics of the initial use case is successively transformed. First, the use case is formalized in the dataflow design (i.e. a high-level application semantics), then the C code is generated (i.e. an intermediate-level application semantics) and finally, it is compiled into the binary (i.e. a low-level application semantics). In order to preserve accurate traceability between the C code and the generated binary, the compilation is usually performed without optimizations. However, a coarser traceability could be preserved in the presence of compilation optimizations, so as to relate the high-level events to low-level functionalities (e.g. at function-level). In this work we consider compiler optimizations, however, regardless of the optimized/un-optimized compilation, it remains the fact that the use case semantics is obfuscated by this chain of transformations and, at the binary level, the use case characteristics (structural and functional) are difficult to identify and reason about. This motivates a runtime monitoring which should preserve some traceability features of the MBD compilation chain. Obviously, such a multilayered runtime monitoring implies the existence of execution environments at each level in the MBD workflow.

IV. Monitoring System

In the following, we briefly present some design decisions behind our multilayer runtime monitoring. We distinguish three levels of interest, starting with a dataflow Polygraph design and ending at the hardware hardware level and we organize this section as such.

A. Application level monitoring

The timing properties derived from the Polygraph design are used by the code generator to instantiate a set of software monitors. Here, we rely on the traceability of dataflow compilation chains. These monitors are inserted into the generated C code; each monitor is dedicated to observe a specific set of events, and related causal or temporal properties (as specified in the dataflow design). Then, these monitors output a trace of timestamped events for further conformance checking with respect to an expected behavior. As previously mentioned, the conformance checking could be of static or statistical nature.

B. Binary level monitoring

In case of safety-critical applications (i.e. requiring safe and tight timing bounds), timing analysis should be addressed in worst-case scenarios. There are two types of worst-case timing analysis: based on static analysis and on measurements. In the measurement-based approach, the binary is actually executed on the architecture or in a simulation environment and the results are interpreted with respect to a set of requirements. The instrumentation is necessary in this case to ensure that the collected executions are representative for the input application. In the context of runtime execution, and thus, in the presence of a monitoring system, the measurement-based

timing analysis seems to be the obvious choice. The code is instrumented at meaningful program points (actor job start, job end, message send/receive) so that runtime observations are possible whenever the code execution passes these points.

The binary instrumentation could and should be coupled with the non-intrusive hardware performance monitoring, which is detailed next.

C. Architecture level monitoring

The software-level runtime monitoring is complemented by a dedicated hardware infrastructure in the form of a Performance Monitoring Unit (PMU), available in most hardware architectures. The PMU allows recording of architectural and micro-architecture events for profiling purposes. Most modern CPU architectures contain a PMU embedded in the silicon, which provides performance counters (PMCs), a set of performance events to be counted Events such as preemptions, cache hits/misses could be tracked via the PMU to analyze the execution time of the system under test [6], [25]. Several commercial tools like ARM Streamline gator kernel module and daemon [2] are available to leverage PMUs in order to provide real-time feedback to engineers and help them to diagnose bugs or identify bottlenecks in software.

V. Case-Study and Experimentation

We showcase our approach by setting up a practical prototype platform, which embeds the monitoring layers described above into a use-case implemented as a Polygraph design.

In Figure 2, we present our use-case. This application is representative of the kind of real-time computing loads associated with advanced driver assistance systems (ADAS): the `Camera` actor generates image frames at a fixed rate e.g. 15 frames per second. These frames are analyzed by two mostly independent sub-chains. On one hand, lanes are detected on every frame using classic computer vision algorithms, and illustrates a time-critical chain. On the other hand `Objects detection` is performed using a deep neural network, and processes only a subsampled set of the input frames, in a soft real-time way. The `Display` sink actor serves as visual output, and additionally checks for deadline misses.

A. Case study implementation

The case study was implemented on a set of two Raspberry Pi multicore embedded computers, equipped with the Linux Operating system patched with PREEMPT_RT to support real-time applications, connected over Ethernet.

The RPi model 3B+ has a Broadcom BCM2835, 4-core ARMv7 processor, with 16KiB private L1 instruction, 16KiB L1 data cache, a 512KiB shared L2 cache.

Actors communicate through ZeroMQ messages using the publish/subscribe communication pattern. ZeroMQ [16], available as both C and Python-compatible, was chosen for its performance and versatility, supporting equally well inter-thread, inter-process process and Ethernet-based communications, independently from actual scheduling-related configuration. In comparison the popular ROS framework, available in C++ and Python, uses by default a single event queue for scheduling callbacks in a run-to-completion manner. This makes configuration and verification of real-time properties complex (see e.g. [34]).

Below is a simplified view of the implementation for an actor thread, where the business logic is inserted into a task and communication structure generated from the Polygraph model with inline tracing statements. The actor detailed, `Perspective Warp`, has a simple Polygraph specification: one input port, one output port, and no clock constraint, i.e. firings are triggered by incoming messages.

Listing 1: Example actor code

```
// actor thread
void *actor_perspective_warp(void *arg) {
  // Input channel: create subscriber socket
  void *z_in = zmq_socket(zctx, ZMQ_SUB);
  // connect to TCP endpoint
  zmq_connect(z_in, "tcp://...");
  // subscribe to topic
  zmq_setsockopt(z_in, ZMQ_SUBSCRIBE, "frame");

  // Output channel: Create publisher socket
  z_out = zmq_socket(zctx, ZMQ_PUB);
  // bind to inter-process com endpoint
  zmq_bind(z_out, "inproc://persp_warp");

  // Allocate image memory space
  img_t *im_in = img_new(...);
  img_t *im_out = img_new(...);

  // Main loop
  while (1) {
    trace_job_start(...);
    // receive (blocking) input message
    zmq_recv_image(z_in, im_in, ...);
    trace_token_received(...);

    // transform front view to bird-eye view
    img_perspective_warp(im_in, im_out, ...);

    // send output message
    trace_token_send(...);
    zmq_send_image(z_out, im_out, ...);
    trace_job_end(...);
  }
}
```

1) Test stubs: The ADAS camera is simulated by a `Image source` actor, implemented with Python and OpenCV. It periodically sends images (300x200 pixels, RGB format) extracted from a video file, shown in Figure 3.

The `Display` actor, implemented in Python, subscribes to the outputs of both lane and object detection chains, and additionally monitors the end-to-end latency of each processing sub-chain.

2) Lane detection: The image processing pipeline is inspired from the many Python implementations of Advanced Lane Detection published by participants to the popular ”Self Driving Car” NanoDegree from Udacity, e.g. [27].

These actors represent a hard real-time processing sub-chain. They are implemented in bare C, as three separate

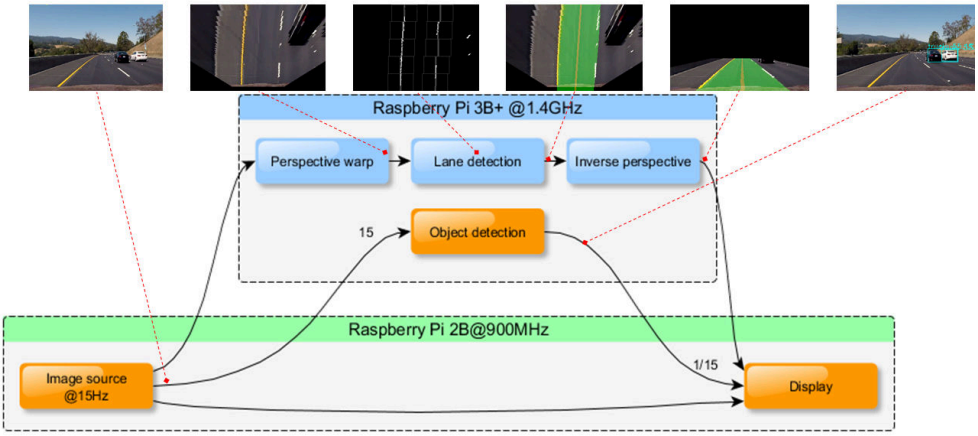


Fig. 2: Use-case dataflow

Fig. 3: Input image

threads. They are run on the isolated cores with high real-time priority.

The `Perspective Warp` actor transforms the front view image into a bird-eye-view image, following a manually-calibrated perspective transform, as shown in Figure 4a.

The `Lane Detection` actor then uses a color filter to extract lane markings, then a boxed histogram search to identify the most plausible markings (local maxima) for each lane, shown in Figure 4b.

A 2nd degree polynomial is fitted through the local maxima. The polynomial is used to extrapolate and draw curved lane besides the detected markings, shown in Figure 4c.

Finally, the `Inverse Perspective` actor projects the lane-marked bird-eye-view into a front view, shown in Figure 4d.

3) Object detection: The `Object detection` actor uses a deep neural network (DNN) to detect objects such as cars, trucks or motorcycles (see 5. The network used is the SSD Mobilenet v3, pre-trained on 300x300 color images using the COCO Small dataset [31]. This network is not trained specifically for ADAS applications, but is however representative of the type of computational load typical of computer vision DNNs.

This actor is implemented in Python, and uses the Tensorflow Lite runtime. Due to its intensive CPU and memory usage, this actor only processes 1 frame per second. This actor is mapped onto the non-isolated cores.

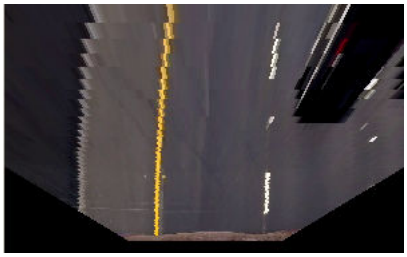
4) Variability sources: In II, we recalled the most prominent sources of temporal variability in real-time applications. The following measures are configured so as to reduce variability to a minimum:

- data-dependent behavior is reduced to a minimum: the memory allocation is authorized only during setup, and the execution flow depends on image size but not image content. Only the Gauss-Jordan matrix inversion used in the polynomial fit step has non-deterministic execution flow, however this represent a small fraction of the execution time (less than $100\mu s$);
- time-critical tasks are scheduled as high-priority real-time threads
- time-critical tasks are run on isolated CPU cores: the `isolcpu` and `taskset` commands prevent the OS from scheduling background services on two CPU cores dedicated to real-time tasks, and consequently ensure interference-free access to private L1 caches;
- inter-task synchronization is limited to blocking on message reception, thus enforcing a correct Polygraph dataflow semantics.

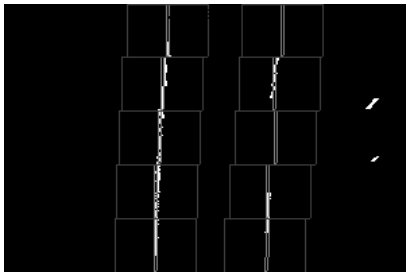
The execution times of the non-critical Object detection sub-chain on the other hand show much more variability, due to e.g.:

- data-dependent behavior: due to its black box nature, we could not inspect to what extent the TFLite runtime behavior does depend on actual image content. To the least, we observe a variable execution time, and assume that some operations depend on e.g. the number of objects detected.

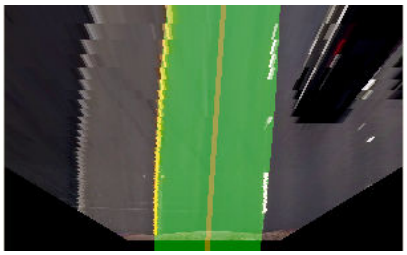
This task configuration is expected to reduce the risk of interference between non-critical and critical tasks. However, the shared L2 cache remains a possible source of interference,



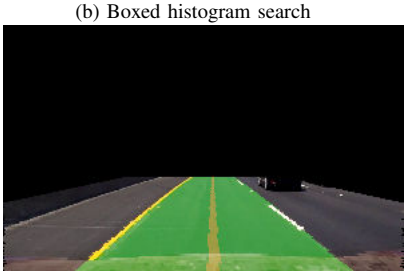
(a) Perspective warp



(b) Boxed histogram search



(c) Polynomial-fitter lane markings



(d) Inverse-perspective warped lane markings

Fig. 4: Lane detection chain: intermediate stages

Fig. 5: Detected objects

since both critical and non-critical task chains process memory chunks larger than the last-level L2 cache.

5) Monitoring infrastructure: The monitoring infrastructure is integrated within each Python and C-based actor thread, using the low-overhead LTTNG-UST infrastructure (Linux Tracing Toolkig - New Generation, User-Space Tracing). In our configuration, LTTNG-UST tracing was tested to cost approximately $5\mu s$ per trace point, well below the time granularity of the monitored events. We monitor all Polygraph dataflow relevant events, that is: actor job start, job end, message sent, message received. Using these events, we can compute the execution times of each actor job.

In addition, the `Image Source` actor generates a timestamp, which is transported together with the image payload during all processing steps. This enables other actors computing the input and output latency of each actor (time of arrival of input data, and time of departure of output data, relative to original input timestamp).

The real-time, NTP synchronized, clock `CLOCK_REALTIME` is used to compare timestamps from distributed platforms with millisecond-level precision, whereas the monotonic clock `CLOCK_MONOTONIC` is used to compare timestamps inside a given platform, up to microsecond precision for e.g. actor execution times.

The event trace generated by this monitoring infrastructure, showed in Figure 6 allows to analyze the evolution in time of execution times of critical actor jobs (in μs; top section). In particular, the execution time of critical actors is relatively stable (standard deviation is below 2% of the average). The input/output latency is computed at each actor job (in ms; middle section).

In the PMU trace (lower section), we see hardware-level metrics for the `Perspective warp` actor. In particular, from PMU monitored metrics we compute the instruction per cycle ratio (IPC), and miss rate at private (L1) as well as shared (last level) cache. In our application, a significant variability of the miss rate at shared cache LL ($stddev \approx 12\% \cdot avg$) is probably caused by interference from the non-critical, memory-intensive, `Object detection` actor. This variable LL miss rate would cause a variable access time for data outside L1 cache, and variable instruction per cycle (IPC). However since most data reads only hit the L1 cache (miss rate $< 1\%$), the effect of LL cache interference is not much visible on actor execution time or the IPC metric in this run.

B. Static WCET Analysis

In order to better establish correlations between the monitoring infrastructure and the architecture timing, we have considered a cycle-accurate, WCET analysis for the `Perspective Warp` actor. More precisely, we consider a reference implementation of this actor which is then evaluated with Otawa static timing analyzer [5] for ARM processors. Whereas specialized analyses are necessary to address shared caches or the possibility to accommodate the preemption, Otawa proposes single-core, non-preemptive analysis, hence we only directly analyze L1 cache effects. Let us elaborate next on these two aspects of the experimentation.

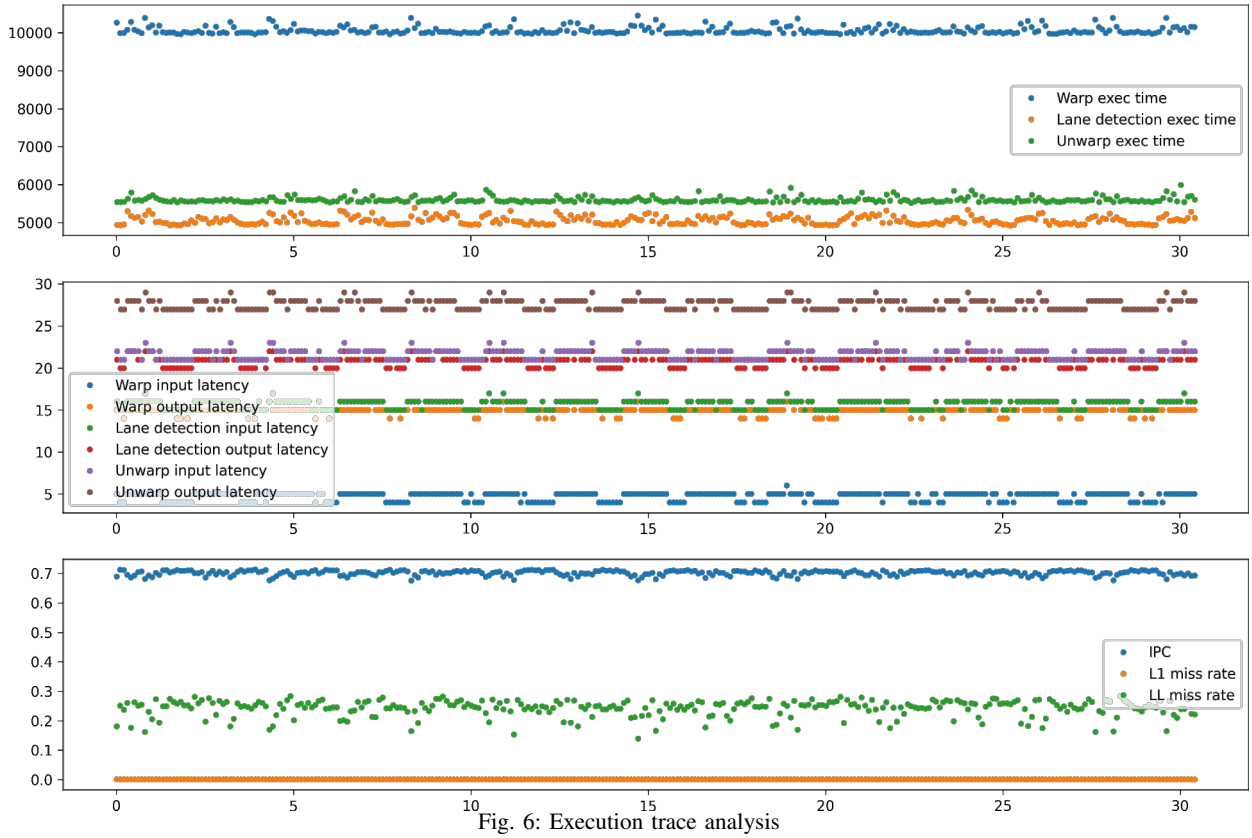


Fig. 6: Execution trace analysis

The reference implementation of the `Perspective Warp` actor is characterized by the following elements. First, the input image, as shown in Figure 3 is of fixed size (i.e. 300x200 pixels) and the output image, as shown in Figure 4a is also of fixed side (i.e. 200x133 pixels). We note that both image sizes are modifiable, but we fix them as a referenced input-output, as we have previously stated. Second, the code is organized in three stages: memory allocation of all the manipulated images, followed by a perspective transformation algorithm (through a resolution of a system of linear equations) and finally, the generation of the resulted imaged. We note that our contribution (wrt. the monitoring infrastructure) is to observe the timing contribution of the memory accesses. Third, the code is stripped of auxiliary functionalities, namely calls to the ZMQ library and other tracing operations. Finally, we also addressed a highly-optimized version (`-O3`) of this code (as used in all the other tests). From the architecture point of view, we analyze with Otawa single-core timing, using cache analyses for L1 caches (and without considering the L2 shared cache). As such, the resulting code presents its core functionality which is then evaluated with the static timing analyzer in order to obtain a timing bound in the absence of other interferences (i.e. from `ObjectDetection` and Linux background services).

The Otawa timing analyzer is also highly configurable, featuring, on the architecture side, an infrastructure for ARM, RISC-V or PPC architectures (to name a few) and on the analysis side, a wide range of abstractions to compute flow-facts from both the input program and the underlying architecture (e.g. standard cache may- and must- analyses). Also, Otawa proposes an advanced scripting to facilitate the extraction of loop addresses, in order to introduce loop bounds. We note that Otawa is accompanied by oRange [10] tool to compute these loop bounds, however, due to the fact that our code contains only for-loops, it is not necessary to consider oRange for these bounds. In our experimentation with the `Perspective Warp` actor, we consider the ARM architecture of Otawa with a simple pipeline and instruction and data caches, for which the aforementioned analyses are selected. In this setting, Otawa returns slighly more than 12 million cycles for each image processed, for a code which is dominated by memory accesses (i.e. 55% instructions). This estimate is roughly three times the cycle count measured by runtime monitoring.

The Otawa analyzer only considers a 1 level cache, whereas our target features 2 cache levels; for this reason we cannot expect Otawa to provide an accurate figure. However since the L1 miss rate is low (Fig. 6, consistent with Otawa analysis), the effect of 2nd level cache cannot explain the large difference. This discrepancy needs to be further investigated.

While these results are used as a baseline WCET behavior for `Perspective Warp`, similar investigations could be performed for the other time-critical actors of our case study.

VI. Related Work

Model-based design (MBD) approaches for real-time systems consider as, for the high-level application, a design developed using synchronous languages like Lustre [13]. For this particular language (as for others in the same family), a notion of observer could be defined [14] to address design properties. More precisely, an observer, according to [14] is another program which observers the behavior of the original application (in Lustre) and determines eventual deviations from stipulated correct behaviors. In other words, an observer is a monitor for the high-level application and a high-level Polygraph design, as in our work, could also support such an approach.

The Polygraph formal model is also used by Alkalee, a spin-off of CEA LIST, as the base of their modeling tool Euphilia [32]. Alkalee moreover provides the Receef runtime environment [33], which can supervise actors' communication events according to the input system model. The Receef monitors can additionally trigger various containment strategies for communication faults, e.g. when an expected data sample is not received on time. To the best of our knowledge, the Receef monitoring infrastructure does not cover, nor interface to, low-level architectural observation points such as hardware performance counters.

In the context of the low-level application and the underlying execution platform (wrt. the same MBD workflow), a typical monitoring systems relies on hardware performance counters (HPCs) to observe the behavior of the system under analysis. As such, HPCs are used to address security properties, e.g. in [7], safety properties, e.g. in [20] and non-functional requirements, e.g. energy [18] and timing [22]. The key aspects in using HPCs to cover a wide range of properties are that modern processors support them and also that their overhead wrt. the runtime analysis is minimal. We also consider HPCs in our current work for the same reasons, however, we also aim to correlate the results of their measurements with different observation methods (i.e. at different semantics levels in the system design). Such correlations define our framework of multilayer monitoring for mixed-criticality systems.

The worst-case timing analysis is necessary to ensure that critical tasks in mixed-criticality systems are able to meet their timing deadlines. There are two types of timing analysis - using static-based [30] and measurement-based [29] methods. Our multilayer monitoring framework draws inspiration from both, as follows. The static timing analysis, when applied to MBD workflows (i.e. a survey of methods and techniques is presented in [4]) considers the application to be represented and analyzed at each level [19] and aims to establish traceability properties between these levels. In the same way, our approach aims to exploit the traceability towards increasing the confidence between the observed timing behavior at the various levels. The measurement-based timing analysis uses the HPCs to estimate timing bounds of critical tasks, having problems of compositionality [21]. Another way of performing measurement-based timing analysis is by code instrumentation followed by the application of statistical and/or probabilistic methods on the collected results [3]. In the same way, our approach considers the HPCs, but with a broader goal, that of establishing (composable) connections with different levels in a MBD workflow.

VII. Conclusion

We explained in this document how we used a MBD methodology to define a monitoring system for a distributed execution platform in order to validate their temporal behaviour during their execution. Using a high-level dataflow language, i.e. Polygraph, the correct temporal behavior is specified. From these specifications, source code is generated or written to embed software monitors. These monitors are compiled into the application, and inserted into the application execution flow, and track dataflow-relevant events. At the binary level, two types of worst-case timing analysis are considered. For the first one, static analyses are performed on the control flow graph as a high level abstraction of the binary code. The second one a measurement-based approach where the binary is actually executed on the architecture and the results are interpreted with respect to these measurements. Finally, at the architecture level, execution is profiled using tools such as Performance Monitoring Unit (PMU). We have shown that these various monitoring layers provide complementary data, especially useful to modelling the actual execution time in presence of inter-task interference.

References

[1] https://www.absint.com/ait/index.htm

[2] https://github.com/ARM-software/gator

[3] https://www.rapitasystems.com

[4] Mihail Asavoae and Claire Maiza and Pascal Raymond, "Program Semantics in Model-Based WCET Analysis: A State of the Art Perspective", In 13th International Workshop on Worst-Case Execution Time Analysis (WCET), pp. 32–41, 2013.

[5] Ballabriga C., Cassé H., Rochange C., Sainrat P. (2010) OTAWA: An Open Toolbox for Adaptive WCET Analysis. In: Min S.L., Pettit R., Puschner P., Ungerer T. (eds) Software Technologies for Embedded and Ubiquitous Systems. SEUS 2010. Lecture Notes in Computer Science, vol 6399. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-16256-5_6

[6] W. L. Bircher and L. K. John, "Complete System Power Estimation Using Processor Performance Events," in IEEE Transactions on Computers, vol. 61, no. 4, pp. 563-577, April 2012, doi: 10.1109/TC.2011.47.

[7] Malcolm Bourdon and Eric Alata and Mohamed Kaaniche and Vincent Migliore and Vincent Nicomette and Youssef Laarouchi, "Anomaly detection using hardware performance counters on a large scale deployment", In ERTS 2020.

[8] J.-L. Colaço, B. Pagano and M. Pouzet, "SCADE6: A Formal Language for Embedded Critical Software Development", in 11th International Symposium on Theoretical Aspects of Software Engineering (TASE), pp 1-11, 2017

[9] J. Dabney, T. Harman, "Mastering Simulink", Pearson Ed, 2004

[10] Marianne De Michiel and Armelle Bonenfant and Hugues Cassé and Pascal Sainrat, "Static Loop Bound Analysis of C Programs Based on Flow Analysis and Abstract Interpretation", The Fourteenth IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), pp 161–166, 2008

[11] Dubrulle P., Gaston C., Kosmatov N., Lapitre A., Louise S. (2019) A Data Flow Model with Frequency Arithmetic. In: Hähnle R., van der Aalst W. (eds) Fundamental Approaches to Software Engineering. FASE 2019. Lecture Notes in Computer Science, vol 11424. Springer, Cham. https://doi.org/10.1007/978-3-030-16722-6_22

[12] Francalanza, A., Pérez, J.A. and Sánchez, C., 2018. Runtime verification for decentralised and distributed systems. Lectures on Runtime Verification, pp.176-210.
[13] Nicolas Halbwachs, "A synchronous language at work: the story of Lustre", International Conference on Formal Methods and Models for Co-Design, MEMOCODE, pp.3–11, 2005.
[14] Nicolas Halbwachs and Fabienne Lagnier and Pascal Raymond, "Synchronous Observers and the Verification of Reactive Systems", in Algebraic Methodology and Software Technology (AMAST), pp. 83–96, 1993
[15] Hamelin, Etienne and Ait Hmid, M and Naji, Amine and Mouafo-Tchinda, Yves, "Selection and evaluation of an embedded hypervisor: Application to an automotive platform", European Congress of Embedded Real Time Software and Systems, 2020
[16] P. Hintjens, "ZeroMQ: Messaging for Many Applications", O'Reilly Media, 2013
[17] R. Kirner, R. Lang, G. Freiberger and P. Puschner, "Fully Automatic Worst-Case Execution Time Analysis for Matlab/Simulink Models", in 14th Euromicro Conference on Real-Time Systems (ECRTS), pp 31-40, 2002
[18] Ghislain Landry and Tsafack Chetsa and Laurent Lefevre and Jean-Marc Pierson and Patricia Stolf and Georges Da Costa, "Exploiting performance counters to predict and improve energy performance of HPC systems", in Future Gener. Comput. Syst, pp. 287–298, 2014.
[19] Claire Maiza and Pascal Raymond and Catherine Parent-Vigouroux and Armelle Bonenfant and Fabienne Carrier and Hugues Cassé and Philippe Cuenot and Denis Claraz and Nicolas Halbwachs and Erwan Jahier and Hanbing Li and Marianne De Michiel and Vincent Mussot and Isabelle Puaut and Christine Rochange and Erven Rohou and Jordy Ruiz and Pascal Sotin and Wei-Tsun Sun, "The W-SEPT Project: Towards Semantic-Aware WCET Estimation", In 17th International Workshop on Worst-Case Execution Time Analysis (WCET), pp. 9:1–9:13, 2017.
[20] Corey Malone and Mohamed Zahran and Ramesh Karri, "Are hardware performance counters a cost effective way for integrity checking of programs", In Workshop on Scalable trusted computing, STC@CCS, pp. 71–76, 2011
[21] Cristian Maxim and Adriana Gogonel and Irina Mariuca Asavoae and Mihail Asavoae and Liliana Cucu-Grosjean, "Reproducibility and representativity: mandatory properties for the compositionality of measurement-based WCET estimation approaches", In SIGBED Rev., pp. 24–31, 2017.
[22] Jan Nowotsch and Michael Paulitsch and Arne Henrichsen and Werner Pongratz and Andreas Schacht, "Monitoring and WCET analysis in COTS multi-core-SoC-based mixed-criticality systems", In Design, Automation & Test in Europe Conference & Exhibition (DATE) pp. 1–5, 2014
[23] Navabpour S., Bonakdarpour B., Fischmeister S. (2015) Time-Triggered Runtime Verification of Component-Based Multi-core Systems. In: Bartocci E., Majumdar R. (eds) Runtime Verification. Lecture Notes in Computer Science, vol 9333. Springer, Cham.
[24] Reinbacher T., Függer M., Brauer J. (2013) Real-Time Runtime Verification on Chip. In: Qadeer S., Tasiran S. (eds) Runtime Verification. RV 2012. Lecture Notes in Computer Science, vol 7687. Springer, Berlin, Heidelberg.
[25] R. Rodrigues, A. Annamalai, I. Koren and S. Kundu, "A Study on the Use of Performance Counters to Estimate Power in Microprocessors," in IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 60, no. 12, pp. 882-886, Dec. 2013, doi: 10.1109/TCSII.2013.2285966.
[26] Norman Scaife and Christos Sofronis and Paul Caspi and Stavros Tripakis and Florence Maraninchi, "Defining and translating a "safe" subset of Simulink/Stateflow into Lustre", In EMSOFT, pp.259–268, 2004
[27] Siddharth Sharma, Advanced Lane Lines Detection, https://github.com/sidroopdaska/SelfDrivingCar/
[28] Stoller, S.D., Bartocci, E., Seyster, J., Grosu, R., Havelund, K., Smolka, S.A. and Zadok, E., 2011, September. Runtime verification with state estimation. In International conference on runtime verification (pp. 193-207). Springer, Berlin, Heidelberg.
[29] Ingomar Wenzel and Raimund Kirner and Bernhard Rieder and Peter P. Puschner, "Measurement-Based Timing Analysis", In Leveraging Applications of Formal Methods, Verification and Validation, Third International Symposium, (ISoLA), pp. 430–444, 2008.
[30] Reinhard Wilhelm and Sebastian Altmeyer and Claire Burguière and Daniel Grund and Jörg Herter and Jan Reineke and Björn Wachter and Stephan Wilhelm, "Static Timing Analysis for Hard Real-Time Systems", In Verification, Model Checking, and Abstract Interpretation, 11th International Conference, (VMCAI), pp. 3–22, 2010.
[31] Hongkun Yu and Chen Chen and Xianzhi Du and Yeqing Li and Abdullah Rashwan and Le Hou and Pengchong Jin and Fan Yang and Frederick Liu and Jaeyoun Kim and Jing Li, TensorFlow Model Garden, https://github.com/tensorflow/models, 2020
[32] Euphilia, the systems modeling tool, ALKALEE, https://www.alkalee.fr/products/euphilia/
[33] Receef, the embedded features orchestration software, ALKALEE, https://www.alkalee.fr/products/receef/
[34] Daniel Casini and Tobias Blaß and Ingo Lütkebohle and Björn B. Brandenburg, Response-Time Analysis of ROS 2 Processing Chains Under Reservation-Based Scheduling, ECRTS, 2019

SAFETY & SECURITY MONITORING CONVERGENCE AT THE DAWN OF OPEN HARDWARE

Sylvain Girbal, Jimmy Le Rhun, Daniel Gracia Pérez, David Faura
Thales Research & Technology, Palaiseau, France
{sylvain.girbal, jimmy.lerhun, daniel.gracia-perez, david.faura}@thalesgroup.com

***Abstract*—The emergence of multi-core processors into the embedded world one decade ago led to the IT/OT convergence. In the last few years, a second convergence is ongoing in the domains of safety-critical and security-critical systems. Nowadays both safety protection systems and security protection systems are relying on monitoring to ensure the expected critical software behaviour. However, all these systems incur a performance overhead to fulfill the service, that could be an issue with time-critical systems.**

The safety monitoring process that was mostly involved at design time, focusing both on the software and the hardware to ensure hard real-time behaviour and propose some mitigation to faults and errors, is now also targeting the integration and deployment phases with adaptive runtime engines to deal with the timing interference issue of multi-core architectures.

The security monitoring process that was mostly used to focus on protecting against software vulnerabilities at runtime now has to consider unreliable hardware that some cyberthreats such as Spectre and Meltdown are able to exploit.

This paper proposes a short survey of existing Heath & Usage Monitoring Systems (HUMS) and Hardware Intrusion Detection Systems (HIDS) in safety-critical and security-critical systems, and their associated monitoring features.

We then promote the benefits of communalizing these monitoring features to reduce the performance impact of HUMS and HIDS systems. In this context, open hardware architectures are a major opportunity, allowing us to analyze the hardware design without black box, to seek formal proof of critical properties, to implement mechanisms for improved predictability, and to enhance hardware-level observability.

I. INTRODUCTION

During the last decades, we observed successive convergences of computing systems. It started 20 years ago with the convergence between high-performance computing (HPC) that focused on power efficiency, and the mobile market that was seeking for more performance and functionalities, both were dealing with the challenge of controlling the balance between low power and high performance.

A decade ago, thanks to the multi-core processors, a second convergence [35, 47, 37] started between the mission critical market (such as avionics, automotive, healthcare and robotics) and the mainstream consumer electronics market, also known as the IT/OT convergence (for *Information Technology* and *Operative Technology*). This convergence was fueled by the increasing requirements of the mission-critical market for computing performance, as well as the growing need of embedding more critical functionalities in the mobile devices, that started to be connected to cars or healthcare systems.

During the past years, dealing with both safety and security has become a prime requirement for embedded cyber-physical systems [65] leading to a third convergence. However, the practice is different for safety and security.

The paper is organized as follows: Section I-A and Section I-B respectively present the safety- and the security-related practices during the system development life-cycle. Section II presents the impact of the introduction of multi-core architecture on these practices, and the common safety and security trends with regard to monitoring. Section III provides a survey of existing HUMS and HIDS systems focusing on their monitoring features. Finally, Section IV presents our view on the future of monitoring techniques promoting the communalization of monitoring techniques to reduce the performance costs.

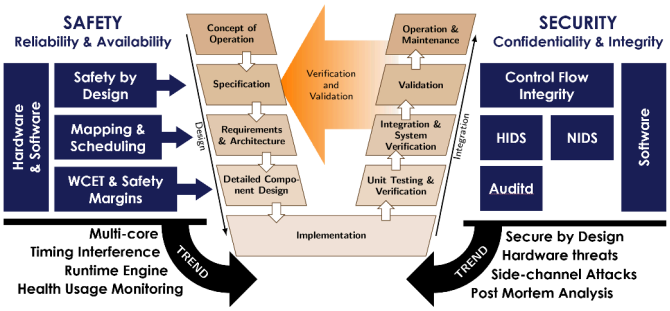


Fig. 1. Safety & Security trends with regards to monitoring

A. Safety Practices

Figure 1 present the usual V-shaped system development life-cycle, and how safety and security integrates into this scheme. Regulation standards [44, 45, 71] led the **safety critical** industry to focus mainly on the design phases to ensure reliability and availability by design [39], both at the software [72] and the hardware [73] levels.

Safety critical applications are also usually characterized by stringent real-time constraints, which are usually guaranteed by determining the application Worst Case Execution Time (WCET). This WCET computation usually relies on analysis tools based on static program analysis tools [90, 70], detailed hardware model, as well as measurement techniques through execution or simulation [38] to provide an estimated upper

bound of the execution time, introducing some safety margins as depicted in Figure 2.

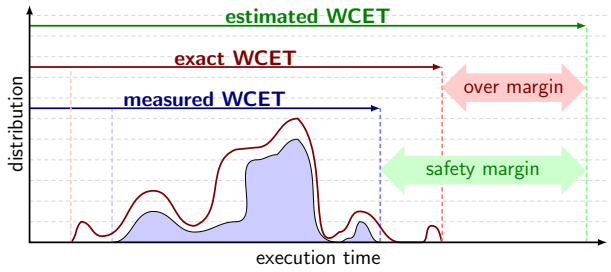


Fig. 2. Estimation of the Worst-Case Execution Time, and the over-estimation problem

The current best practices include the use of many statically defined mechanisms, such as static scheduling of periodic tasks, static memory allocation and mapping, as a way to improve the ability to demonstrate a deterministic behaviour of the system.

The introduction of multi-core architecture to safety-critical systems as presented in Section II was shown to have a significant negative impact on worst-case performance [11, 66], while enhancing average performance. The current trend to deal with this problem is to add safety nets that involve both the integration and the operational phases.

B. Security Practices

In computing systems (hardware plus software), **security** has typically been provided as specialized software executed on top of them during operation, to overcome their weaknesses, especially software vulnerabilities and intrusion detection systems [10].

For example at the device level *host intrusion detection/protection systems* (HIDS/HIPS) are used as security software applications running on the device itself, while at the network level the *network intrusion detection/protection systems* (NIDS/NIPS) are used to protect the computing systems from malicious communications.

Furthermore, external observation is nowadays not enough, requiring the integration of security solutions to be integrated at the application level or underlying layers, like operating system and even hardware. Examples are control flow integrity (CFI) [78] solutions that integrate program flow supervision to detect the unexpected exploitation of software by attackers.

A raising trend is the exploitation of hardware vulnerabilities of computing systems. Exploits such as Spectre [53] and Meltdown [58] have demonstrated that attackers can exploit vulnerabilities at hardware level. While software security solutions at different levels (compiler, operating system, firmware, etc.) have partially mitigated these vulnerabilities with some performance costs, they can only be fully and efficiently addressed at hardware level. However, the design of security solutions at hardware level is not a simple feature, and must be implemented with care to avoid introducing additional vulnerabilities [36, 56, 27].

II. Impact of Multi-Core Architecture on Safe & Secure Systems

The recent shift from single-core COTS (component off-the-shelf) to **multi-core COTS processors** for safety-critical and security-critical products was appealing both in terms of average performance and in terms of size, weight and power (SWaP) [8], actually fitting with the exponential growth in performance requirements in the embedded domain.

However, multi-core processor architectures are introducing both new sources of time variations, and new vulnerabilities: multi-core architectures are characterized by the fact that they are embedding **shared hardware resource** between the cores, as depicted in Figure 3.

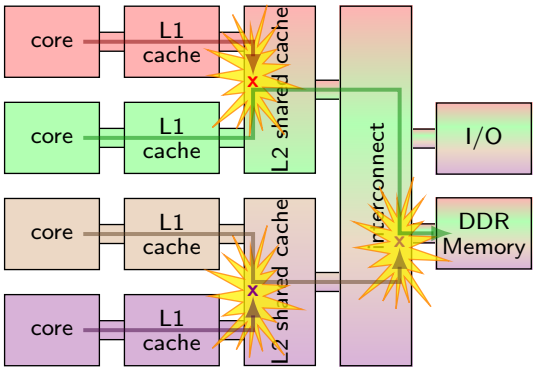


Fig. 3. Shared hardware resources & timing interference

In this figure, each core is associated with a different color, private resources (such as L1 caches) are colored with the same color as their cores, and **shared hardware resources** are represented with a shade of the involved core colors (such as L2 caches, the interconnect and the main memory).

On a multi-core COTS processor, different pieces of software will be executed on different cores at the same time. Such different software will compete electronically to use these shared hardware resources, eventually involving hardware arbiters to deal with concurrent accesses, and introducing inter-task or inter-application delay and jitter, defined as **timing interference** [31].

The lack of precise documentation coupled with the black-box aspects of these hardware arbiters impact the WCET analysis tools, that have difficulties to deal with real industrial programs running on multi-core COTS architectures [52, 64], while resource over-provisioning is no longer an option. The industry is therefore facing a trade-off between performance and predictability [90].

The literature [32] proposes several Deterministic Platform Solutions to tackle this problem, including control solutions aiming at completely preventing such timing interference and regulation solutions reducing the amount of interference below a harmful level. However, all these techniques have to circumvent the black-box aspect of the COTS architecture, at the cost of decreased performance.

On the **security** side, shared hardware resources are also a source of vulnerabilities against the confidentiality and the integrity properties, as the other cores have an opportunity to access or alter private data of co-running applications.

Multi-core processors have also impacted the security of the devices. The addition of new shared resources (e.g. bus, interconnect, L3 shared caches) or new mechanisms (or variations of existing ones) to operate in a multi-core configuration (e.g. cache coherence protocol) have introduced new attack possibilities. For example, the shared bus has been exploited to perform timing covert channels [92], the introduced cache coherence protocols have been exploited in Meltdown and Spectre variations [85] or to perform covert channel attacks [62], and dynamic frequency scaling of the different cores used to perform covert channel attacks [3].

Those examples are mainly from the IT domain, but are equally applicable to modern embedded systems with high connectivity (e.g. edge- and fog-computing), using equally complex computer architectures [88] and speculative processor cores with similar vulnerabilities (e.g. ARM v8 cores subject to Spectre attacks [7]).

Like for safety, the lack of precise documentation of those new resources and mechanisms, or their unexpected usage (as frequent in security exploits) make the defense against these attacks difficult to anticipate, endangering the confidentiality and integrity of the system and running applications.

A. Trends in Safe & Secure Systems

The introduction of multi-core architecture in safe or secure systems already led to a couple of common solutions like memory-space partitioning [71, 82], allowing to integrate several applications with different safety/security requirements in the same platform. In such a scheme, the operating system and some hardware components (MMU, TLB) are responsible for providing each software partition with its own protected memory space, actually ensuring data segregation.

Multi-cores also led to some converging trends between safety and security practice appearing in Figure 1. The timing interference problem on time-critical systems for instance has led to a set of control-based or regulation-based solutions [32] that involves the integration phase or even the operational phase with dedicated run-time engines. Health usage monitoring systems (HUMS) [63] are also targeting this operational life-cycle.

Being **secure-by-design** [60] has become an hot topic for security-critical systems with a particular focus on the design phases of the V-shaped model with approaches dedicated to security. Also, the hardware should rather not be considered as reliable anymore, but as a potential source for side-channel attacks instead [62].

A common trend for both safety and security critical systems is therefore to focus on all phases of the life-cycle from conception to operation. Also, all these new practices rely at different degrees on the ability to monitor the system.

III. Current Monitoring in Safe & Secure Systems

This section provides a survey of emerging monitoring techniques for safe & secure systems. Each subsection emphasizes a specific technique as well as the associated application domain, and how it is integrated relatively to domain-specific safety/security requirements.

A. Control Flow Integrity (CFI)

A classic security vulnerability consists in exploiting buffer overflow bugs to alter the program stack and perform code-reuse attacks [83, 42] executing malicious operations. Return Oriented Programming [80] exploits this weakness by altering the return address stored in the stack, hijacking the control flow when the function returns. Jump Oriented Programming [12, 16] later extended this threat to register corruption of direct branch targets.

Control Flow Integrity (CFI) [1, 78, 13] is a well known security approach to detects control flow hijacking, as deviation from the expected application control flow.

At compilation time, a Control Flow Graph (CFG) is generated. Each node corresponds to an uninterruptible instruction sequence (also called basic block) without any branch or return instruction. Forward edges correspond to jumps and function calls while backward edges correspond to return statements.

At runtime, to prevent control flow hijacking, we have to ensure that all jumps and all returns correspond to a legitimate edge of the CFG. However, it usually involves some code instrumentation and therefore both code intrusiveness and performance cost penalties.

1) System call instrumentation: In [49], Kadar et al. propose the less costly alternative of instrumenting/monitoring the system calls rather than the branches and returns, unexpected system call succession being also a sign of malicious code execution. They proposed a methodology to evaluate the system call instrumentation of the PikeOS real-time hypervisor in terms of performance overhead.

If they were able to keep the per system-call tracing overhead quite low with a maximum overhead of 700ns each, the high variability in number of system calls between applications as well as the critical impact of the number of context switches both made the global overhead quite complex to determine, and to be very application dependent.

Furthermore, detecting unexpected syscall succession would require to previously learn this expected succession. Applying machine learning techniques to this looks promising but will lack the certificability of being able to build an exact CFG at compile-time as required by CFI.

2) Hardware-assisted CFI: Another alternative to reduce the CFI performance overhead is to rely on the hardware rather than the software to perform branch trace collection. In [54], Kuzhiyelil and al. exploit the CoreSight hardware core coupled with hypervisor partitioning to transparently perform control flow tracing.

The CoreSight is a hardware component in ARM-v8 systems performing real-time tracing and debugging of applications. It embeds private trace FIFO queues and dedicated data paths ensuring an interference-less gathering of debugging information. Among the collectible information, the CoreSight is able to collect taken branches, thus actively monitoring the application control flow.

As depicted in Figure 4(a), a first step at compile time consists in generating the control flow graph in addition to a non-instrumented binary. During runtime, as shown in 4(b),

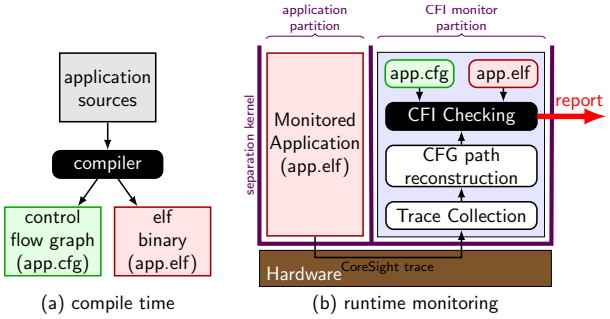


Fig. 4. Transparent control flow integrity as defined in [54]

the trace of taken branches and returns is collected through the CoreSight debug core to reconstruct the path followed in the CFG and check and report for unexpected jumps to malicious code sections.

This notion is then extended in [48] with a particular focus on mixed-critical and safety-critical systems running both critical and non-critical software. The later usually offers a larger attack surface and fewer guarantees. A particular focus in these systems is to ensure freedom from interference as required by the safety standards standards [45, 44, 72] to prevent an attack on the non-critical software from impacting the critical software.

From the security point of view, such methods show excellent detection results both in terms of false positive and false negative rates. However the performance overhead of CFI can be significant, even with a partitioned RTOS. In [54], the authors measured a worst case overhead of up to 55% for branch-heavy applications. In [48], the compromize between overhead and coverage of the detection system can be configured by the monitoring partition. Setting an overhead of 10% was shown to be sufficient to correctly classify 99% of the samples.

B. HIDS for Avionics

Safety has for long being a prime concern in avionics, however next generation aircraft systems aim at providing more and more connectivity and services to the passengers. From the security point of view, however, it continuously increases the attack surface [69]. In such a context is critical to protect aircraft software from malicious modifications of the onboard applications, and airworthiness regulations have evolved to that extent [74, 75].

The industrial process associated with avionic applications currently involve several segregated actors. The software components are developed by different *solution providers* working under computing resource budget constraints from the *platform provider*. Later these solutions are put together to build an avionic system by the *integrator*. Ideally, if the budget constraints are respected the integration should be seamless.

Protecting the aircraft software therefore involves protecting third party blackbox software components where the sources may not be available, with no possibility to add source code instrumentation or analysis as required for CFI.

Host Intrusion Detection Systems (HIDS) and anomaly detection [19, 41] have been introduced in a IT context as a way to detect such malicious modification threats, and they are now widely used for the security of information systems as a mechanism to detect abnormal or suspicious activities.

Introducing HIDS to existing safety-critical avionics products requires to take additional domain-based constraints, such as preserving the real-time constraints and the freedom of interference between components from different solution providers; keeping the footprint small, both in terms of memory footprint and resource requirement; provide explainable and reproducible results; being efficient even on blackbox components.

In [22], Damien and al. especially study how to bring HIDS to Integrated Modular Avionics (IMA) systems, considering above-mentioned specific avionics requirements. The author proposes to observe the ARINC 653 API calls performed as a model to the normal or altered behaviour of the application. This could be performed from the specific avionics RTOS by capturing both the call sequence and each call duration, but it requires a significant amount of resources to so, especially memory resources for the call trace.

Different strategies are studied to reduce the amount of data being logged, including only logging memory communication related ARINC 653 calls, or keeping only call frequency information rather than a full trace information. Several detection algorithms are also considered.

The results demonstrates that the solution keeping the whole trace, and therefore with the larger memory footprint is not the one providing the best detection results. Keeping on the more meaningful communication-related data helps with obtaining a more efficient classifier. Frequency information has also shown to be an efficient way to detect malicious modifications while requiring a much smaller memory footprint.

This approach is extended in [21] with the ability to provide a first onboard diagnosis of the anomalous behaviour, paving the way to future reactive systems with the capacity to block an attack. The author proposes the adjunction of an evolutive knowledge database as depicted in Figure 5.

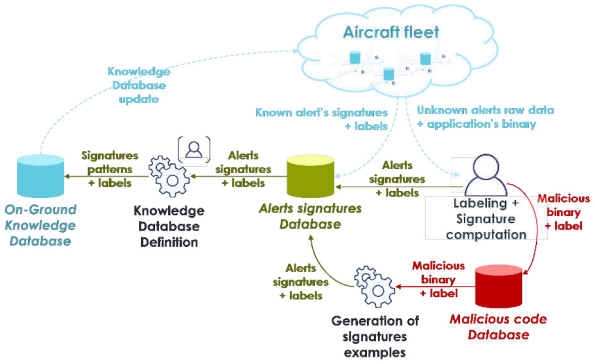


Fig. 5. Knowledge database of anomalous behaviour, as defined in [21]

This onboard knowledge database includes already known anomalous or malicious behaviour including cyber-attacks or safety-related failures, and relies on alert signatures. This database has to be regularly updated with new signatures during ground maintenance operations.

Within such a scheme, an anomaly is defined as an unknown sequence of ARINC 653 API calls, or an abnormal duration of such a call. When too many anomalies are observed during the same time frame, an alert is raised.

The anomaly signature is then searched for a match within the onboard knowledge database in order to identify the current alert and to provide a feedback message usable immediately or during later onground investigations.

The overall approach shows interesting results: In terms of error detection, it exhibits a detection accuracy and correct anomaly labeling of 87% after 70 samples. In terms of resource usage, the trace monitoring caused a +2.7% on API call runtimes and a +3.3% impact when including the early onboard diagnosis.

C. NIDS for Safety-Critical Networks

Network Intrusion Detection Systems (NIDS) are widely used in IT computing to analyze network communication traffic [10], usually performing network packet inspection and comparing them to a database of attack patterns. Such NIDS could also be combined with neural networks to increase the detection rate [30, 87].

Safety critical-systems usually embeds specific deterministic networks such as the AFDX or the CAN bus [5, 6]. In such a safety critical-context, the key properties is to guarantee the deterministic behaviour of the network and maintain a high level of integrity [84]. These mechanisms ensure a safe end-to-end transfer of information between different subsystems, preventing the propagation of network packet errors at runtime.

A664P7 [5] implements a network monitoring protocol, which tracks relevant events and communication protocol errors at the switch and end-system level. The monitoring is based on Hardware PMC dedicated to Network observability, defined in a Management Information Base (MiB) [43].

The Network Management system [2] is in charge of realizing the correlation of multi-protocol information collected from all the components to detect/localize network failure. A path of improvement is to distribute the advanced monitoring computing functionality between the aircraft systems and the airline maintenance center [25].

D. HW/SW characterization for Aerospace systems with performance counters

In the context of multi-core processors for aerospace systems, safety standards require the analysis and mitigation of undesirable contentions due to concurrent usage of shared hardware resources, called timing interference. A joint HW/SW characterization of the system behaviour is needed, with a high level of precision, including system calls and fine-grain interaction on shared resources.

To achieve such a precision level, hardware assistance is mandatory, and we can take advantage of mechanisms initially developed for performance tuning, called the performance monitoring counters (PMC). These are simple hardware registers, able to count various events within a processor core or in a SoC infrastructure, such as CPU clock cycles, the number of instructions of a certain type, cache access or misses, branch mispredicts, bus access, etc. There are typically few counters per core (e.g. 4 to 8), but up to hundreds of events to choose from.

The main advantage of this technique is the precise and timely sampling of low-level information. Time is cycle-accurate, and reading a counter only takes a couple of instructions. Activation of counters and event selection typically requires supervisor privileges, but in most cases reading the counter can be allowed in user mode, for lower overhead.

The Measurement Environment for Time-Critical Systems (METrICS) [34] is a self-contained approach to characterize multicore processor behaviour and timing interference. The basic part is a *probe*, a small piece of code inserted in an application program to perform a sampling of selected hardware events. The probe is designed to minimize timing overhead, making use of macros and inline assembly and completely avoiding system calls. Sampled counters data is stored in the main memory, with a structure designed to fit in a single cache line along with a unique probe identifier, core and process identifiers.

The probe typically takes less than 190ns to run on 1.8GHz PowerPC. The timing overhead is therefore very low, at the cost of a small source code intrusivity.

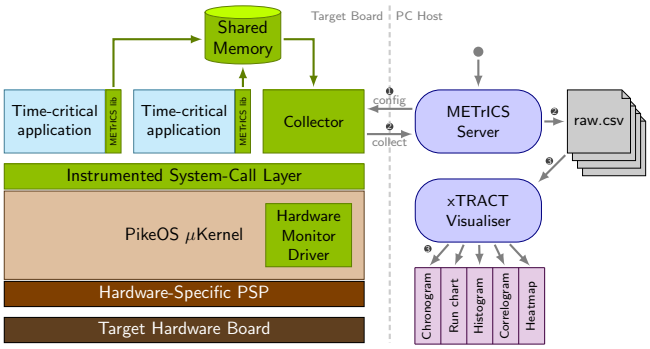


Fig. 6. METrICS architecture, as defined in [34]

The extraction of stored data is performed out of the real-time section by another component, named Collector. This is an independent partition, that is not scheduled during the measurement phase. The collector is also in charge of initialization, and data link with the host (typically using Ethernet). On-target computation is kept to a minimum, and all data is preserved for post-processing.

Several statistical techniques can then be used offline to analyze the collected data. For duration of tasks or system calls, two probes can be paired and the raw values of counters can be subtracted. Time series can be derived, as well as full histograms exhibiting the observed-WCET. Correlation between counters helps identifying possible cause of time interference.

E. Online monitoring with PMCs

As the timing overhead to collect the PMCs presented in the previous section is very low, the same principles can be extended as an online monitoring technique, as part of the health and safety usage monitoring subsystem. Multiple online interference mitigation techniques have been exploiting the

PMCs' monitoring. MemGuard [93] exploits memory accesses PMCs to monitor the number of accesses done by the different partitions in a multi-core system and perform on-board decisions guided by budget limits associated to these partitions to minimize the interference impact. Similarly, in [20] the time and instructions counters are used in addition to the memory access counters to enhance the interference control. BB-RTE (Budget-Based RunTime Engine) [33] proposes a variation of the previous two where all the shared resources PMCs, i.e. not only memory accesses, are considered to perform online interference mitigation.

To achieve the same goals without source code modification, another approach presented by Airbus in [29] makes use of an external Safety-Net processor to monitor the operational multi-core processor. As depicted in Figure 7, an external FPGA device contains a soft-core processor, connected with a high-speed link to the debug infrastructure of the multi-core. It is able to periodically access the performance counters through this link, and record the application behaviour without intrusivity.

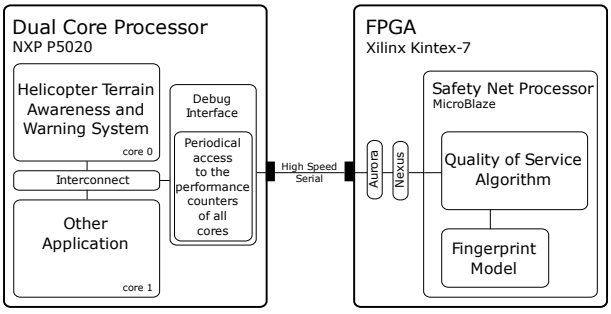


Fig. 7. Safety-Net architecture, as defined in [29]

In this case, the rate of executed instructions (in instr/μs) is the value of interest, and its evolution over the duration of a major scheduling frame. Such fingerprints can be concatenated over several time slices for a given partition, and compared between isolated and concurrent executions. The presence of timing interference can be detected by a shift of the fingerprint, as the rate of executed instruction is slightly lower and its variation pattern is slightly delayed. A slowdown of 1.5% can be reliably detected in less than 15ms.

Likely, solutions to detect security incidents exploit the PMCs monitoring techniques [28]. For example, Li [55] proposes a PMC monitoring solution able to detect a Spectre attack using the cache miss and branch missprediction PMCs available in most processors. Similarly, Chiappetta [17] exploits the L3 accesses PMC to detect side-channel attacks exploiting the caches.

These and other studies have shown the capability of PMCs usage to develop safety and security characterization and monitoring solutions. However, multiple studies [81, 86, 9] have proven that the exploitation of these same PMCs for the development of security attacks.

F. Miscellaneous Monitoring techniques

Some monitoring-base detection solutions introduce creative techniques either for trace collection or for classification.

In automotive, services used to be integrated as distinct electronic control units (ECUs) each with a specific hardware. With the multiplication of the number of services, as well as optimizing Size, Weight and Power (SWaP), ECUs are now integrated on the same hardware as virtual machines.

The HIDS introduced in the [50] position paper models the system interactions both in terms of OS service usage as well as hardware activity collected through Performance Monitor Counters. Traces are collected as words and sentences, applying Natural Language Processing techniques to predict further traces from the already collected sequence. Threat detection is then performed by comparing the actual trace to the predicted one, not requiring any form of previous offline learning.

HUMS systems on their side can consider input data beyond the software or the hardware behaviour. In [59], Airbus captures the impact of vibration in an helicopter on the mechanical components wear-out to guide out maintenance tasks.

Such an activity used to being performed with on-ground calculators during stress-test procedures. The paper proposed to shift this activity onboard.

The main asset is to implement the fundamental ability to analyze on the fly the data collected from various embedded aircraft sensors and quickly identify safety/security-related events to take the most appropriate actions. However, the authors pointed some critical missing features to collect information from the physical layer, such as a lightweight tracing and timestamping mechanism, as well as strong cycle-accurate synchronized time requirements.

The goal is going toward experimentation in onboard machine learning [46] to pave the way to predictive maintenance as part of Flight Data Monitoring [25].

G. AI usage in Monitoring

In the survey presented this section, several monitoring techniques embed different flavours of artificial intelligence: Some [34, 20, 33] are just relying on statistical analysis to compute a threshold used for outlier/anomaly detection. Some others [29, 48] rely on machine learning techniques to compute such a threshold, but are keeping the inference as a simple comparison to this threshold. Machine learning is used to a greater extent by [55, 17, 22, 21] with a more complex online inference system based on neural networks. Alternative AI approaches such as genetic algorithms or sequence prediction are also used in [22, 50]. Finally a few papers [49, 59] are considering machine learning techniques as a possible future work.

It corresponds to a recent trend of using artificial intelligence for anomaly detection, that was first introduced for IT systems [15, 57], and is now considered for embedded safety and security critical systems [51].

IV. The Future of Monitoring in Safe & Secure Systems

As illustrated by Section III, software and hardware monitoring have become prime requirements for both safety and

security, however this extra monitoring activity comes with complexity, security and performance costs. As a consequence, the convergence between safety-critical and security-critical systems would benefit from communalizing the monitoring features.

A. Multi-level aspects

Figure 8 shows the different layers composing the technology stack. This includes the user-mode applications, the domain-specific middlewares, the operating system layer, the embedded hardware SoC and the physical communication layer. The arrows show regular component interactions, for instance the operating system scheduler sets which application should be running, setting up the proper MMU entries for logical to physical address translation and flushes the hardware TLB caching the MMU.

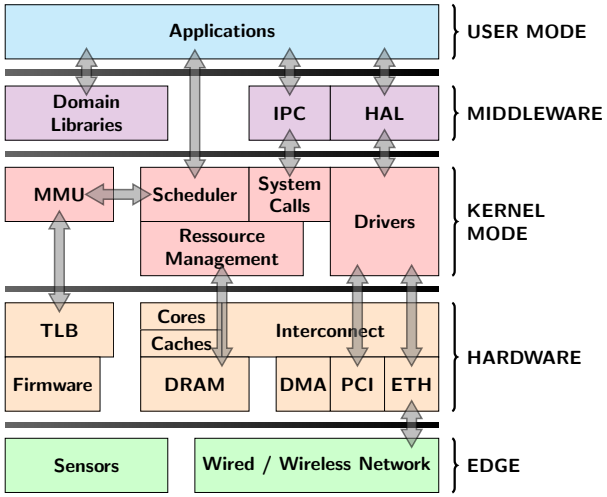


Fig. 8. Multi-layer software/hardware stack

Among all the monitoring techniques presented in the survey of Section III, many techniques are already multi-level implying several layers.

For instance, hardware-assisted CFI [54] monitors the application path behaviour from the debug module of the hardware layer. To perform a transparent, uninstrumented monitoring of the application the [21] HIDS gathers ARINC 653 call information from the operating system/middleware layers. Network monitoring has the opportunity to be performed within the hardware abstraction layer (HAL), at the Ethernet driver level, or further down on the physical link.

In fact, each layer has only limited information and lacks the semantics of the other layers: the hardware layer has efficient and immediate pipeline-related information from the Performance Monitor Counters (PMCs) such as branches or even cache misses but does not know which application, task or thread is running. This information is only available either directly from the application layer or from the scheduler of the operating system layer.

Accessing the PMCs registers is also doable from the application layer, but at the cost of additional code to be executed to capture this information and also the cost of traversal time from the application layer down to the hardware (e.g. syscalls), providing slightly outdated and noisy information.

Gathering information from multiple layers is therefore necessary to perform efficient monitoring. Pushing this concept further, we might also benefit from gathering information from multiple sources, communalizing the monitoring information from different subsystems. It will reduce the overall performance costs, as many HIDS/HUMS are accessing the same information, and provide opportunity to identify new correlations for detection. However, it comes with a set of associated challenges.

B. The Requirements for Communalizing Monitoring Information

The survey from Section III identified several challenges or specific requirements for the different detection techniques:

1) Trace collection: All previously detailed techniques optimize trace collection either in terms of intrusiveness or performance:

At user **application level**, [54, 21, 29, 55] minimize application intrusiveness, avoiding any code modification by either performing the trace collection automatically, at hardware level or externally at the cost of some performance. [34, 50] focus on minimizing time intrusiveness and optimizing performance by requiring some user instrumentation.

Some techniques perform their collection at **operating system level**, such as [54, 21, 50, 20, 93], requiring the source code of the RTOS. Instead, [34, 55] rely on a kernel driver to perform the actions requiring privileged mode.

Many collection techniques rely on **hardware level** information: [54, 29] use a specific but COTS hardware component to gather debug traces. [34, 29, 55, 50, 20, 93] are gathering the hardware-level Performance Monitor Counters.

2) Classification/detection: is usually performed outside of the monitored application, as an distinct adhoc process. To limit the impact of such a process on the monitored application, [54, 21, 33, 50, 20, 93] are relying on a separation kernel or partitioning hypervisor.

From the **hardware** point of view, [29, 59] are relying on specific onboard hardware to perform the detection, whereas [34] relies on an external host to perform the statistical analysis as a post-processing action.

3) Exploiting monitoring for side channel attacks: As stated before, most classification algorithm are performed in software from trace data extracted from the hardware [54, 21, 34, 33, 50, 55, 17]. As a consequence the monitored information is available in user-mode and therefore also available for malicious purpose to implement side-channel attacks. [29, 59] are alleviating this risks by performing this on dedicated hardware.

Ideally, to reduce the attack surface and performance cost, supervision solution should be implemented at hardware level with private resources. However, customizing hardware could be costly.

C. Open Hardware in Safe & Secure Systems and the benefits of low-level monitoring

Many of the issues faced in critical embedded systems are rooted in the incomplete or imprecise knowledge of the processor system inner operation. The high complexity of current processors necessitates abstraction for efficient usage, and many implementation details are hidden to the developer, either with each software layer or even at hardware register level. While those hidden details are specifically designed not to impact functional correctness or average-case performance, they can have a significant impact on worst-case performance that matters for critical systems.

In reaction to the increasing complexity of hardware implementation of processor systems, we observe a current rise in popularity for **Open Source Hardware**. Several institutions team up to develop more generic hardware platforms, with a common set of requirements, and share under a more or less permissive license the burden of development, verification and documentation on an accessible code base. From this point, extensions and product differentiation is possible. The open-source nature also allows security audit of the source code, and precise documentation of low-level mechanisms such as shared resource arbitration.

1) RISC-V ecosystem: One of those Open-Source Hardware initiatives, currently gathering a large momentum in the industry, is **RISC-V** [89]. As a fifth-generation processor instruction set specification, it aims at becoming the "Linux of the processors" with applications from simple IoT devices up to supercomputers. The specification is maintained by RISC-V International [76], a non-profit organization structured in several working groups and strong of more than 280 members, including all major actors of the computing industry.

The RISC-V ISA is modular and specifies 32-, 64- and 128-bits versions, with various optional extensions such as bitwise operations or hardware virtualization. In addition, industry associations such as **OpenHardware Group** [67] or **Chips Alliance** [18] focus on open-source implementations of RISC-V processors.

2) Safety & Security in the context of Open Hardware systems: Safety- and security-critical systems are niche markets, compared to mainstream computing products, and face difficulty to have their stringent requirements satisfied in the COTS market. The Open Source ecosystem allows several stakeholders to team for the specification, the implementation and the validation of processors with suitable features, whereas they would not have had the resources to do so or to influence COTS vendors individually. Within RISC-V International, these topics are notably addressed in the Security Standing Committee and the Functional Safety Special Interest Group.

Among the specific mechanisms required for critical systems, a guarantee of time-, memory-space and computer-space isolation is a common need to ensure that a dysfunctional application cannot impact other critical applications. Health monitoring, integrity checks and run-time monitoring are needed to ensure proper operation and react to errors/attacks.

In the context of multi-core processors, achieving those properties present extra difficulty as explained in Section II. However the openness of Open Hardware allows to lift the curtain on previously black-box subsystems such as the arbiters on the interconnect and other shared hardware resources, and take into account their scheduling policy in interference mitigation strategies.

Furthermore, the access to such low-level design elements offers the opportunity to perform formal validation of safety- and/or security-related properties, for example using tools such as Yosys [91].

Lastly, it provides the opportunity to add or modify specific features in the implementation, and tune hardware mechanisms to increase predictability or immunity to attacks, e.g. by reducing the sources of speculation.

3) Benefits of Open Hardware on Monitoring: In addition to the well-known advantages of openness listed above, Open-Source Hardware presents several interesting opportunities for monitoring activities, as it enables addition or modification of hardware mechanisms specifically optimized.

Low-level instrumentation can generate large amounts of data, as it operates at very fine grain. It is therefore beneficial to implement filtering capabilities in order to focus on relevant information for a given observation goal. Filter examples include address range of memory transaction, initiator core, or type of request, but very complex detection can be crafted with a combination of several monitors and filters.

Another opportunity is to propagate semantic information across system layers, that would otherwise be lost or ignored. For example, an application parameter or loop iteration could be logged along with hardware performance counters. As described in [23] this information and the filtering capabilities are actually required to perform an efficient monitoring, as those required in [55, 17].

Many low-level monitoring mechanisms focus on each processor core, but the most worrisome aspect of critical multi-core lies in the interactions at SoC-level, notably within the interconnect. Additional monitors and counters in the SoC infrastructure allow a better understanding of interference channels, and in some cases enable mitigation techniques. This is particularly interesting when combined with filtering capabilities.

A simple interrupt mechanism triggered when a performance counter reaches a given threshold allows the implementation of budget-based interference mitigation, e.g. de-schedule a task when it has exhausted its budget of memory access to ensure it is not slowing down other tasks, either because of a bug or an attack.

A recent development in this direction is the Safe Statistics Unit from BSC [14], consisting of three novel kinds of performance counters tailored for the monitoring of timing interference on multi-core processors: the Maximum-Contention Control Unit (MCCU), the Request Duration Counter (RDC), and the Cycle Contention Stack (CCS). By analyzing the traffic on a shared processor bus, it allow new ways of understanding and controlling the interactions between cores. The MCCU monitors access to a shared resource, and enforce per-core quotas. The RDC can log maximum duration of a bus transaction, and act as a watchdog. The CCS helps in the identification of the initiator at the origin of the interference suffered by each core.

Finally, a dedicated hardware mechanism can be an improvement for security concerns, as it can be designed as a separate entity, not accessible from vulnerable application software or isolated from side-channel attacks. Similarly, designs with private memories and communication channels dedicated to monitoring avoid any overhead on the system operation, and any skew in the observed behaviour.

D. Opportunities for AI-based HIDS/HUMS

Some industry domains are reluctant to introduce artificial intelligence in safety critical systems. For decades, autonomous piloting, one of the most critical function in avionics has been relying on deterministic algorithms, and the benefits in shifting to AI-based systems is not likely to cover the additional certification costs. As the monitoring subsystem runs at a lower criticality level, such a system is a good candidate to introduce AI, with lower certification costs. However embedded safety-critical systems have additional constraints to be taken into account by the AI-based systems.

In avionics, post-mortem analysis is critical to maintain flight authorization from the authorities. As a consequence it is not only a matter of successfully performing the classification/detection with an acceptable amount of false-positive and false-negative, but also a matter of identifying the root causes of a detected event. Also post-mortem analysis involves a large degree of replaying the conditions that led to the studied event. As a consequence, we expect determinism in terms of the same input causing the same answer from the AI system. Such a behaviour seems to be incompatible with continual/continuous learning [68], but more in cope with explainable AI [24] such as symbolic AI [40, 61].

Embedded systems also have stringent requirements in terms of resource usage, including processing power, real-time behaviour and memory footprint [26]. Machine learning [4, 79] seems to be in adequation with such requirements: the learning phase that builds neural networks requires both processing power and memory, but could be performed offline on external systems, while only the inference part is performed onboard with a small neural network memory footprint, and involves a constant number of multiply-add operations leading to deterministic time processing.

Another challenge for the application of AI techniques in critical systems is the low occurence of some safety/security hazards in these systems. This difficults the usage of AI techniques, particularly during the learning phase. Frugal learning [77] especially focuses on being less dependent on large collections of input data, learning from few samples with additional semantic information.

Beyond these additional challenges for embedding AI in safety/security critical systems, it also comes with new benefits and potential new markets, especially predictive maintenance, and the ability to propose more complex embedded HUMS/HIDS systems.

V. Conclusion & Future Works

In this paper, we presented a survey of emerging monitoring technologies implemented in Health Usage Monitoring Systems or Host Intrusion Detection Systems in the domain of safety-critical and security-critical devices.

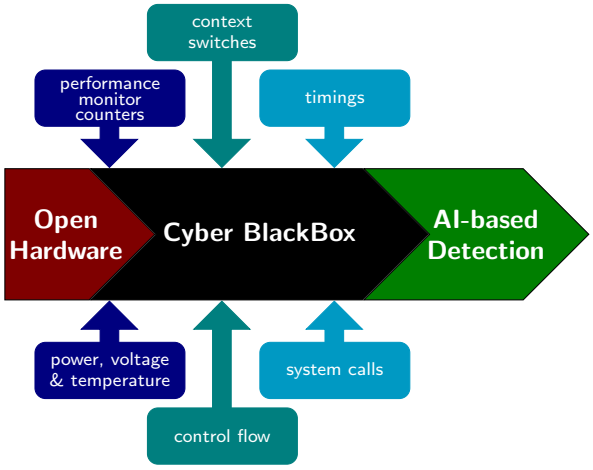


Fig. 9. Cyber BlackBox: a multi-level/muti-source approach to monitoring

We then promoted the communalization on monitoring resources in a multi-layer and multi-source approach, as depicted in Figure 9.

We foresee Open Hardware as being an enabler to implement the supervision system with dedicated software and especially hardware resources, allowing us both to capture the necessary hardware-related information, but also as a way to protect from side-channel attacks exploiting the monitoring features.

Finally, using Artificial Intelligence-based techniques looks like a promising opportunity to merge monitoring data of different natures, layers and sources to identify new correlations allowing us to detect either new safety failures or security threats, or to detect already known ones sooner.

Machine Learning in particular is usually decomposed into two phases: a preliminary learning phase, and an inference phase performing the classification. The resource-consuming learning phase could be performed offline, only leaving the inference phase to be embedded onboard and therefore reducing the resource footprint of such systems, while keeping open the explainability challenge.

The Cyber BlackBox approach presented in Figure 9 considers each monitoring source/technique as an optional plugin to the overall supervision infrastructure, so that it could be adapted the the specifics of each safety/security critical domain.

Acknowledgements

This research work has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreements No 871385 and 869945.

References

[1] M. Abadi, M. Budiu, U. Erlingsson, and J. Ligatti. Control-flow integrity principles, implementations, and applications. *ACM Transactions on Information and System Security (TISSEC)*, 2009.

[2] AIRBUS. US8190727B2, airbus patent, network management system for an aircraft, 2012.

[3] M. Alagappan, J. Rajendran, M. Doroslovački, and G. Venkataramani. DFS covert channels on multi-core platforms. In *IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, 2017.

[4] E. Alpaydin. *Introduction to machine learning*. MIT press, 2020.

[5] ARINC. Aircraft data network part 7 avionics full duplex switched ethernet (afdx) network, 2005.

[6] ARINC. General standardization of can (controller area network) bus protocol for airborne use, 2007.

[7] ARM. Whitepaper: Cache speculation side-channels, version 2.5, 2020.

[8] T. G. Baker. Lessons learned integrating COTS into systems. In *Proceedings of the First International Conference on COTS-Based Software Systems*, ICCBSS '02, 2002.

[9] S. Bhattacharya and D. Mukhopadhyay. Utilizing Performance Counters for Compromising Public Key Ciphers. *ACM Trans. Priv. Secur.*, 21(1), jan 2018.

[10] E. Biermann, E. Cloete, and L. M. Venter. A comparison of intrusion detection systems. *Journal on Computers & Security*, 2001.

[11] J. Bin, S. Girbal, D. Gracia Pérez, A. Grasset, and A. Merigot. Studying co-running avionic real-time applications on multi-core COTS architectures. *Embedded Real Time Software and Systems conference*, 2014.

[12] T. Bletsch, X. Jiang, V. W. Freeh, and Z. Liang. Jump-oriented programming: A new class of code-reuse attack. In *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '11, 2011.

[13] N. Burow, S. A. Carr, J. Nash, P. Larsen, M. Franz, S. Brunthaler, and M. Payer. Control-flow integrity: Precision, security, and performance. *ACM Comput. Surv.*, Apr 2017.

[14] G. Cabo, F. Bas, R. Lorenzo, D. Trilla, S. Alcaide, M. Moretó, C. Hernández, and J. Abella. Safesu: an extended statistics unit for multicore timing interference. In *2021 IEEE European Test Symposium (ETS)*, pages 1–4, 2021.

[15] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41(3), jul 2009.

[16] S. Checkoway, L. Davi, A. Dmitrienko, A.-R. Sadeghi, H. Shacham, and M. Winandy. Return-oriented programming without returns. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, CCS '10, 2010.

[17] M. Chiappetta, E. Savas, and C. Yilmaz. Real time detection of cache-based side-channel attacks using hardware performance counters. *Applied Soft Computing*, 2016.

[18] CHIPS (Common Hardware for Interfaces, Processors and Systems) Alliance. https://chipsalliance.org/. [Online].

[19] M. Christodorescu, S. Jha, and C. Kruegel. Mining specifications of malicious behavior. In *Proceedings of the the 6th Joint Meeting of the European Software Engineering Conference*, ESEC-FSE '07, 2007.

[20] A. Crespo, P. Balbastre, J. Simó, J. Coronel, D. Gracia Pérez, and P. Bonnot. Hypervisor-based multicore feedback control of mixed-criticality systems. *IEEE Access*, 2018.

[21] A. Damien, P.-F. Gimenez, N. Feyt, V. Nicomette, M. Kaâniche, and E. Alata. On-board diagnosis: A first step from detection to prevention of intrusions on avionics applications. In *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*, 2020.

[22] A. Damien, M. Marcourt, V. Nicomette, E. Alata, and M. Kaâniche. Implementation of a host-based intrusion detection system for avionic applications. In *2019 IEEE 24th Pacific Rim International Symposium on Dependable Computing (PRDC)*, 2019.

[23] S. Das, J. Werner, M. Antonakakis, M. Polychronakis, and F. Monrose. SoK: The Challenges, Pitfalls, and Perils of Using Hardware Performance Counters for Security. *2019 IEEE Symposium on Security and Privacy (SP)*, May 2019.

[24] F. K. Došilović, M. Brčić, and N. Hlupić. Explainable artificial intelligence: A survey. In *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 0210–0215, 2018.

[25] European Union Aviation Safety Agency. Flight data monitoring on atr aircraft, 2016.

[26] M. Evchenko. Frugal Learning: Applying Machine Learning with Minimal Resources. 2016.

[27] S. Fei, Z. Yan, W. Ding, and H. Xie. Security vulnerabilities of SGX and countermeasures: A survey. *ACM Comput. Survey*, July 2021.

[28] J. C. Foreman. A survey of cyber security countermeasures using hardware performance counters. *CoRR*, abs/1807.10868, 2018.

[29] J. Freitag and S. Uhrig. Quality of Service for Integrated Modular Avionics (IMA) on Multicore Processors using a Safety Net Architecture. In *ERTS 2018*, 9th European Congress on Embedded Real Time Software and Systems (ERTS 2018), Toulouse, France, Jan. 2018.

[30] F. Garzia, M. Lombardi, and S. Ramalingam. Artificial neural networks framework for security/safety systems management and support. In *International Carnahan Conference on Security Technology*, 2017.

[31] S. Girbal, D. Gracia Pérez, J. Le Rhun, M. Faugère, C. Pagetti, and G. Durrieu. A complete toolchain for an interference-free deployment of avionic applications on multi-core systems. In *Proceedings of the 34th Digital Avionics Systems Conference*, DASC'2015, 2015.

[32] S. Girbal, X. Jean, J. Le Rhun, D. Gracia Pérez, and M. Gatti. Deterministic Platform Software for hard real-time systems using multi-core COTS. In *Proceedings of the 34th Digital Avionics Systems Conference (DASC)*, 2015.

[33] S. Girbal and J. Le Rhun. BB-RTE: a Budget-Based RunTime Engine for Mixed & Time Critical Systems. In *Embedded Real Time Software and Systems*, ERTS '18, 2018.

[34] S. Girbal, J. Le Rhun, and H. Saoud. METrICS: a measurement environment for multi-core time critical systems. In *Embedded Real Time Software and Systems*, ERTS '18, 2018.

[35] S. Girbal, M. Moretó, A. Grasset, J. Abella, E. Quiñones, F. J. Cazorla, and S. Yehia. On the convergence of mainstream and mission-critical markets. In *50th IEEE Design Automation Conference (DAC)*, 2013.

[36] B. Gras, K. Razavi, E. Bosman, H. Bos, and C. Giuffrida. ASLR on the line: Practical cache attacks on the MMU. In *NDSS*, 2017.

[37] B. Gyselinckx, R. Vullers, C. Van Hoof, J. Ryckaert, R. F. Yazicioglu, P. Fiorini, and V. Leonov. Human++: Emerging technology for body area networks. In *2006 International Conference on Very Large Scale Integration*, 2006.

[38] R. Heckmann and C. Ferdinand. Verifying safety-critical timing and memory-usage properties of embedded software by abstract interpretation. In *Proceedings of the conference on Design, Automation and Test in Europe*, DATE'05, 2005.

[39] C. Hobbs. *Embedded software development for safety-critical systems*. CRC Press, 2019.

[40] V. Honavar and L. Uhr. Symbolic artificial intelligence, connectionist networks & beyond. 1994.

[41] N. Hubballi, S. Biswas, and S. Nandi. Sequencegram: n-gram modeling of system calls for program based anomaly detection. In *3rd International Conference on Communication Systems and Networks*, 2011.

[42] A. Humayed, J. Lin, F. Li, and B. Luo. Cyber-physical systems security—a survey. *IEEE Internet of Things Journal*, 2017.

[43] IETF. Rfc4293: Management information base for the internet protocol (ip), 2006.

[44] International Electrotechnical Commission. IEC 61508: Functional safety of electrical, electronic, or programmable electronic safety-related systems, 2011.

[45] International Organization for Standardization (ISO). ISO 26262: Road Vehicles – Functional Safety, 2011.

[46] ITU-T Focus Group on Aviation Applications of Cloud Computing for Flight Data Monitoring. Existing and emerging technologies of cloud computing and data analytics. 2016.

[47] A. Jahn, M. Holzbock, J. Muller, R. Kebel, M. de Sanctis, A. Rogoyski, E. Trachtman, O. Franzrahe, M. Werner, and F. Hu. Evolution of aeronautical communications for personal and multimedia services. *IEEE Communications Magazine*, 2003.

[48] M. Kadar, G. Fohler, D. Kuzhiyelil, and P. Gorski. Safety-aware integration of hardware-assisted program tracing in mixed-criticality systems for security monitoring. In *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2021.

[49] M. Kadar, S. Tverdyshev, and G. Fohler. System calls instrumentation for intrusion detection in embedded mixed-criticality systems. In *CERTS*, 2019.

[50] M. Kadar, S. Tverdyshev, and G. Fohler. Towards host intrusion detection for embedded industrial systems. In *2020 50th Annual IEEE-IFIP International Conference on Dependable Systems and Networks-Supplemental Volume (DSN-S)*, 2020.

[51] G. Kasparaviciute, M. Thelin, P. Nordin, P. Söderstam, C. Magnusson, and M. Almljung. Online encoder-decoder anomaly detection using encoder-decoder architecture with novel self-configuring neural networks & pure linear genetic programming for embedded systems. In *Proceedings of the 11th International Joint Conference on Computational Intelligence*, IJCCI 2019, page 163–171, Setubal, PRT, 2019. SCITEPRESS - Science and Technology Publications, Lda.

[52] R. Kirner and P. Puschner. Obstacles in worst-case execution time analysis. In *Proceedings of the 11th IEEE Symposium on Object Oriented Real-Time Distributed Computing*, 2008.

[53] P. Kocher et al. Spectre attacks: Exploiting speculative execution. In *40th IEEE Symposium on Security and Privacy (S&P'19)*, 2019.

[54] D. Kuzhiyelil, P. Zieris, M. Kadar, S. Tverdyshev, and G. Fohler. Towards transparent control-flow integrity in safety-critical systems. In *ISC*, 2020.

[55] C. Li and J.-L. Gaudiot. Online Detection of Spectre Attacks Using Microarchitectural Traces from Performance Counters. In *2018 30th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, 2018.

[56] M. Li, Y. Zhang, Z. Lin, and Y. Solihin. Exploiting unprotected I/O operations in AMD's secure encrypted virtualization. In *Proceedings of the 28th USENIX Conference on Security Symposium*, 2019.

[57] B. Lindemann, B. Maschler, N. Sahlab, and M. Weyrich. A survey on anomaly detection for technical systems using lstm networks. *Computers in Industry*, 131:103498, 2021.

[58] M. Lipp et al. Meltdown: Reading kernel memory from user space. In *27th USENIX Security Symposium*, Baltimore, Aug. 2018.

[59] M. Lo, N. Valot, F. Maraninchi, and P. Raymond. Real-time on-Board Manycore Implementation of a Health Monitoring System: Lessons Learnt. In *9th European Congress on Embedded Real Time Software and Systems (ERTS 2018)*, Toulouse, France, Jan. 2018.

[60] S. Longari, A. Cannizzo, M. Carminati, and S. Zanero. A secure-by-design framework for automotive on-board network risk analysis. In *2019 IEEE Vehicular Networking Conference (VNC)*, 2019.

[61] J. Mattioli, P.-O. Robic, and T. Reydellet. L'intelligence artificielle au service de la maintenance prévisionnelle. 07 2018.

[62] C. Maurice, C. Neumann, O. Heen, and A. Francillon. C5: Cross-cores cache covert channel. In *DIMVA*, 2015.

[63] S. Mekid. IoT for health and usage monitoring systems: mitigating consequences in manufacturing under cbm. In *18th IEEE International Multi-Conference on Systems, Signals & Devices (SSD)*, 2021.

[64] E. Mezzetti and T. Vardanega. On the industrial fitness of WCET analysis. In *Proceedings of the 11th International Workshop on Worst Case Execution Time Analysis (WCET2011)*. 2011.

[65] H. Mun, K. Han, and D. H. Lee. Ensuring safety and security in can-based automotive embedded systems: A combination of design optimization and secure communication. *IEEE Transactions on Vehicular Technology*, 2020.

[66] J. Nowotsch and M. Paulitsch. Leveraging multi-core computing architectures in avionics. *European Dependable Computing Conference*, 2012.

[67] OpenHW Group: Proven processor IP. https://www.openhwgroup.org/. [Online].

[68] G. I. Parisi, R. Kemker, J. L. Part, C. Kanan, and S. Wermter. Continual Lifelong Learning with Neural Networks: A Review. *Neural Networks*, 113:54–71, 2019.

[69] S. Parkinson, P. Ward, K. Wilson, and J. Miller. Cyber threats facing autonomous and connected vehicles: Future challenges. *IEEE Transactions on Intelligent Transportation Systems*, 2017.

[70] P. Puschner and A. Burns. Guest editorial: A review of worst-case execution-time analysis. *Real-Time Systems*, 2000.

[71] Radio Technical Commission for Aeronautics (RTCA) and EURopean Organisation for Civil Aviation Equipment (EUROCAE). DO-297: Software, electronic, integrated modular avionics (IMA) development guidance and certification considerations.

[72] Radio Technical Commission for Aeronautics (RTCA) and EURopean Organisation for Civil Aviation Equipment (EUROCAE). DO-178B: Software considerations in airborne systems and equipment certification, 1992.

[73] Radio Technical Commission for Aeronautics (RTCA) and EURopean Organisation for Civil Aviation Equipment (EUROCAE). DO-254: Hardware considerations in airborne systems and equipment certification, 1992.

[74] Radio Technical Commission for Aeronautics (RTCA) and EURopean Organisation for Civil Aviation Equipment (EUROCAE). DO-326: Airworthiness security process specification, 2010.

[75] Radio Technical Commission for Aeronautics (RTCA) and EURopean Organisation for Civil Aviation Equipment (EUROCAE). DO-356: Airworthiness security methods and considerations, 2015.

[76] RISC-V International. https://riscv.org/. [Online].

[77] H. Sahbi, S. Deschamps, and A. Stoian. Frugal Learning for Interactive Satellite Image Change Detection. In *2021 IEEE International Geoscience and Remote Sensing Symposium IGARSS*, pages 2811–2814. IEEE, 2021.

[78] S. Sayeed, H. Marco-Gisbert, I. Ripoll, and M. Birch. Control-flow integrity: Attacks and protections. *Applied Sciences*, 2019.

[79] J. Schmidhuber. Deep learning in neural networks: An overview. *CoRR*, abs/1404.7828, 2014.

[80] H. Shacham. The geometry of innocent flesh on the bone: Return-into-libc without function calls (on the x86). In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, 2007.

[81] M. Spisak. Hardware-Assisted Rootkits: Abusing Performance Counters on the ARM and X86 Architectures. In *Proceedings of the 10th USENIX Conference on Offensive Technologies*, 2016.

[82] R. Strackx, F. Piessens, and B. Preneel. Efficient isolation of trusted subsystems in embedded systems. In *International Conference on Security and Privacy in Communication Systems*, 2010.

[83] L. Szekeres, M. Payer, T. Wei, and D. Song. SoK: Eternal war in memory. In *2013 IEEE Symposium on Security and Privacy*, 2013.

[84] P. Toillon, P. B. Champeaux, D. Faura, W. Terroy, and M. Gatti. An optimized answer toward a switchless avionics communication network. In *2015 IEEE/AIAA 34th Digital Avionics Systems Conference (DASC)*, 2015.

[85] C. Trippel, D. Lustig, and M. Martonosi. MeltdownPrime and SpectrePrime: Automatically-synthesized attacks exploiting invalidation-based coherence protocols, 2018.

[86] L. Uhsadel, A. Georges, and I. Verbauwhede. Exploiting Hardware Performance Counters. In *2008 5th Workshop on Fault Diagnosis and Tolerance in Cryptography*, 2008.

[87] D. W. F. L. Vilela, A. D. P. Lotufo, and C. R. Santos. Fuzzy artmap neural network ids evaluation applied for real ieee 802.11w data base. In *International Joint Conference on Neural Networks (IJCNN)*, 2018.

[88] X. Wang, Y. Yang, and Y. Han. Enforcing security for real-time multicore embedded system. In *2018 IEEE 4th Information Technology and Mechatronics Engineering Conference (ITOEC)*, pages 1551–1556. IEEE, 2018.

[89] A. Waterman, Y. Lee, D. A. Patterson, and K. Asanovic. The RISC-V instruction set manual, 2014.

[90] R. Wilhelm et al. The worst case execution time problem, overview of methods and survey of tools. *ACM Trans. Embed. Comput. Syst.*, 2008.

[91] C. Wolf. Formal verification with symbiyosys and yosys-smtbmc. *URL http://www.clifford.at/papers/2017/smtbmc-sby/slides.pdf*, 2017.

[92] Z. Wu, Z. Xu, and H. Wang. Whispers in the hyper-space: High-speed covert channel attacks in the cloud. In *Proceedings of the 21st USENIX Conference on Security Symposium*, Security'12, 2012.

[93] H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, and L. R. Sha. Memguard: Memory bandwidth reservation system for efficient performance isolation in multi-core platforms. In *19th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, Apr. 2013.

Session Th.5.B

Process modelling

Thursday 2nd June

17:00

–

Room Lauragais

Towards an agile, model-based multidisciplinary process to improve operational diagnosis in complex systems

Nikolena Christofi[1], Xavier Pucel[2], Claude Baron[3], Marc Pantel[4], Sébastien Guilmeau[5], Christophe Ducamp[6]

Toulouse, France

Abstract

Systems' online diagnostics require multidisciplinary system knowledge and experience by their operators. When the complexity of the system rises, operational (in-service) diagnostics become a complex task. In an effort to improve the efficiency while better handling the complexity of diagnostics during operations, the authors propose a methodology aiming to increase the agility in complex systems' development processes. This paper introduces a new way to construct operational models early on the development cycle so as to improve the performance of monitoring activities and, ultimately, increase the confidence on the systems' resilience provided by its design.

Keywords: Satellite Systems, Health Monitoring, Online Diagnosis, Model-Based Systems Engineering (MBSE), Model-Based Safety Assessment (MBSA), Maintainability, Behaviour Trees, Model-Based Operations, Agile Techniques

1. Introduction

To this day, monitoring activities and associated tools used for Fault Detection and Diagnosis (FDD) [1] during system operation are limited to specific functions provided by certain subsystems, while largely relying on the operators' experience [2]. Past data i.e. Return of Operating Experience (ROE), also feed the algorithms used for FDD. However, the ever growing complexity of embedded systems demand that the tools used for system monitoring have an underlying knowledge of the whole system structure and behaviour, so as to provide a faster, and more reliable diagnosis, especially when time is a key operational constraint. What is more, without precise knowledge of the system design, it is ambitious for diagnostic tools to be able to provide a unique possible failure source, or the correct troubleshooting procedure, in case the monitoring data contain alarm inconsistencies.

Diagnostic tools shall thus be able to provide the operators with dedicated views of the system, which will help them perform diagnosis more efficiently. These views must incorporate the appropriate system information with the defined level of detail while keeping monitoring and diagnosis in the main focus. An overly detailed view of the system would make the operation model too heavy and long to exploit, while if the view is too coarse or stays only at high level, the model might lack crucial information to assist in diagnosis. To this end, one approach consists in collecting available information from the system architecture designed during Systems Engineering (SE), as well as from the Safety Analyses (SA) documents and models, the latter regarding the potential failures. The aim would then be to use the collection of data, information and models, to produce a tool that will support the operators in their monitoring task: interpreting housekeeping data, troubleshooting problems, and performing system maintenance.

Email address: nikolena.christofi@irt-saintexupery.com (Nikolena Christofi)
[1]IRT Saint Exupéry, LAAS-CNRS, Airbus Defence & Space, INSA Toulouse, Université de Toulouse
[2]ONERA, DTIS, Université de Toulouse
[3]INSA Toulouse, LAAS-CNRS, ISAE-SUPAERO, Quartz-Supméca, Université de Toulouse
[4]IRIT, Toulouse INP, Université de Toulouse
[5]IRT Saint Exupéry
[6]Airbus Defence & Space

Our goal is thus to use the system design information to construct operational diagnosis tools. We postulate that this is possible with the introduction of an Operations-Dedicated Model (ODM), which shall include both system design information –SE data, as well as the information produced by selected system dysfunctional analyses –SA data. The latter shall help in the diagnosis activities by suggesting a possible source for the error, and its location in the system.

We explore Behaviour Trees (BTs) as a solution towards the construction of ODMs to be used for operational diagnosis. Considering that BTs can provide an intuitive way to model and simulate system behaviour, they can also be used to meet the needs for a diagnostic model which includes the system's dynamic aspects and not only its structural information –as it is the case for the currently available diagnosis models. BTs have shown a lot of potential in the last decade, mainly with their application to robotics and Artificial Intelligence (AI). Initially developed within the gaming community to replace Finite State Machines (FSM) with more user friendly models, their use could be widened to the field of operational diagnosis to model, implement and monitor the state of complex systems.

This paper explores the use of BTs to create a system behavioural model –namely ODM, oriented towards system monitoring, which can be exploited during operations in order to increase efficiency in diagnosis. We show in place that the BT model can be built during the design phase and along the production of other design models and documents, in a co-design manner, thus contributing to an agile system development process. The co-design of system and BT models is a process within which documents and models of system function and dysfunction and BT models of system monitoring for operational diagnosis are mutually constituted.

The structure of this paper is as follows. Section 2 outlines the context of our research and the problematic addressed. Section 3 presents a literature review on the involved topics. The proposed approach is presented in Section 4, using an illustrating example in Section 4.1, and perspectives on the use of ODMs inside monitoring tools for diagnostic purposes in Section 4.2. Finally, Section 5 draws conclusions and discusses future work opportunities.

2. Context and Motivation

Despite the recent flourishing of model-based languages, methods and tools for system development –system design and Verification and Validation –V&V activities, diagnostic tools created for the same systems, have known a different path. Existing diagnosis tools are often designed after the system is built; hence the opportunities to make the system easy to operate –which would be the case if the tools were constructed during system design, are lost. Either under the form of data-based [3] or model-based [4], these tools are often targeting single system functions/subsystems –e.g formation flight [5], reaction wheels [6], or thrusters [7, 8, 9]. Consequently, their specificity to one function or subsystem makes them non-generic, and thus not reusable.

Moreover, *diagnostic models fail to include safety analyses methods' derived information*; methods such as Failure Modes and Criticality Effects Analysis (FMECA), Fault Tree Analysis (FTA), Failure Logic Modelling (FLM) [10, 11], Failure Propagation Modelling (FPM) [12], or Model-Based Safety Assessment [13, 12]. On the other hand, system diagnostic tools *can be very generic* i.e. agnostic of the domain usage: they use general diagnosis techniques, such as Machine Learning (ML) [14], *not specific to spacecraft operations*.

Moreover, operational diagnostics, as a core part of satellite maintenance, is limited by many constraints, should thus consist one of the major concerns in satellite design. Therefore, by integrating a diagnostic tool in the system development process and co-designing it along with the system itself, we can ensure that the operational diagnosis activities are well taken under consideration before the satellite deployment.

As mentioned in [15], this would mean that the system design and its monitoring models influence each other. The proposed process is depicted in Figure 1, where it is illustrated with a space satellite system application. Following the system's deployment, the operator, having at their disposal the housekeeping data the Operation Centre (OC) has received from on-board the satellite, can perform diagnosis with the help of the monitoring tool. The latter is using the ODM built during the system development process (concurrently with the system design models) in the form of BTs. This would add confidence in the diagnosis results as well to the operations model. For the needs of our project we assume that the ODM is representative of the actual system emitting the monitoring data.

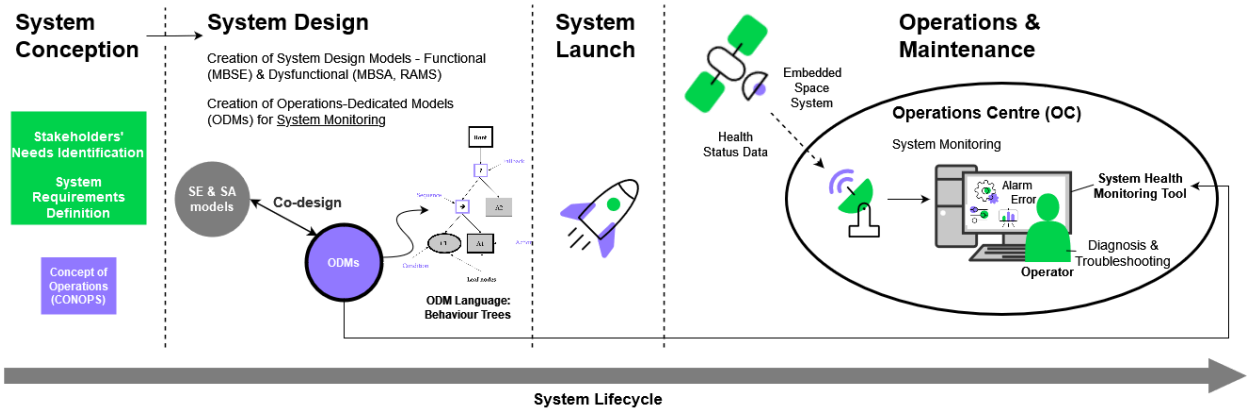


Figure 1: Overview of the proposed approach: creating ODMs during system design phase (along with SE & SA models), to be used as a basis for the "System Health Monitoring Tool".

3. Literature Review

This section consists of a description of the current State of the Art (SoA) in the fields of MBSE –section 3.1, and MBSA, as a prominent System Safety Assessment (SSA) methodology –section 3.2. The aim is to introduce the reader to the topics involved in our approach. Finally, subsection 3.3 presents the basic semantics of BTs, which is the formalism used to create the ODMs.

3.1. Model-Based Systems Engineering (MBSE)

MBSE facilitates the fulfillment of the SE requirements through design –structural and behavioural system modelling in various decomposition levels. SE *models* allow the engineers to verify the solution's design, functions and envisaged operations, early on in the development process, resulting to the reduction of the overall project cost. MBSE allows design alterations to take place in the beginning of system development and not when the solution is furtherly developed.

By applying MBSE, one can create *architectural models* (express system structure and decomposition), *functional models* (express system functions), and *behavioural models* (express system behaviour in different operational scenarios). To represent these models, several types of formalism are used, along with their associated languages and methodologies. The most common languages used in MBSE [16] are SysML [17] –based on OMG's[7] UML [18] standard, and EAST-ADL [19]. ARCADIA [20], IDEF [21], and OPM [22] consist both MBSE languages and methodologies. In terms of tools, IBM's Rhapsody [23] and Magic Draw's Cameo [24]–both based on SysML, and Eclipses' open-source Capella [25]–based on the Arcadia method, are the most popular.

3.2. System Safety Assessment (SSA) and Model-Based Safety Assessment (MBSA)

System Safety Assessment (SSA) is an analysis which leads to the production of a series of documents reporting all the identified hazards for the system in question, and the complying with the safety requirements [26]. For each identified system Failure Condition (FC)–how the system could fail, characterised by their level of impact to the environment, humans and the system itself (for space systems: catastrophic, critical, major, minor/negligible) [27], an analysis shall be made, in order to identify the possible causes–and combination of, that can lead to these FCs.

A recently developed approach for performing SSA or Reliability, Availability, Maintainability and Safety (RAMS) analysis, is MBSA. It is a technique which models the system's structure and behavior in order to provide safety analysis results. In this approach, various SSA-related activities are based on formal models which contain system dysfunctional data. In order to perform dysfunctional analysis at system level, it is required to have a fundamental

[7]The Object Management Group® (OMG®) is an international, open membership, not-for-profit technology standards consortium, founded in 1989. OMG standards are driven by vendors, end-users, academic institutions and government agencies.

knowledge of (a) the nominal system behavior, limited to the scope and the level of abstraction useful for dysfunctional analysis, in particular the reconfiguration and protection systems defined in the SE model, and (b) the various ways the failures can occur and propagate inside the system [28].

Consequently, MBSA uses a formal model describing both the nominal system behavior and the possible faulty behaviors, to analyse combinations of faults and their consequences in terms of feared events, which affect operational availability (when a critical error occurs, the system is not available until the error is resolved). The resulting analytical models are manually built by safety engineers from the available SE information. These models can be used to compute quantitative and/or qualitative safety indicators. Several mathematical formalisms are used to support the computations, mainly Markov chains [29], Petri nets [30] or Finite State Machines (FSMs) [31], each with different variants. These types of underlying formalisms are important for the qualification of computation tools.

As in MBSE, a number of modelling languages supporting MBSA were developed, such as *AltaRica* (AltaRica LaBRI [32], AltaRica DataFlow [33], AltaRica 3.0 [34]), *Figaro* [35], *SAML* [36], *HiP-HOPS* [37], *Component Fault Trees* [38], *Generalized Stochastic Petri Nets* [39] and *Safety Architect* [40]. In our approach we use the AltaRica language, since it has has become a de-facto European industrial standard for MBSA [41]. The most mature industrial tools for MBSA based on AltaRica are Dassault's Cecilia OCAS [42] and Apsys' SimfiaNeo [43].

3.3. Behaviour Trees

BTs were first invented as a tool to build modular Artificial Intelligence (AI) in computer games. A known alternative to FSMs, BTs are meant to provide better modularity, scalability, extendibility, adaptability and reuse of code [44, 45, 46, 47, 48, 49, 50], and to be easier to understand for humans, thus allowing incremental functionality design and efficient testing. On the benefits of modularity, Bagnell et al. state that *"individual behaviors can be reused in the context of another higher-level behavior, without needing to specify how they relate to subsequent behaviors"* [51].

According to Colledanchise and Ögren, *"a Behavior Tree is a way to structure the switching between different tasks (assuming that an activity can somehow be broken down into reusable sub-activities called tasks) in an autonomous agent, such as a robot or a virtual entity in a computer game"* [52]. According to García et al., a Behavior Tree is *"a mathematical model of plan execution that allows composing tasks in a modular fashion through a set of nodes representing tasks and connections among them"* [53].

BTs are widely used in systems control since they provide an easy way to perform conditional state switching. However, the main reason we have chosen BTs as the means to construct the ODM is that they provide an answer to the question *"What is the system currently doing?"*, which is precisely the first question the operators ask themselves, both when the system is in nominal mode, as well as when an issue has been identified.

One of the benefits of BTs we have additionally identified is that their elements can be in the form of blackboxes and be furtherly developed when new information is available; that being during the design phase or the operations phase –in the case of telemetry data reception. In the following section we demonstrate that BTs are easy to integrate within the design process. Based on the MBSE and MBSA models, and with operational objectives, we can create a model that can then be used inside a monitoring tool, suited for diagnosis. In our modelling approach we use the

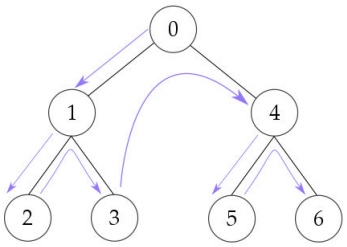


Figure 2: BTs' sequence execution.

classic formulation of BTs as described in [52], where BTs can be considered as a form of directed tree, where the flow amongst its nodes and edges is sequential in a left to right, depth first manner. The ticking execution sequence is illustrated in Figure 2.

BTs represent the possible system behaviours and their changes, in contrast to decision trees, which represent a logical formula, usually for automated decision purposes. BT nodes are visited at each time step (called "tick") during execution, starting from the root node. Parent nodes specify which children nodes must be visited, and in which order. Fallback and Sequence (parent) nodes define how the behaviour evolves when one of its child behaviour succeeds or fails. Action (child) nodes contain executable information. A classical BT formulation [52] is depicted in Figure 3.

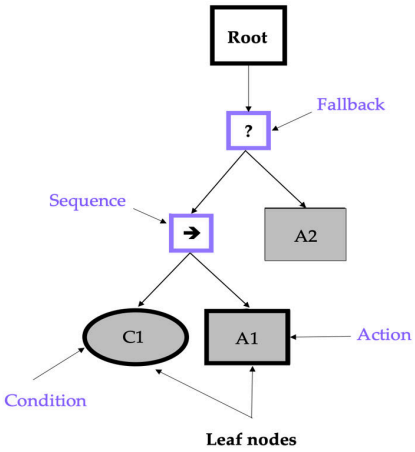


Figure 3: BTs' basic elements.

Regarding our BT implementation to construct the ODMs, we use the *PyTrees* (python implementation of BTs) library [54][55], which is relatively easy and intuitive to code with, so any engineer/scientist can build a BT model, with no particular programming background or developer skills. In our approach, BTs consist of *sequential tasks*: although PyTrees' ticking mechanism is compatible with parallel execution, we do not use this feature in our modelling approach, since no need has risen so far. Parallel execution in PyTrees is possible with the use of the the "parallel" composite node, which visits all its children simultaneously at each tick [56].

In terms of the trigger mechanism, we propose a *time-triggered* (vs event-driven [57] pg. 199) BT execution, in a way that the BT nodes are triggered in a time-increment manner and not by the order of the modelled events' occurrence. When representing structural system information, we often use time-triggered modelling –in contrast with dynamic system information, where event-driven modelling is preferred. Therefore, when dealing with *highly complex* systems, we prefer to represent them with time-triggered models that can periodically check for the occurrence (or not) of the modelled events, in an effort to reduce modelling complexity.

Regarding the *periodicity* of BTs, BT nodes are independent concurrent processes, that may or may not follow a mutual clock, e.g. they may terminate at any time. However, the BT is only active at every tick, which means BT events (starting a new action, mostly) only take place upon a tick. Ticks should be relatively fast with respect to the system's dynamics.

BTs can be modelled to be state-full i.e. having memory of all the visited nodes' state –state information can be either in the node type or the node instance (its source code), or state-less i.e. not "remembering" the state of multiple nodes, rather than saving only the state of the lastly triggered node. Clearly this option changes the behaviour of the composite nodes, while PyTrees can support both styles. By making the state-less modelling choice, one can affect the performance of the BT: state-less BTs offer less control over the workflow, thus this control must be implemented in the Action leaf nodes.

The BT implementation presented in this paper is using the default PyTrees' setting for Sequence and Fallback nodes, which is memory-full and memory-less, respectively. More specifically, the class syntax for the Sequence and Fallback nodes is respectively the following[56]:

```
class py_trees.composites.Sequence(name=Sequence, memory=True,  children=None)
class py_trees.composites.Selector(name=Selector, memory=False, children=None)
```

In regard to the *interruption* of BT behaviours, there may be many reasons to interrupt a node, some related to the BT, some completely unrelated. Action nodes are free to implement their internal interruption management

mechanism: it integrates easily in the BT paradigm. However, many libraries (including PyTrees) also include an interruption mechanism for BT-related interruptions (in Fallback nodes mostly). In PyTrees, the memory parameter of Selector nodes inhibits the interruption mechanism, e.g. with memory=True the Selector node does not try to restart "high priority" terminated children, so there are no interruptions. With memory=False, the Selector node always queries the high priority children first, so they can "restart" and interrupt a currently running low priority child.

A simple example would be a system that tries to buy an item, but if it does not have enough money, works to earn it. The version *without interruption* could be:

- Acquire item (Sequence)
  - Work until enough money
  - Buy item

In this version, the "Work until enough money" must internally check whether there is enough money, and terminate or perform work accordingly. The version *with interruption* could be:

- Acquire item (Selector(memory=False))
  - Try to buy (Sequence)
    - Has enough money (Condition)
    - Buy item
  - Work

In this tree, the "Has enough money" and "Work" activities are separated. Because of "memory=False", even when the "Work" node is running, the BT still executes the "Try to buy" node. When the latter fails, it goes back to the "Work" node. When the "Try to buy" node finally succeeds, the BT automatically interrupts the "Work" node.

Existing BT libraries are built for control purposes, and rely almost exclusively on the past behaviour statuses "success", "failure" and "running" to decide to start or interrupt other behaviours. In our approach, we require that the status of each behaviour can be enriched with health indicators, such as alarms, specific to each behaviour. This enriches the ticking mechanism with an alarm gathering mechanism, that can be used for User-Interface (UI) or automated reasoning purposes. This mechanism can be flexibly implemented in the PyTrees library through the use of visitors. This feature is illustrated in the next section.

4. Proposal

In traditional satellite system development process, the SE, SA and Operations stages occur sequentially –each stage commences after the precedent's completion, and evolve linearly in time –no recurring loop is connecting the three stages. There is however an exception regarding the SE and SA stages, the activities of which interlock during system design. Nevertheless, SE and SA activities finish long before the system operation phase starts. Therefore, if any issue is detected at operation time (which could have been avoided by a change in design), there will be no modification in the SE or SA documents and models, since the latter would significantly augment the project cost. This modification would thus be dismissed, even if the operational impact would be beneficial in the long term.

In our proposal, SE and SA documents and models are being concurrently developed with the ODM. Hence operational aspects can be taken into account in the SE and SA activities. This would not only increase the confidence of the system design in regards to its representation of the actual system, but also to the foreseeing of possible operational issues, which can be avoided by modifying the system design, early on the system life cycle [58]. This would also decrease the system development cost, while optimising system design.

SE is defined by an iterative system development process. All teams involved in the system design (SE/MBSE, RAMS/SA) are striving for continued improvement through constant dialogue and joint meetings at each iteration step –concept of concurrent design [59]. Our proposal consists in *incorporating the ODMs' team in the feedback loop*, so as to *increase the agility* of the system design process. This shall imply the involvement of more stakeholders (operators / ODM architects) in the design phase.

The proposed methodology implies the constant amelioration of system design at each iteration loop, which increases the efficiency of the system design activities. The co-conception and co-design of system models means that any modifications are made early on the life cycle development process; in contrast to making changes at the end, during the V&V and testing phase. Hence an increase of efficiency is achieved, by reducing the total time and cost of the design phase. The proposal thus not only is agile, but also improves the agility of current methodologies, widely used in the aerospace industry.

4.1. Illustration of the ODM construction process

As mentioned in section 3.3, one of the benefits of BTs is that its elements can be in the form of black boxes and be furtherly developed when new information is available. This way, BT models can be easily integrated within the design process. Based on the SE and SA models, and with operational objectives, we can create a model that can then be used inside a monitoring tool, suitable to diagnosis. An illustration of our proposal is presented in Table 1, where a juxtaposition of SE and SA information and the implementation into a BT is depicted.

As shown in Table 1, information can be added in the BT models gradually, when new information concerning the system is available. In this example we tackle the case of an Earth Observation satellite that shall fulfill its primary mission, which is to take photos of the area(s) of interest on Earth and transmit them back to the OC, as we can see in the first model iteration. The information coming from preliminary SE activities, here shown in the first column, is "Satellite must perform Earth Observation". This information can be translated in a BT model–as shown in the 3rd column, where a single action node "Observe Earth" is included in the tree, under the "Mission" node–here a Fallback node.

Throughout the system design development, the FMEA team can produce new system information, as shown in the second column, as for example, "Mission may fail". This information can be integrated in the BT model with the addition of a new action node ("Mission Fail"), on the right side of the "Observe Earth" node, which represents the system's nominal mode. This would mean that, if the "Observe Earth" node ticking returns "failure", the "Mission Fail" node will be ticked. If the "Observe Earth" node returns "success" or "running", the "Mission Fail" node will not be ticked.

In the third iteration, we assume the information "A mission fail can sometimes be mitigated by putting the Satellite on Standby Mode" is available by the SE team. This information can be integrated in the BT model by adding a new action node between the "Observe Earth" and "Mission Fail" node, which represents the "Standby" mode of the system. Consequently, if the "Observe Earth" node ticking returns "failure", the "Standby" node will be ticked. If the "Standby" node also returns "failure", the "Mission Fail" node is ticked. If the "Observe Earth" node returns "failure" and the "Standby" node returns "success" or "running" the "Mission Fail" node will not be ticked. The latter would signify–in case of system monitoring, that the system is currently in Standby mode.

The SE activities might then conclude that the Earth observation activity has three phases: capture photos, save photos, send photos to Operations Centre. The "Observe Earth" action node can then be turned into a Sequence node in the BT model, and have three children, namely "Capture photos", "Save photos" and "Send photos". This would mean that if the "Capture photos" activity is successful, then the "Save photos" activity can be performed. Similarly, if the "Save photos" activity is successful, the "Send photos" activity is able to be performed. If one of the activities cannot be performed, the "Observe Earth" activity will fail, and so the BT will return "failure" as well.

Lastly, we assume that the FMEA teams communicate to the SE teams that a fault leading to instrument overheating thus causing the mission to fail can be detected by the operator, if able to monitor the instrument's health status data during the "Capture photos" phase. The SE team can then conclude that adding temperature monitoring capabilities in the picture capturing instrument can help prevent system failure, and implement it in the system design. This new design modification shall also be added in the BT; this way the BT model is up-to-date with the SE and SA system models.

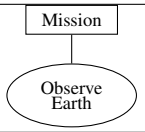
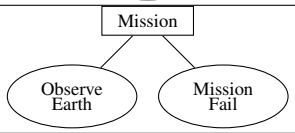
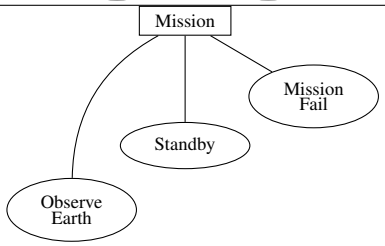
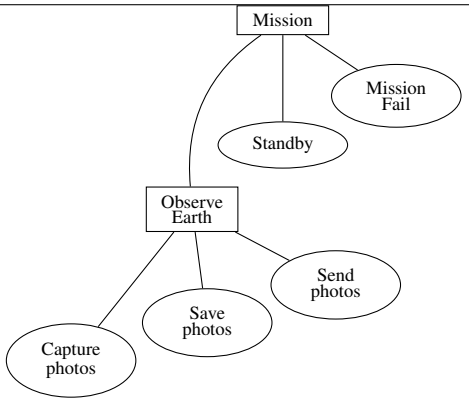
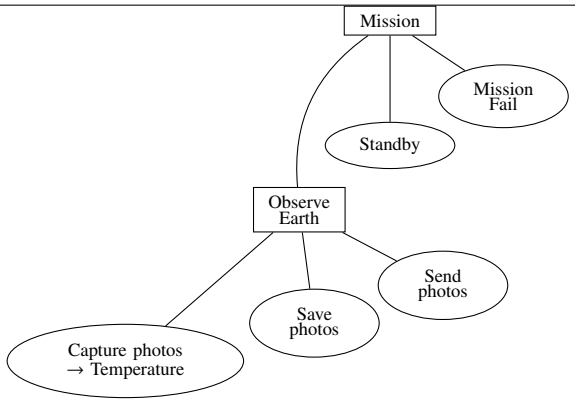
4.2. Exploitation of the ODM: Diagnostic Tools

The ODM is being created in parallel with the system functional and dysfunctional models and documents. How this activity ultimately improves the system's resilience is out of the scope of this paper, since there are many more steps in between. However, we can already point out that since the ODM is created at the design stage, it is possible to analyze it, identify design flaws that may impede operations, and derive new SE and SA requirements to address these flaws.

The most direct way to use the ODM is to build a UI component out of it, that displays the behaviour statuses in a hierarchical way, including possible alarms. The hierarchical nature of the model helps operators focus on relevant behaviours to quickly pinpoint problems and launch appropriate troubleshooting procedures.

Automated diagnosis techniques based on models [3] or data [4] can easily be integrated when they focus on particular components or functions of the system. Their output can be used to raise alarms in the associated behaviours,

Table 1: Illustration of the first iterations of a satellite design. At each iteration, the BT model can incorporate information coming from the system design process related activities, while remaining at each step a valid model.

Functional textual information	Dysfunctional textual information	Behaviour Tree
Satellite must perform Earth Observation		
	Mission may fail	
A mission fail can sometimes be mitigated by putting the Satellite on Standby Mode		
Earth observation activity has 3 phases: capture photos, save photos, send photos to OC		
	A fault leading to instrument overheating causing the mission to fail can be detected by operator monitoring health status data during “Capture photos” phase	
Add instrument temperature monitor for operator		

or alternatively the automated diagnosis tool can be considered a behaviour in itself, that raises alarms when detecting abnormal situations.

The ODM is a model that can be used to support automated reasoning techniques. A pattern of alarms in some behaviours could be automatically associated to a diagnosis, or a troubleshooting procedure. There are many ways to implement such reasoning (case-based reasoning, decision rules, constraint programming), that depend on aspects specific to the system and its environment.

Taking the example illustrated in Table 1 as a use case, we created a tool to display the status of the BT nodes called at each execution i.e. the BT nodes' response to the embedded ticking mechanism, which can be one of the three: "Success", "Failure" or "Running". Each execution is defined by an incremental time step $T(t_n)$, where $t = t_0, ..., t_{n-1}$, $n \in \mathbb{Z}$, $t_0 = 0$. We also implemented an "Alarm" feature, so that raised alarms associated with a behaviour can also also appear in the tool interface.

Table 2 shows an example interface of a diagnostic tool using the BT model of the last row and column of Table 1 as its ODM, after 3 time executions. In this case scenario, we can observe that in the first execution (T0), the status of the root node "Mission" is "Running", since the status of its first node "Observe Earth" is also "Running". As a Fallback node, the "Mission" node "inherits" the status of its first successful or running child node. After another "Running" interval, the status of the satellite mission function returns "Sucess", indicating the mission's successful completion (with or without errors in between).

The Sequence node "Observe Earth" needs all of its children nodes to succeed in order for it to take the status "Sucess"; on the contrary, if at least one of its children nodes returns "Failure", the "Observe Earth" function would equally fail. In this case, its first child node "Capture Photos", along with the temperature check of its camera sensor has successfully completed its function. Then, we can see that the following function "Save Photos" is "Running", meaning that the satellite is in the process of saving the photos taken. Thus the function "Observe Earth" is in process, hence the satellite is currently undergoing its principal mission.

Table 2: Diagnostic tool UI example.

Type of Node	Behaviour Name	Status T0	Status T1	Status T2
Fallback	Mission	Running	Running	Success
Sequence	Observe Earth	Running	Running	Success
Action	Capture Photos / Check Temperature	Success	Success	Success
Action	Save Photos	Running !Alarm!	Success	Success
Action	Send Photos	–	Running	Success
Action	Standby	–	–	–
Action	Mission Fail	–	–	–

However, we can see that the "Save Photos" function has raised an alarm, which would indicate to the operator that there might be a problem related to the On-Board Computer (OBC), the memory e.g. over-warmed electronics, etc. In such case, the operator shall have a troubleshooting sequence related to each error or alarm raised, which they would follow in order to establish a road-map leading to the fault resolution. In the next time interval (T1), we can safely assume that the satellite is in the process of sending the photos to the OC, while in the third (T2), that the satellite has successfully completed its mission for the moment.

If the operator had access solely to the information presented in the last column (T2), which only displays the results of the "Observe Earth" function –and not the underlying process, they would not be able to "see" the alarm raised during the photos saving function. A model-based diagnostic tool which uses an ODM with a BT formalism can have several benefits over traditional diagnostic procedures, such as the one illustrated in this use case scenario. Effectively, a comparison with current monitoring methods can only be made by using a real-life model of a complex satellite system, and feedback from operators. In this paper we present only a proof-of-concept.

5. Conclusion

Monitoring tools for diagnosis during system operations are, to this day, still lacking important information needed for the operators to perform their supervision and diagnostic tasks efficiently. They mainly rely on knowledge acquired through previous experience as well as engineers' and operators' know-how, rather than system design elements. For this reason we propose a methodology for the concurrent construction of a model dedicated to system monitoring for diagnosis, along with the system design (construction of SE and SA models). Our methodology would not only increase the agility of the system development process, but also the confidence on the system design by creating a monitoring tool which is equally based on its design information as well as ROE. Eventually, our proposal shall be validated by real-life operators, after having tested the model's integration in a specific tool, as part of a representative case study. We plan to evaluate it through feedback from industrial partners, research labs, and space agencies.

References

[1] W. Kim, S. Katipamula, A review of fault detection and diagnostics methods for building systems, Science and Technology for the Built Environment 24 (2018) 3–21.

[2] K. Djebko, F. Puppe, H. Kayal, Model-based fault detection and diagnosis for spacecraft with an application for the sonate triple cube nano-satellite, Aerospace 6 (2019).

[3] J. de Kleer, J. Kurien, Fundamentals of model-based diagnosis, IFAC Proceedings Volumes 36 (2003) 25–36. 5th IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes 2003, Washington DC, 9-11 June 1997.

[4] Y. Lei, B. Yang, X. Jiang, F. Jia, N. Li, A. K. Nandi, Applications of machine learning to machine fault diagnosis: A review and roadmap, Mechanical Systems and Signal Processing 138 (2020) 106587.

[5] A. Barua, K. Khorasani, Hierarchical fault diagnosis and fuzzy rule-based reasoning for satellites formation flight, IEEE Transactions on Aerospace and Electronic Systems 47 (2011) 2435–2456.

[6] P. Baldi, M. Blanke, P. Castaldi, N. Mimmo, S. Simani, Combined geometric and neural network approach to generic fault diagnosis in satellite reaction wheels, IFAC-PapersOnLine 48 (2015) 194–199. 9th IFAC Symposium on Fault Detection, Supervision andSafety for Technical Processes SAFEPROCESS 2015.

[7] C. Pittet, A. Falcoz, D. Henry, A model-based diagnosis method for transient and multiple faults of aocs thrusters, IFAC-PapersOnLine 49 (2016) 82–87. 20th IFAC Symposium on Automatic Control in AerospaceACA 2016.

[8] R. J. Patton, F. J. Uppal, S. Simani, B. Polle, Robust fdi applied to thruster faults of a satellite system, IFAC Proceedings Volumes 40 (2007) 1–6. 17th IFAC Symposium on Automatic Control in Aerospace.

[9] A. Valdes, K. Khorasani, A pulsed plasma thruster fault detection and isolation strategy for formation flying of satellites, Applied Soft Computing 10 (2010) 746–758.

[10] O. Lisagor, L. Sun, T. Kelly, The illusion of method: Challenges of model-based safety assessment, in: 28th international system safety conference (ISSC), p. Num Pages: 10.

[11] Y. Papadopoulos, M. Walker, D. Parker, E. Rüde, R. Hamann, A. Uhlig, U. Grätz, R. Lien, Engineering failure analysis and design optimisation with hip-hops, Engineering Failure Analysis 18 (2011) 590–608.

[12] Society of Automotive Engineers (SAE) International, Guidelines and methods for conducting the safety assessment process on civil airborne systems and equipment, SAE International, p. 331.

[13] O. Lisagor, T. Kelly, R. Niu, Model-based safety assessment: Review of the discipline and its challenges, in: The Proceedings of 2011 9th International Conference on Reliability, Maintainability and Safety, pp. 625–632.

[14] S. K. Ibrahim, A. Ahmed, M. A. E. Zeidan, I. E. Ziedan, Machine learning techniques for satellite fault diagnosis, Ain Shams Engineering Journal 11 (2020) 45–56.

[15] W. Wolf, Hardware-software co-design of embedded systems, Proceedings of the IEEE 82 (1994) 967–989.

[16] INCOSE - International Council on Systems Engineering, Guide to the Systems Engineering Body of Knowledge (SEBoK), `https://www.sebokwiki.org/`, 2021.

[17] Object Management Group (OMG), Systems Modeling Language (SysML), `https://www.omg.org/spec/SysML/Current`, 2021.

[18] Object Management Group (OMG), Unified Modeling Language (UML), `http://www.omg.org/spec/UML/current`, 2021.

[19] EAST-ADL Association, EAST Architecture Description Language (EAST-ADL), `https://www.east-adl.info/Specification.html`, 2021.

[20] Thales, ARChitecture Analysis & Design Integrated Approach (ARCADIA), `https://www.eclipse.org/capella/arcadia.html`, 2021.

[21] Knowledge Based Systems, Inc. (KBSI), Integration DEFinition (IDEF), `https://www.idef.com/`, 2021.

[22] International Organisation for Standardization (ISO), Object Process Methodology (OPM), `https://www.iso.org/standard/62274.html`, 2021.

[23] International Business Machines Corporation (IBM), IBM®Rational®Rhapsody®Architect for Systems Engineers, `https://www.ibm.com/products/systems-design-rhapsody`, 2021.

[24] No Magic, Cameo Systems Modeler, `http://www.nomagic.com/products/cameo-systems-modeler.html`, 2021.

[25] Eclipse Foundation, Eclipse Capella™, `https://www.eclipse.org/capella/`, 2021.

[26] M. Verrastro, I. Dimino, Chapter 21 - morphing devices: Safety, reliability, and certification prospects, in: A. Concilio, I. Dimino, L. Lecce, R. Pecora (Eds.), Morphing Wing Technologies, Butterworth-Heinemann, 2018, pp. 647–682.

[27] European Cooperation for Space Standardization, ECSS-Q-ST-40C Rev.1: Space Product Assurance - Safety, 2017.

[28] M. Machin, E. Saez, P. Virelizier, X. de Bossoreille, Modeling Functional Allocation in AltaRica to Support MBSE/MBSA Consistency, in: Y. Papadopoulos, K. Aslansefat, P. Katsaros, M. Bozzano (Eds.), Model-Based Safety and Assessment, Springer International Publishing, Cham, 2019, pp. 3–17.
[29] P.-A. Brameret, J.-M. Roussel, A. Rauzy, Preliminary system safety analysis with limited markov chain generation, IFAC Proceedings Volumes 46 (2013) 13–18. 4th IFAC Workshop on Dependable Control of Discrete Systems.
[30] N. Leveson, J. Stolzy, Safety analysis using petri nets, IEEE Transactions on Software Engineering SE-13 (1987) 386–397.
[31] X. Chen, J. Jiao, A fault propagation modeling method based on a finite state machine, in: 2017 Annual Reliability and Maintainability Symposium (RAMS), pp. 1–7.
[32] Laboratoire Bordelais de Recherche en Informatique (LaBRI), AltaRica Project - MEthods and Tools for AltaRica Language, `https://altarica.labri.fr/wp/`, last accessed January 2022.
[33] M. Boiteau, Y. Dutuit, A. Rauzy, J.-P. Signoret, The altarica data-flow language in use: modeling of production availability of a multi-state system, Reliability Engineering & System Safety 91 (2006) 747–755.
[34] T. Prosvirnova, M. Batteux, P.-A. Brameret, A. Cherfi, T. Friedlhuber, J.-M. Roussel, A. Rauzy, The altarica 3.0 project for model-based safety assessment, IFAC Proceedings Volumes 46 (2013) 127–132. 4th IFAC Workshop on Dependable Control of Discrete Systems.
[35] M. Bouissou, H. Bouhadana, M. Bannelier, N. Villatte, Knowledge modelling and reliability processing: Presentation of the figaro language and associated tools, IFAC Proceedings Volumes 24 (1991) 69–75.
[36] M. Lipaczewski, S. Struck, F. Ortmeier, Using tool-supported model based safety analysis – progress and experiences in saml development, in: Proceedings of IEEE International Symposium on High Assurance Systems Engineering, pp. 159–166.
[37] Y. Papadopoulos, J. McDermid, Hierarchically performed hazard origin and propagation studies, in: Proceedings of the 18th SAFECOMP International Conference, volume 1698, pp. 139–152.
[38] B. Kaiser, D. Schneider, R. Adler, D. Domis, F. Möhrle, A. Berres, M. Zeller, K. Höfig, M. Rothfelder, Advances in component fault trees, CRC Press, p. Num Pages: 9.
[39] G. Balbo, Introduction to Generalized Stochastic Petri Nets, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 83–131.
[40] ALL4TEC, Safety Architect - Model Based Safety Assessment software, `https://www.all4tec.com/en/safety-architect-fmeca-fta-sofware/`, last accessed January 2022.
[41] M. Bozzano, A. Cimatti, O. Lisagor, C. Mattarei, S. Mover, M. Roveri, S. Tonetta, Safety assessment of AltaRica models via symbolic model checking, Science of Computer Programming 98 (2015) 464–483.
[42] P. Bieber, J. P. Blanquart, G. Durrieu, D. Lesens, J. Lucotte, F. Tardy, M. Turin, C. Seguin, E. Conquet, Integration of formal fault analysis in ASSERT: Case studies and lessons learnt, in: Embedded Real Time Software and Systems (ERTS2008), Toulouse, France, p. Num Pages: 9.
[43] M. Machin, L. Sagaspe, X. de Bossoreille, Simfianeo, complex systems, yet simple safety, in: Embedded Real Time Software and Systems - ERTS 2018, p. 4.
[44] M. Colledanchise, A. Marzinotto, D. V. Dimarogonas, P. Oegren, The Advantages of Using Behavior Trees in Mult-Robot Systems, in: Proceedings of ISR 2016: 47st International Symposium on Robotics, pp. 1–8.
[45] P. Ogren, Increasing Modularity of UAV Control Systems using Computer Game Behavior Trees, in: AIAA Guidance, Navigation, and Control Conference, American Institute of Aeronautics and Astronautics, Minneapolis, Minnesota, 2012, pp. 1–8.
[46] M. Colledanchise, P. Ögren, How Behavior Trees modularize robustness and safety in hybrid systems, in: 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 1482–1488. ISSN: 2153-0866.
[47] M. Colledanchise, P. Ögren, How Behavior Trees Modularize Hybrid Control Systems and Generalize Sequential Behavior Compositions, the Subsumption Architecture, and Decision Trees, IEEE Transactions on Robotics 33 (2017) 372–389. Conference Name: IEEE Transactions on Robotics.
[48] M. Colledanchise, R. Parasuraman, P. Ögren, Learning of Behavior Trees for Autonomous Agents, IEEE Transactions on Games 11 (2019) 183–189. Conference Name: IEEE Transactions on Games.
[49] A. Klöckner, Interfacing Behavior Trees with the World Using Description Logic, in: AIAA Guidance, Navigation, and Control (GNC) Conference, Guidance, Navigation, and Control and Co-located Conferences, American Institute of Aeronautics and Astronautics, 2013, pp. 1–11.
[50] F. Rovida, B. Grossmann, V. Krüger, Extended behavior trees for quick definition of flexible robotic tasks, in: 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 6793–6800. ISSN: 2153-0866.
[51] J. A. Bagnell, F. Cavalcanti, L. Cui, T. Galluzzo, M. Hebert, M. Kazemi, M. Klingensmith, J. Libby, T. Y. Liu, N. Pollard, M. Pivtoraiko, J.-S. Valois, R. Zhu, An integrated system for autonomous robotics manipulation, in: 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 2955–2962.
[52] M. Colledanchise, P. Ögren, Behavior Trees in Robotics and AI: An Introduction, arXiv:1709.00084 [cs] (2018). ArXiv: 1709.00084.
[53] S. García, P. Pelliccione, C. Menghi, T. Berger, T. Bures, High-level mission specification for multiple robots, in: Proceedings of the 12th ACM SIGPLAN International Conference on Software Language Engineering, SLE 2019, Association for Computing Machinery, New York, NY, USA, 2019, pp. 127–140.
[54] Google JAX, PyTrees Library Documentation, `https://py-trees.readthedocs.io/en/devel/`, last accessed January 2022.
[55] Google JAX, PyTrees Github space, `https://github.com/splintered-reality/py_trees`, last accessed January 2022.
[56] Google JAX, PyTrees Library Documentation - Composites, `https://py-trees.readthedocs.io/en/devel/composites.html`, last accessed January 2022.
[57] R. Ghzouli, T. Berger, E. B. Johnsen, S. Dragule, A. Wasowski, Behavior trees in action: a study of robotics applications, in: Proceedings of the 13th ACM SIGPLAN International Conference on Software Language Engineering, ACM, Virtual USA, November 2020, pp. 196–209.
[58] W. K. Vaneman, The system of systems engineering and integration "vee" model, in: 2016 Annual IEEE Systems Conference (SysCon), pp. 1–7.
[59] S. FINGER, M. S. FOX, F. B. PRINZ, J. R. RINDERLE, Concurrent design, Applied Artificial Intelligence 6 (1992) 257–283.

Authors index