



HAL
open science

ERTS 2024 - 12th European Congress on Embedded Real Time Software and Systems

Mohamed Kaâniche, Philippe Cuenot, Kevin Delmas, Jean Marc Gabriel,
Adrien Gauffriau, Christophe Grand, Eric Jenn, Christine Rochange, Marie de
Roquemaurel

► **To cite this version:**

Mohamed Kaâniche, Philippe Cuenot, Kevin Delmas, Jean Marc Gabriel, Adrien Gauffriau, et al..
ERTS 2024 - 12th European Congress on Embedded Real Time Software and Systems. 12th European
Congress on Embedded Real Time Software and Systems (ERTS 2024), pp.1–489, 2024. hal-04643654

HAL Id: hal-04643654

<https://hal.science/hal-04643654v1>

Submitted on 10 Jul 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



EUROPEAN
CONGRESS

EMBEDDED
REAL TIME SYSTEMS

ERTS 2024

11-12 JUNE, TOULOUSE - FRANCE
DIAGORA CONGRESS CENTER

ORGANIZED BY



PROCEEDINGS

Editors

Mohamed Kaaniche	LAAS-CNRS
Philippe Cuenot	Continental Automotive
Marie de Roquemaurel	Airbus Defence & Space
Kevin Delmas	Onera
Jean-Marc Gabriel	Ampere Software Technology
Adrien Gauffriau	Airbus
Christophe Grand	Onera
Eric Jenn	IRT St Exupéry
Christine Rochange	IRIT

Contents

Program Committee	7
Tu.1.A – ML/AI Embedded	9
Real-Time Semantic Segmentation of Aerial Images Using an Embedded U-Net: A Comparison of CPU, GPU, and FPGA Workflows	11
Exploring Neural Network Architectures for Satellite Imagery on FPGA devices . .	23
Tu.1.B – Realtime System 1	33
Runtime Performance Evaluation of a Non-Preemptive Cooperative Multithreading Framework Through Tracing	35
A Novel Heuristic Framework for Offline IMA Schedule Generation for Multicore Platforms	45
Tu.1.C – Network	57
Specifying network switches using the P4 language: lessons learned	59
Yet another experience on TSN tools interoperability for critical embedded networks	67
Tu.2.Po – Poster overview	77
Acceleration of Embedded Reasoning in Symbolic AI	79
How to efficiently handle real world ECU traffic in MICROSAR Adaptive	83
Optimal PMP+OR onboard Controls for Multiple Electrified Automotive applications	89
Towards Compact Surface Languages for Specific Modelling Aspects in EAST-ADL	95
Development and Evaluation of a Prototyping Platform for the Simulation, Transmission, and Real-Time Analysis of Realistic AUTOSAR Security Event Traffic . .	99
Timing Architecture Model for Embedded Systems Anomaly Detection	103
Signal integrity challenges of complex high-speed serial links up to 25 Gbps in an aeronautic environment	109
The Security Analysis of a BLE Connected Health Device	113
Towards Designing a Cybersecurity Testbed for Critical Industrial Control Systems	119
Predictive Maintenance and Control of Memory for Availability in Safety Systems	125
Integrating operator’s cognitive profile for dynamic and human-centric adaptation of industrial processes	129
Tu.3.A – ML/AI Certification	133
Perspectives on ML Safety Assurance	135
A study of an ACAS-Xu exact implementation using ED-324/ARP6983	147
On the Feasibility of EASA Learning Assurance Objectives for Machine Learning Components	159

Tu.3.B – Realtime System 2	171
Performance and confidence in feasibility analysis of real-time multi-core distributed systems	173
Towards the Certification of Hybrid Architectures: Analysing Interference on Hardware Accelerators through PML	183
Tu.3.C – Hardware Security	195
Onchip Traffic Injection to Counteract Timing Side-Channel Attacks	197
Approach for High-Performance Random Number Generators for Critical Systems	207
Considering the Aeronautics Cyber-Security Standards for Multi-Core Platforms .	217
Tu.4.A – Assurance Case for ML	229
Assurance Cases to face the complexity of ML-based systems verification	231
Uncertainty in Assurance Case Template for Machine Learning	243
Tu.4.B – Logical Execution Time	253
Separation of functional and time interferences concerns for efficient AMC 20-193 compliance	255
Reducing End-to-End Latencies of Multi-Rate Cause-Effect Chains in Safety Critical Embedded Systems	265
Tu.4.C – Security	275
Security by Default - CHERI ISA Extensions Coupled with a Security-Enhanced Ada Runtime	277
Problems and New Approaches for Crypto-Agility in Operational Technology . . .	291
We.1.A – ML/AI for Critical System 1	303
Software-Only Semantic Diverse Redundancy for High-Integrity AI-Based Functionalities	305
Formal description of ML models for unambiguous implementation	315
We.1.B – Realtime Interference	325
A Refinement Method for Interference Analysis using the PHYLOG Modeling Language	327
Kryptonite++: Localizing Program Interference on Multi-core Embedded Systems	337
We.1.C – Benchmarking	347
An Evaluation Bench for the Exploration of Machine Learning Deployment Solutions on Embedded Platforms	349
Multi-core WCET Analysis Using Non-Intrusive Continuous Observation	359
We.4.A – Autonomous System & Digital Twins	369
Digital twin for embedded software. State of art in industry and deployment at Renault Group for powertrain	371
Towards safe obstacle detection for autonomous train operation: Combining track and switch detection neural networks for robust railway ego track detection	379
Partially trustworthy action planning thanks to an easily certified plan validator .	389
We.4.B – Testing	399
Design by contract formal verification for automotive embedded software robustness	401
Automated Test Suite Augmentation using Language Models: Applying RAG to Improve Robustness Verification	411
Mixing tests and mathematical analysis - A launcher use case	423

We.4.C – Model Driven Development	433
Large legacy systems design maintainability through modeling	435
Coupling optimization using Design Structure Matrices (DSM) and Genetic Algorithm	445
Specializing SysMLv2 for Real-Time Safety- Critical Systems – an Experiment with AADLv2	455
We.5.A – ML/AI for Critical System 2	465
Certified ML Object Detection for Surveillance Missions	467
How to design a dataset compliant with a ML-based system ODD?	477
Authors index	487

Program Committee

Ahiad, Samia	VALEO
Anguenot, Yves	Aerospace Valley
Armengaud, Eric	Armengaud Innovate GmbH
Barrilado, Andres	NXP
Baufreton, Philippe	Safran Electronics & Defense
Belmonte, Fabien	Alstom Transport SA
Boyer, Marc	ONERA
Braband, Jens	Siemens AG
Bruguier, Florent	LIRMM
Cazorla, Francisco J	Barcelona Supercomputing Center
Christofi, Nikolena	IRT Saint Exupery
Claraz, Denis	Vitesco Technologies France SAS
Cuenot, Philippe	Continental
Delmas, Kevin	ONERA
Dreiseitel, Stefan	Draexlmaier
Faucou, Sebastien	Université de Nantes
Florent, Meurville	Valeo
Frezouls, Benoit	CNES
Gabriel, Jean-Marc	Renault
Gallina, Barbara	Mälardalen University
Gauffriau, Adrien	Airbus
Gimenez, Pierre-François	CentraleSupélec
Gogniat, Guy	Université de Bretagne Sud - UEB
Grand, Christophe	ONERA
Guerin, Joris	Université de Montpellier
Guiochet, Jérémie	LAAS-CNRS
Habli, Ibrahim	University of York
Heckmann, Reinhold	AbsInt GmbH
Jan, Mathieu	CEA LIST
Jean-Louis, Boulanger	certifer
Jenn, Eric	Thales Avionics
Johnson, Chris	Queen's University Belfast
Kaaniche, Mohamed	LAAS-CNRS
Le Calvez, Gilles	VALEO
Lecomte, Thierry	CLEARSY
Lecote, Bertrand	Airbus Operations SAS
Lednot, Emmanuel	THALES Research & Technology
Lonn, Henrik	Volvo Group

Mader, Ralph	Vitesco Technologies GmbH
Maillet-Contoz, Laurent	STMicroelectronics
Malenfant, Jacques	Sorbonne Université – LIP6
Mamalet, Franck	IRT Saint Exupery
Mekki-Mokhtar, Amina	ANSYS
Morgan, Benoît	IRIT
Mraidha, Chokri	CEA LIST
Métayer, Natacha	Institut VEDECOM
Navet, Nicolas	University of Luxembourg
Niemetz, Michael	OTH Regensburg
Pagetti, Claire	ONERA
Pautet, Laurent	ENST
Picard, Celia	ENAC
Pinot, Frédéric	Hitachi rail STS
Pons, Philippe	Aerospace Valley
Prof. Dr. Mottok, Juergen	LaS ³ , OTH Regensburg
Quere, Philippe	Stellantis
Rochange, Christine	IRIT - Université de Toulouse
Ruiz, Jose	AdaCore
Shagdar, Oyunchimeg	Renault
Stea, Giovanni	University of Pisa
Thomas, Carsten	HTW Berlin
Totel, Eric	Supelec
Troubitsyna, Elena	KTH
Verdier, Damien	EasyMile
Vigouroux, David	IRT Saint-Exupery
Voget, Stefan	Continental Automotive GmbH
Warns, Timo	Airbus
Wartel, Franck	AIRBUS Defence and Space
de Roquemaurel, Marie	Airbus Defence & Space

Session Tu.1.A
ML/AI Embedded

Tuesday 11th June

11:30

–

Auditorium

Real-Time Semantic Segmentation of Aerial Images Using an Embedded U-Net: A Comparison of CPU, GPU, and FPGA Workflows

Julien Posso*, Hugo Kieffer^{†‡}, Nicolas Menga^{†§}, Omar Hlimi[†], Sébastien Tarris^{†‡},
Hubert Guerard[¶], Guy Bois^{*¶}, Matthieu Couderc^{†§}, Eric Jenn[†]

* École Polytechnique de Montréal

† IRT Saint Exupéry

‡ Viveris Technologies

§ Airbus Defence and Space

¶ Space Codesign Systems

Abstract—This study introduces a lightweight U-Net model optimized for real-time semantic segmentation of aerial images, targeting the efficient utilization of Commercial Off-The-Shelf (COTS) embedded computing platforms. We maintain the accuracy of the U-Net on a real-world dataset while significantly reducing the model’s parameters and Multiply-Accumulate (MAC) operations by a factor of 16. Our comprehensive analysis covers three hardware platforms (CPU, GPU, and FPGA) and five different toolchains (TVM, FINN, Vitis AI, TensorFlow GPU, and cuDNN), assessing each on metrics such as latency, power consumption, memory footprint, energy efficiency, and FPGA resource usage. The results highlight the trade-offs between these platforms and toolchains, with a particular focus on the practical deployment challenges in real-world applications. Our findings demonstrate that while the FPGA with Vitis AI emerges as the superior choice due to its performance, energy efficiency, and maturity, it requires specialized hardware knowledge, emphasizing the need for a balanced approach in selecting embedded computing solutions for semantic segmentation tasks.

Index Terms—Deep Learning, Neural Networks, Computer Vision, Semantic Segmentation, Inference, Embedded Systems, Aerospace, CPU, GPU, FPGA, MPSoC

I. INTRODUCTION

The advent of deep neural networks, especially Convolutional Neural Networks (CNNs), has revolutionized computer vision [13], introducing advanced capabilities for embedded systems in areas such as autonomous navigation [29] and earth observation [7], [16], [20]. Efficient hardware acceleration is vital for leveraging this technology, involving CPUs, GPUs, ASICs, FPGAs [27], and neural network compilers that bridge the gap between high-level Python libraries and hardware accelerators [5]. These topics have recently gained significant attention, as discussed in Section II. However, prior research has predominantly focused on image classification networks, specific hardware platforms, and compilers.

In this article, we present a pioneering, comprehensive, transversal study on the optimized implementation of image segmentation tasks for UAVs (Unmanned Aerial Vehicles) and satellites: specifically, the semantic segmentation of aerial images. We have enhanced a U-Net model for improved

embeddability, reducing its parameters and MAC (Multiply-Accumulate) operations by a factor of 16 while maintaining accuracy. We evaluate and compare five implementation schemes (workflows) across three COTS (Commercial Off-The-Shelf) embedded computing platforms (GPU, CPU, FPGA), assessing them using metrics such as IoU (Intersection over Union), accuracy, power, throughput, energy efficiency, and memory footprint. We also consider engineering metrics like workflow maturity, usability, documentation, and community support. This study addresses key practical challenges and provides valuable insights for those looking to integrate deep neural networks into real-world applications.

The structure of this paper is organized as follows: Section II reviews the literature pertinent to our research, providing foundational context. Section III details our computer vision task, specifically focusing on the semantic segmentation of aerial images using a lightweight U-Net to enhance its suitability for embedded systems. Section IV discusses the embedded computing platforms and examines the five workflows employed for implementing the neural network on these platforms. Section V synthesizes the main results, compares the workflows, and discusses the limitations of our study. Finally, Section VI summarizes the study, highlighting the effectiveness of the workflows and the suitability of the hardware selections for our specific application domain.

II. RELATED WORKS

The quest for hardware accelerators is crucial for enabling real-time neural network inference. Central to this acceleration are technologies such as CPUs, GPUs, ASICs, and FPGAs [27]. The role of compilers in bridging the gap between hardware capabilities and neural network performance is well-documented [5]. Additionally, there is a noticeable shift in the embedded sector towards the adoption of Commercial Off-The-Shelf (COTS) computers [24].

Zhao et al. [35] and Li et al. [14] meticulously review prevalent neural network compilers, including TVM, focusing

primarily on their optimization mechanisms and their impact on the speedup of state-of-the-art image classification networks. Xing et al. [33] provide an in-depth analysis of throughput, energy efficiency, and user-friendliness of six compilers, including TVM, aligning closely with our research. However, their analysis is confined to image classification networks such as ResNet50 and SqueezeNet, and they overlook potential quality degradation in neural network output due to the compilation and optimization processes.

Mittal et al. [18] provide a detailed survey of Nvidia Jetson GPUs within the context of embedded systems, including their application in semantic segmentation networks. Abdelouahab et al. [1] and Guo et al. [9] review designs for neural network accelerators, with a particular emphasis on enhancing FPGA inference within image classification networks. Reuther et al. [27] offer a comprehensive yet succinct survey of machine learning accelerators, focusing on performance and energy efficiency. Peccerillo et al. [26] examine approximately 100 accelerators, exploring their diverse workflows.

Comparative studies on FPGA and GPU inference performance and energy efficiency for standard image classification networks are detailed by Nurvitadhi et al. [22]. Feng [8] compares FPGA and GPU inference, focusing solely on semantic segmentation networks on GPUs, notably excluding FPGAs. Li et al. [15] highlight a performance comparison between FPGA and GPU inferences of binarized neural networks, revealing a trade-off between throughput and energy efficiency.

In the embedded domain, Dimitrovski et al. [7] review neural network architectures for aerial imagery, primarily focusing on image classification accuracy while neglecting real-time inference capabilities. Wang et al. [31] and Wu et al. [32] propose new neural network architectures for real-time semantic segmentation of aerial images, yet their deployment on embedded hardware remains unexplored. Moreover, existing research often limits its focus to single COTS platforms and toolchains for real-time inference [18], [30].

The literature exhibits significant limitations, predominantly focusing on image classification networks, which are less relevant for earth observation via UAVs and satellites. Furthermore, the research largely relies on benchmark datasets (e.g., ImageNet) and often restricts its experimental scope to single COTS platforms and toolchains. Studies encompassing multiple hardware targets or compilers are typically classified as surveys rather than experimental research.

In contrast, our research stands out due to its comprehensive approach in several key areas:

- A focus on semantic segmentation, an essential task for analyzing imagery from UAVs and satellites, diverging from the common focus on image classification.
- The adoption of a U-Net architecture for image segmentation, which includes both down-sampling (encoder) and up-sampling (decoder) paths, contrasting with the solely down-sampling nature of image classification networks. This approach exposes unique challenges in certain workflows that previous studies have not addressed.

- The utilization of the *Inria Aerial Image Labeling Dataset* for real-world applications, moving away from the conventional use of benchmark datasets like ImageNet.
- A comprehensive evaluation involving multiple workflows and hardware targets, providing a holistic view of their performance and limitations.

III. EMBEDDABLE U-NET-BASED SEMANTIC SEGMENTATION OF AERIAL IMAGES

A. Semantic Segmentation of Aerial Images

Our research is situated within the context of earth observation, focusing primarily on two application domains: satellites and UAVs. These platforms are pivotal in acquiring high-resolution terrestrial imagery, offering spatial resolutions ranging from 0.2 to 10 meters, which are critical for numerous remote sensing applications [7], [16], [20]. The primary limitation lies in the downlink capacity, as satellites and UAVs lack the capability to transmit all captured images to ground stations. Consequently, on-board analysis becomes essential to ensure that only relevant data is transmitted to Earth, optimizing both bandwidth and data relevance [10].

In this context, semantic segmentation is indispensable as it enables precise on-board analysis of the high-resolution imagery acquired by satellites and UAVs. We employ the *Inria Aerial Image Labeling Dataset* provided by Inria, renowned for its utility in benchmarking the generalization capabilities of semantic segmentation methodologies [17]. This dataset includes 180 colored satellite photographs, each measuring 5000x5000 pixels (25 Megapixels). The primary task of the dataset involves semantic segmentation, which entails classifying each pixel of an input image into a specific category; in our case, this means distinguishing every pixel as either 'building' or 'not building'. This classification results in a segmentation map. Figure 3 illustrates this process. To optimize for training and model embeddability, we dissect these images into smaller segments of 256x256 pixels, maintaining slight overlaps. These segments are subsequently merged to reconstruct the original 5000x5000 segmentation map post-inference.

B. U-Net Architecture

We selected a U-Net architecture for our workflow comparison. The U-Net [28], initially proposed for biomedical image segmentation, has since become a widespread neural network architecture. It features a low number of parameters, a small memory footprint, and fewer MAC operations compared to other semantic segmentation networks, while still maintaining high accuracy. Additionally, it is designed to be trained with a limited amount of data, a common scenario in the embedded domain. These characteristics make the U-Net an ideal candidate for an embedded neural network.

However, we modified the U-Net to enhance its embeddability. We trained multiple versions of the U-Net, varying the number of layers and channels per layer. Figure 1 demonstrates the necessity of this process in an embedded context. In the down-sampling path of the U-Net, each block contains two

convolutional layers and one max pooling layer. Similarly, in the up-sampling path, each block includes one transposed convolution and two convolutional layers. We adjusted the number of channels on each layer from 1/32 to 1/2 of the original U-Net and varied the number of blocks (*i.e. the number of layers*) from one to four, while maintaining symmetry between the down-sampling and up-sampling paths of the U-Net.

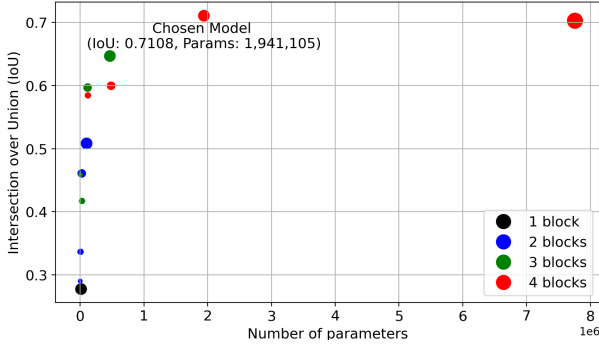


Fig. 1: IoU on the validation set vs. the number of parameters of the U-Net. Circle size represents the number of channels.

We preserved the core structure of the original U-Net, which consists of four blocks, but reduced the number of channels per layer to one-fourth of the original. This adjustment significantly decreased the number of parameters (from 31 million to 1.9 million) and MAC (Multiply-Accumulate) operations (from 55 billion to 3.4 billion) required to process a single 256x256 image, while still maintaining accuracy on the *Inria Aerial Image Labeling Dataset*. The number of parameters and MAC operations is proportional to the square of the number of channels, underscoring the importance of adapting neural network architectures to new datasets, especially in embedded contexts. Figure 2 provides a detailed view of the U-Net architecture, showing the distribution of MAC operations and the number of parameters across the down-sampling (encoder), middle, and up-sampling (decoder) paths. Notably, the two middle layers of the U-Net contain almost half of the parameters, while the majority of MAC operations occur in the up-sampling path. This path is crucial for reconstructing the feature maps back to the original image size, explaining the higher number of MAC operations required for accurately generating the output segmentation map. The inclusion of transposed convolutions in the up-sampling path, not present in state-of-the-art image classification neural networks, introduces unique challenges in some workflows.

C. Training Details

We trained our U-Net on an Nvidia RTX 3070 GPU, using Keras and TensorFlow 2.6, on the *Inria Aerial Image Labeling Dataset*, as detailed in Section III-A. Training began with random initial Float32 weights and utilized the Adam optimization algorithm [11] with TensorFlow’s default parameters and a learning rate of 1.0×10^{-4} over 108 epochs. Training

was halted after 15 epochs without improvement in the Intersection over Union (IoU) computed on the validation set. We employed the Binary Cross Entropy (BCE) loss function, which is effective for binary segmentation tasks. To enhance the model’s robustness and reduce sensitivity to overfitting, we normalized the input images to a range between 0 and 1 and applied data augmentation techniques using OpenCV 2.5. These techniques included random rotations (multiples of 90 degrees) and horizontal and vertical flipping.

D. U-Net Evaluation

Table I presents the evaluation of our lightweight U-Net, compared with the same data, task, and evaluation metrics used by the Inria team [17]: the IoU of the building class and pixel accuracy. The Inria team employed a FCN (Fully Convolutional Network) followed by a MLP (Multi-Layer Perceptron). Additionally, they discuss the general training process but lack in-depth technical specifics about the architecture configurations, such as the number of parameters and layers. Nevertheless, the evaluations demonstrate that our lightweight U-Net outperforms the Inria team’s neural network. The lightweight U-Net serves as a baseline for evaluating the five workflows explored in this paper.

TABLE I: Evaluation metrics of our lightweight U-Net on the validation set

Model	IoU	Accuracy
Lightweight U-Net (ours)	0.7108	0.9546
FCN + MLP (Inria) [17]	0.6467	0.9442

Figure 3 shows an example of our U-Net’s prediction quality compared to the ground truth on a 256x256 image. The buildings are generally well-predicted by the neural network, even if the contours of the predicted buildings are somewhat blurred, a similar effect was noticed in the original Inria publication [17].

IV. PLATFORMS AND WORKFLOWS

A. Platforms for Real-Time Inference

We selected two COTS platforms, specifically designed for embedded applications, to deploy our U-Net model. The Xilinx Zynq UltraScale+ MPSoC, equipped with four ARM Cortex-A53 processor cores and programmable logic (commonly referred to as an FPGA), has proven effective in both UAV [12] and space domains [24]. For our implementation, we utilized three Xilinx Zynq UltraScale+ boards—Ultra96, ZCU102, and ZCU104—each equipped with the same processor but featuring varying FPGA sizes, to host the hardware accelerators. Nvidia Jetson platforms have also emerged as strong contenders for real-time inference of neural network-based vision algorithms, demonstrating applicability in UAV [30] and space domains [2]. Specifically, we employed the Nvidia Jetson AGX Xavier System on Module, which boasts eight ARM Cortex-A57 processor cores and an integrated GPU, enhancing the acceleration of neural network inference.

Path	Block	Layer type	Input channels	Output channels	Kernel size	Output width	Number of parameters	Number of MAC (million)	Number of parameters	Number of MAC (million)	Number of parameters (%)	Number of MAC (%)					
Down	D1	Convolution	3	16	3	256	448	29.4	293,520	862.7	15.1%	25.0%					
		Convolution	16	16	3	256	2,320	152.0									
	D2	Convolution	16	32	3	128	4,640	76.0									
		Convolution	32	32	3	128	9,248	151.5									
	D3	Convolution	32	64	3	64	18,496	75.8									
		Convolution	64	64	3	64	36,928	151.3									
	D4	Convolution	64	128	3	32	73,856	75.6									
		Convolution	128	128	3	32	147,584	151.1									
Middle	M1	Convolution	128	256	3	16	295,168	75.6	885,248.0	226.6	45.6%	6.6%					
		Convolution	256	256	3	16	590,080	151.1									
Up	U1	Transpose conv.	256	128	2	32	131,200	134.3	762,320	2,354.7	39.3%	68.3%					
		Convolution	256	128	3	32	295,040	302.1									
		Convolution	128	128	3	32	147,584	151.1									
	U2	Transpose conv.	128	64	2	64	32,832	134.5									
		Convolution	128	64	3	64	73,792	302.3									
		Convolution	64	64	3	64	36,928	151.3									
	U3	Transpose conv.	64	32	2	128	8,224	134.7									
		Convolution	64	32	3	128	18,464	302.5									
		Convolution	32	32	3	128	9,248	151.5									
	U4	Transpose conv.	32	16	2	256	2,064	135.3									
		Convolution	32	16	3	256	4,624	303.0									
		Convolution	16	16	3	256	2,320	152.0									
	Final convolution			16	1	1	256	17					1.1	17	1.1	0.0%	0.0%
	Total							1,941,105					3,445.2	1,941,105	3,445.2	100.0%	100.0%

Fig. 2: Detailed architecture of the U-Net model

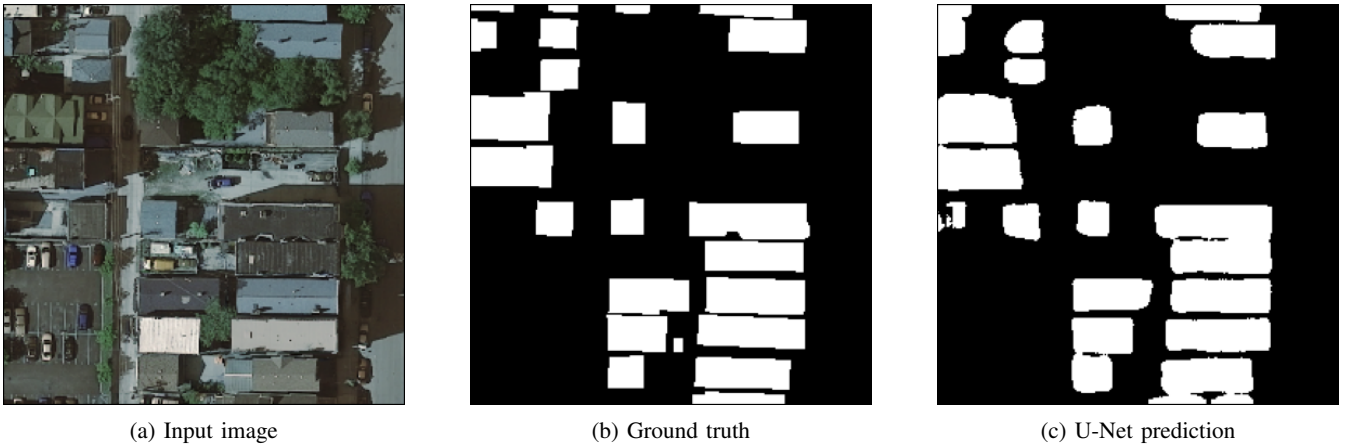


Fig. 3: Qualitative evaluation of our Float32 Keras lightweight U-Net on a 256x256 image of the validation set

B. Workflows Overview

We evaluated various workflows to implement our U-Net on CPU, GPU, and FPGA platforms. On the GPU side, we first assessed the straightforward TensorFlow implementation, comparing it with the more complex but optimized Nvidia cuDNN library to understand the trade-offs between ease of use and performance. For the CPU, we utilized TVM, which is renowned for supporting major Python frameworks and offering the best speedup among neural network compilers [5], further enhanced by its auto-scheduling feature. For the

FPGA, we explored both the open-source FINN framework and Xilinx’s commercial DPU within the Vitis-AI toolchain. Although Vitis-AI is considered more mature, FINN offers experimental yet highly optimized options for creating optimized dataflow implementations [3]. The following sections will delve into the details of each workflow.

C. GPU Implementation with TensorFlow

1) *Workflow Overview*: Figure 4 presents the workflow used to deploy our model on the Nvidia Jetson AGX Xavier

using TensorFlow 2.6. This workflow is straightforward, starting with the training of a Float32 model using Keras, serving as our baseline for evaluating GPU workflows. Notably, the model remains in Float32 format throughout, since quantization is only available in TensorFlow Lite. Our aim was to evaluate the most direct method for deploying a neural network on a Jetson GPU. Furthermore, the Jetson GPU efficiently processes Float32 operations on its CUDA (Compute Unified Device Architecture) cores. The trained model is exported in HDF5 format and then loaded onto the Nvidia Jetson AGX Xavier development kit. Onboard inference is conducted through a Python script, representing the simplest deployment method on the Nvidia Jetson platform, which operates on a Linux-based system with a Python stack, including TensorFlow.

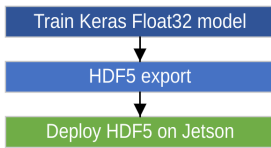


Fig. 4: GPU workflow from Keras/TensorFlow training to Nvidia Jetson AGX Xavier inference using TensorFlow

2) *Quantitative Evaluation*: Table II presents the evaluation metrics measured on the validation set throughout the TensorFlow workflow. The first row shows the results following training with Keras and TensorFlow in a Float32 format. Subsequent rows detail these metrics when the model is deployed on an Nvidia Jetson AGX Xavier board. The consistency observed between the standard computing environment and the embedded deployment is expected because the underlying model remains unchanged between the training and deployment stages.

TABLE II: Evaluation metrics along the TensorFlow workflow

Model	IoU	Accuracy
Float32 Keras	0.7062	0.9594
Jetson implementation	0.7062	0.9594

Table III summarizes the implementation metrics measured on the Jetson AGX Xavier. In this experiment, we varied the batch size to analyze its impact on the implementation metrics. Increasing the batch size to eight proved beneficial for improving throughput and energy efficiency while maintaining a reasonable memory footprint. The memory footprint includes the space needed for the model weights and activation functions, the batch of images, and additional Python libraries such as TensorFlow. Further increases in batch size did not yield significant benefits and resulted in an increased memory footprint, making a batch size of eight an optimal trade-off. A batch size of one is deemed beneficial only when memory footprint or latency is prioritized over throughput or energy efficiency. During the experiments, we noticed some variability in execution time, particularly for the first inference. The first inference with a batch of eight images took 238

milliseconds, while the subsequent inferences averaged around 107 milliseconds (plus or minus 10 milliseconds). The table also reports the average throughput for the entire validation set. The observed variability was consistent across all batch sizes, highlighting the importance of also considering the Worst Case Execution Time (WCET) in embedded systems where it is a critical factor.

TABLE III: Implementation metrics on the Nvidia Jetson AGX Xavier with TensorFlow

Batch size	Throughput (FPS)	Power (W)	Energy efficiency (mJ/image)	Memory (GB)
1	61.6	13.65	221.6	1.7
8	74.6	14.56	195.2	2.2
16	78.6	14.56	185.2	5.05
32	75.8	14.56	192.1	5.3

3) *Qualitative Evaluation*: The TensorFlow workflow targeting the Nvidia Jetson GPU is mature, straightforward, and well-documented, supported by an active community with numerous users, examples, and online tutorials. However, optimization of the neural network is limited within this framework. The high memory footprint presents significant concerns for embedded systems, which are often resource-limited compared to typical desktop or server environments. Furthermore, this high memory footprint could impact performance, energy efficiency, cost, and system stability, especially when the hardware is required to manage multiple applications simultaneously.

D. GPU Implementation with CuDNN

1) *Workflow Overview*: Figure 5 illustrates the workflow used to deploy our model on the Nvidia Jetson AGX Xavier utilizing the Nvidia cuDNN 8.4.1 library. Initially, we train a Float32 version of the model using Keras and export the trained parameters. Similar to the previous workflow, the model remains in Float32 format because quantization is only supported in TensorFlow Lite. The Jetson GPU is capable of efficiently processing Float32 operations on its CUDA cores. Subsequently, the neural network must be manually implemented in C++ with calls to the cuDNN library to execute operations on the GPU. The neural network is then cross-compiled for an ARM target using g++ and NVCC (Nvidia CUDA Compiler), resulting in an executable that is deployed on the Nvidia Jetson AGX Xavier, which operates a Linux-based system with the cuDNN library installed.

2) *Quantitative Evaluation*: During the evaluation, we encountered challenges, particularly due to the lack of a cuDNN implementation for the transposed convolution in the up-sampling path of the U-Net, as well as for the nearest neighbor up-sampling operation. A feasible solution could have been to implement these layers in a custom CUDA program; however, due to limited time and inadequate support on the Nvidia forum, this approach was not viable. We successfully implemented the down-sampling path and the middle convolution of the U-Net using cuDNN. The implementation’s accuracy was

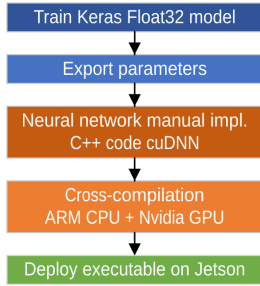


Fig. 5: GPU workflow from Keras/TensorFlow training to Nvidia Jetson AGX Xavier inference with cuDNN

validated by comparing the intermediate tensor outputs from the middle convolution produced by cuDNN with those from TensorFlow, finding them equivalent within an absolute tolerance of $1e-8$. Thus, we conclude that the cuDNN workflow is unlikely to alter the evaluation metrics significantly.

Table IV presents the evaluation metrics measured on the validation set for the implemented down-sampling path and middle convolution of the U-Net using cuDNN. We estimated the full U-Net implementation performance by considering that the down-sampling path and middle convolutions comprise 31.6% of the MAC operations, and we scaled the measured latency accordingly to estimate the total latency. Similarly, since these components represent 60.7% of the parameters and intermediate feature maps, we adjusted the memory footprint to estimate the total memory usage. These estimates should be interpreted with caution.

TABLE IV: Measured and estimated implementation metrics on the Nvidia Jetson AGX Xavier with cuDNN

Model	Latency (ms)	Power (W)	Energy efficiency (mJ/image)	Memory (MB)
Partial U-Net (measured)	5.82	5.61	32.6	795
U-Net (estimated)	18.4	5.61	103.3	1310

3) *Qualitative Evaluation*: The cuDNN workflow for targeting Nvidia-embedded GPUs is mature yet intricate. cuDNN is primarily designed for developers of deep neural network (DNN) frameworks such as PyTorch or TensorFlow [4]. Consequently, it is more complex than other libraries and lacks extensive examples. Additionally, the absence of certain neural network layers necessitates a proficiency in CUDA programming, which is considerably more complex than using cuDNN alone. We also encountered discrepancies between the documentation and the actual implementation, which compounded the difficulty. The level of community activity is low; for instance, some queries on the Nvidia forums, particularly concerning transposed convolutions, have remained unanswered for over a year. While cuDNN is the optimal choice for achieving an optimized GPU implementation, especially where the memory footprint is a concern, this advantage requires a significantly greater development

effort, particularly for neural networks that include layers not supported by the library.

E. CPU Implementation with TVM

1) *Workflow Overview*: Figure 6 presents the workflow used to deploy our model on an ARM processor. We began by training a Float32 model with Keras, then utilized TVM 0.8 to export the model to Relay, TVM’s intermediate graph representation. At this stage, quantization of the neural network is optional, which we discuss further in section IV-E2. We compiled the model using an optimization level of 3, which in our experiments achieved the best trade-off between optimization and neural network accuracy. Subsequently, we employed TVM’s auto-scheduling, conducting 10,000 trials to optimize the scheduling of the inference on the CPU. The model was then ready for deployment on the ARM-A53 target, operating under a Linux-based system with the TVM runtime installed.

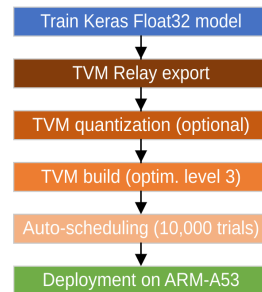


Fig. 6: CPU workflow from Keras/TensorFlow training to ARM CPU inference with TVM

2) *Quantitative Evaluation*: Table V shows the evaluation metrics obtained on a subset of the validation set, consisting of 1500 images, used in the TVM workflow, as the full validation set execution time was prohibitively slow on board. To ensure consistency, we maintained the same sub-validation set from the Keras evaluation through to the onboard evaluation. The table initially reports the metrics following Float32 training with Keras and TensorFlow. Subsequent rows display the metrics obtained when deploying the neural network on an Ultra96 board. The TVM workflow, without quantization, preserved the quality of the neural network’s output. In further experiments, we quantized every weight and activation function to eight bits, except for the first convolutional layer. We found that post-training quantization with TVM had a negligible impact on the evaluation metrics, minimally affecting both IoU and accuracy.

TABLE V: Evaluation metrics along the TVM workflow

Model	IoU	Accuracy
Float32 Keras	0.7170	0.9546
Float32 TVM	0.7170	0.9546
Int8 TVM	0.7007	0.9518

Table VI summarizes the performance metrics measured on the Ultra96 and ZCU104 boards. The ZCU104 demonstrated

approximately ten percent faster execution than the Ultra96, attributable to its faster DDR memory. However, latency on both boards was significant, limiting real-time inference of semantic segmentation neural networks on these CPUs. Power consumption averaged 1.1W at thermal equilibrium, which is relatively low and was consistent across both boards and quantization levels, as the ARM cores were fully utilized under all conditions. Quantization increased the execution time threefold, possibly due to a bug in the version of TVM used, suggesting that the auto-scheduling functionality may not be fully compatible with the quantized version of our network. Energy efficiency was slightly better on the ZCU104, but the difference was minimal, except with the quantized version, which showed a significant increase. The memory footprint was reduced further with quantization.

TABLE VI: Implementation metrics on the Xilinx Zynq Ultrascale+ boards with TVM

Board	Quantization	Latency (ms)	Power (W)	Energy efficiency (J/image)	Memory (MB)
Ultra96	No	540.7	1.05	0.568	68.4
Ultra96	Yes	1687	1.05	1.77	39.5
ZCU104	No	489.2	1.11	0.543	78.7

3) *Qualitative Evaluation*: The TVM workflow for targeting ARM CPUs is well-established, yet it is not without limitations, particularly due to a quantization bug encountered during our evaluations. This issue can be circumvented by utilizing the quantization functionalities of Keras/TensorFlow. The workflow benefits from being user-friendly, supported by extensive documentation and numerous examples. The versatility of the TVM stack allows for deployment on any ARM CPU that operates a Linux-based system, including smartphones and Raspberry Pi devices. Switching the target CPU requires altering only a single line of Python code. The community behind TVM is highly active, annually hosting TVMCon, a conference that fosters collaboration between academia and industry on neural network compilation. TVM’s fully automated build and auto-scheduling processes facilitate the deployment and optimization of state-of-the-art convolutional neural networks, rendering the TVM workflow exceptionally adaptable.

F. FPGA Implementation with FINN

1) *Workflow Overview*: Figure 7 presents the workflow utilized to deploy our model on an FPGA using the FINN library. As FINN is incompatible with Keras or TensorFlow, we re-implemented the U-Net model using PyTorch 1.7.1 and Brevitas 0.6.1. Brevitas is a quantization library designed to facilitate Quantization Aware Training (QAT) with PyTorch and to support deployment through FINN [25]. Initially, we trained a Float32 version of the U-Net using Keras and exported the weights to the PyTorch/Brevitas version of the U-Net. We then proceeded with training a quantized version of the U-Net using QAT in PyTorch/Brevitas, starting from the Float32 weights to significantly reduce QAT duration. Brevitas

supports mixed-precision quantization, enabling layer-wise bit-width parametrization for both weights and activation functions. After training, the model was exported to the ONNX format, which is compatible with FINN. At this stage, the model is transformed into a graph that contains only FINN HLS-compatible nodes. Subsequently, we defined the folding configuration for each graph node to set the parallelism, aiming to match the target latency without exceeding the FPGA’s available resources. If the folding configuration exceeded the FPGA resources, it required returning to the bit-width parametrization step and reiterating the QAT phase or adjusting the target latency. FINN’s built-in functions facilitate the invocation of Vitis HLS to synthesize each node independently, integrate them, and then implement the combined solution as a Vivado 2022.1 project deployed on the FPGA. FINN also offers rapid prototyping capabilities using the Pynq library.

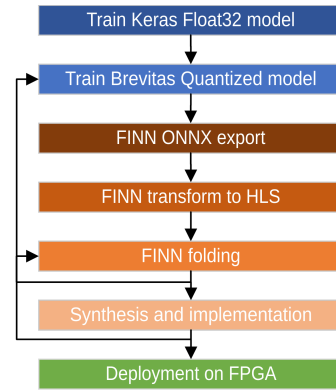


Fig. 7: FPGA workflow from PyTorch/Brevitas training to FPGA inference using FINN

2) *Quantitative Evaluation*: Table VII displays the evaluation metrics measured on the validation set throughout the FINN workflow. The initial row recalls the metrics after Float32 training with Keras. The final row presents the metrics for the quantized U-Net post-training, employing binary weights and 4-bit activation functions across all layers. Despite aggressive quantization, the accuracy and IoU only showed a slight decrease. Due to a suspected bug in the FINN library, we could not perform onboard inference to directly measure the evaluation metrics, a limitation we will discuss further in section IV-F3.

TABLE VII: Evaluation metrics along the FINN workflow

Model	IoU	Accuracy
Float32 Keras	0.7108	0.9531
Quantized Brevitas	0.6837	0.9488

While onboard inference execution was not possible, we derived certain results from the Vivado project, synthesis, and implementation reports. Table VIII summarizes these findings and estimations. The latency was derived from the synthesis reports, considering the highest latency across all graph nodes

(786,432 cycles) as the accelerator’s initiation interval. With a clock frequency of 100 MHz, we estimated the accelerator’s latency to be 7.86 milliseconds, corresponding to a throughput of 127 images per second. The on-chip power consumption, estimated at 5.5 Watts, was obtained from the FINN-generated Vivado project. The estimated energy efficiency is noteworthy, given the implementation of a low-bit quantized U-Net, although these results are provisional and should be approached with caution.

TABLE VIII: Estimation of the implementation metrics on the Xilinx ZCU104 with the FINN workflow

Board	Throughput (FPS)	Power (W)	Energy efficiency (J/image)	Memory (MB)
ZCU104	127.2	5.46	0.043	N/A

Table IX provides a summary of FPGA resource utilization based on the post-implementation report generated by Vivado, highlighting LUTs (Lookup Tables) as the primary limiting factor. The LUTs are predominantly utilized for the convolution computations, namely the im2col algorithm and the matrix-vector multiplication unit. Notably, the multi-threshold layers, representing the quantized activation functions, also consume a substantial number of LUTs, proportional to the square of the bit-width of the activation functions. We chose binary weights and 4-bit activations as an optimal balance between accuracy and estimated throughput. This approach also eliminated the need for DSPs, reducing the resource demands significantly. Our experience has shown that the FPGA resource estimations provided by FINN’s Python script were found to be unreliable.

TABLE IX: FINN FPGA resource usage on ZCU104 board

FPGA resource	Post-implementation utilization	FINN Python estimation	Available
LUT	205,249 (89%)	155,905	230,400
LUTRAM	43,498 (43%)	Not Available	101,760
Flip-Flop	235,448 (51%)	Not available	460,800
BRAM	96 (31%)	233	312
DSP	0 (0%)	0	1,728

3) *Qualitative Evaluation*: The Brevitas library for training quantized neural networks targeting FINN implementations is mature and user-friendly, closely mimicking the PyTorch experience, albeit lacking in examples. Conversely, the FINN library is still under development. We encountered and locally fixed several source code bugs during our experiments. While some of these issues have been addressed recently, indicating active development, the community remains relatively small compared to other libraries. The absence of certain HLS backend templates, such as transposed convolution, posed challenges. We circumvented this by substituting with a nearest neighbor upsampling layer followed by a convolution, which did not alter the U-Net’s parameter count or MAC operations.

Utilizing the FINN library can be challenging, particularly during the transformation phase, which requires users to

meticulously determine the appropriate transformations and their sequence. Often, modifications to the network architecture and quantization scheme are necessary to remove non-HLS compatible nodes. We had to develop two custom transformations not present in FINN to synthesize the U-Net effectively. A significant issue related to the U-Net’s shortcuts prevented us from implementing the neural network on the FPGA. This issue could stem from a problem with our custom transformations, a bug in FINN’s handling of concatenation layers, or FINN’s algorithm not allocating sufficiently large FIFOs to store the activation functions of the down-sampling path, thereby hampering the up-sampling path’s ability to perform its convolutions. Additionally, the documentation, spread across various websites and GitHub pages, is fragmented and challenging to navigate.

The FINN library holds significant potential for energy-constrained applications and is poised to mature into a highly energy-efficient method for executing neural network inference on FPGAs. As it develops, FINN’s approach, with its capacity for mixed-precision quantization and configurable folding, will enable tailored optimization for each layer’s bit-width, accuracy, resource usage, and latency.

G. FPGA Implementation with Xilinx Vitis-AI

1) *Workflow Overview*: Figure 8 outlines the workflow used to deploy a neural network on a Xilinx Zynq Ultrascale+ MPSoC using the Vitis-AI framework. This approach, distinct from rapid prototyping, is focused on actual embedded deployment. The process begins with training a Float32 model using Keras, followed by exporting it through the Vitis-AI toolkit version 2.0. Deployment on the MPSoC involves four primary activities:

- Configuring the DPU (Deep Learning Processor Unit) accelerator and generating the FPGA bitstream. This includes selecting the number of DPU cores and their size, which dictates the operations per clock cycle.
- Generating the application code in C++ to orchestrate model execution using the VART (Vitis AI Runtime).
- Compiling the model using 8-bit quantization with the Vitis-AI tools.
- Creating the Board Support Package (BSP) for the ZCU102 board.

Following these steps, the model is executed on the MPSoC, with the DPU on the FPGA handling most of the network operations. However, the CPU may process some layers, particularly when specific functions like the sigmoid activation at the end are not supported by the Vitis-AI quantization.

2) *Quantitative Evaluation*: Table X presents the evaluation metrics obtained from the validation set using the Vitis-AI workflow, with the initial line providing a baseline from Float32 training with Keras. Following the model’s quantization to 8-bit using Vitis-AI, no loss in accuracy was observed, thanks to the toolkit’s effective calibration function. The quantized model was subsequently deployed on the Xilinx DPU on the ZCU102 board, where no degradation in performance was noted, suggesting a possible regularization effect.

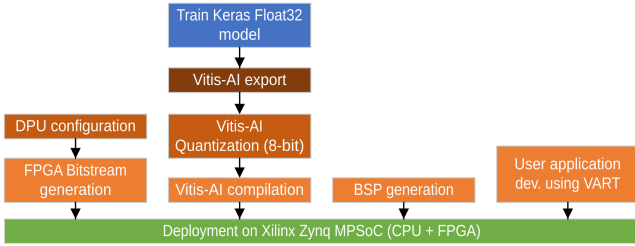


Fig. 8: FPGA workflow from Keras/TensorFlow training to FPGA/CPU inference using Vitis-AI

TABLE X: Evaluation metrics along the Vitis-AI workflow

Model	IoU	Accuracy
Float32 Keras	0.7108	0.9531
Int8 Vitis	0.7156	0.9542
Int8 DPU	0.7263	0.9583

Table XI summarizes the implementation metrics on the Xilinx ZCU102 board, measured on the validation set. The configuration uses three DPU cores, each capable of 4096 operations per clock cycle at 100 MHz. This setup was determined to be the best trade-off for embedded inference, balancing throughput and power consumption for optimal energy efficiency.

TABLE XI: Implementation metrics on the Xilinx ZCU102 with the Vitis-AI workflow

Board	Throughput (FPS)	Power (W)	Energy efficiency (J/image)	Peak memory (MB)
ZCU102	46.9	2.51	53.5	31

Table XII shows the FPGA resource utilization, with DSPs and BRAMs being the primary limiting factors due to their roles in MAC operations and storage of weights and intermediate feature maps, respectively. LUTs, LUTRAMs, and Flip-Flops still have available capacity, providing potential for future increases in the size or number of DPU cores.

TABLE XII: Vitis-AI FPGA resource usage with 3-core DPU on ZCU102 board

FPGA resource	Post-implementation utilization	Available
LUT	133,425 (49%)	274,080
LUTRAM	17,027 (12%)	144,000
Flip-Flop	297,576 (54%)	548,160
BRAM	771 (84%)	912
DSP	2,070 (82%)	2520

3) *Qualitative Evaluation*: The Vitis-AI workflow is robust, demonstrating significant maturity, particularly with toolchain updates in versions 2.0 and 2.5 that resolved previously encountered bugs. This versatile workflow supports a wide array of neural network layers, and users can incorporate custom IP blocks to introduce new operations. Xilinx provides comprehensive documentation and end-to-end examples through the Vitis-AI Model Zoo. The community surrounding Vitis-AI has grown rapidly, although the learning curve remains steep

due to the complexity of integrating various components such as BSP, Vivado, PetaLinux, and Vitis-AI tools. Additionally, while most components of Vitis AI are open source, some elements, such as the Vitis AI Compiler, remain proprietary, and certain tools within the Xilinx ecosystem require a commercial license.

V. SYNTHESIS

A. Synthesis and Workflow Comparison

Table XIII synthesizes the evaluation and implementation metrics results across the five workflows. As discussed in Section IV, onboard implementation was not achievable for the cuDNN and FINN workflows. Consequently, the implementation results from these workflows are estimates and should be interpreted with caution. Quantization is employed only when the hardware target does not support Float32 operations. The CPU and GPU workflows maintain the neural network’s output quality, thus achieving the same accuracy and Intersection over Union (IoU) as their respective baselines. The FINN workflow causes a slight degradation in accuracy and IoU, which is minimal considering the use of low-bit quantization. Conversely, the Vitis-AI workflow marginally improves the evaluation metrics on the validation set due to its quantization and calibration mechanisms, introducing a regularization effect. All workflows are compared at iso-accuracy levels. Nevertheless, there are significant differences in throughput and power consumption across the platforms and workflows. As expected, the CPU exhibits the lowest throughput, resulting in poor energy efficiency. The FPGA workflows, utilizing FINN or Vitis-AI, demonstrate superior energy efficiency. Both FINN and Vitis-AI enable the creation of customizable neural network accelerators, allowing for tailored FPGA resource usage, which in turn affects throughput and power consumption. Additionally, the use of quantization contributes to reduced power consumption. In contrast, GPU workflows and platforms have a considerably higher memory footprint compared to CPU and FPGA workflows and targets, presenting potential challenges in an embedded context.

Table XIV synthesizes the engineering metrics across the five workflows. Overall, TensorFlow and TVM stand out in the comparison. Both are open-source, mature, user-friendly, well-documented, and supported by large, active communities. Close behind, the Vitis-AI workflow exhibits similar positive attributes but is more challenging to use due to its incorporation of proprietary components and a requirement for hardware engineering expertise. Nevertheless, it offers greater customization capabilities than the TVM and TensorFlow workflows. The post-training 8-bit quantization in Vitis-AI, while adding complexity and development time, enhances energy efficiency. The cuDNN workflow is primarily designed for developers of deep neural network frameworks, such as PyTorch and TensorFlow, reflecting its maturity but also its limited suitability for embedded inference. Furthermore, the absence of certain operators, like transposed convolution and nearest neighbor upsampling, necessitates intricate and labor-intensive development. At the bottom of our comparison is

TABLE XIII: Synthesis of the evaluation and implementation metrics of the five workflows

Platform	Nvidia GPU		Xilinx Zynq UltraScale+ MPSoC		
Board	Jetson AGX Xavier		ZCU104		ZCU102
Workflow	TensorFlow	cuDNN	TVM (CPU)	FINN (FPGA)	Vitis-AI (FPGA)
Implementation	Yes	No	Yes	No	Yes
Numeric precision	Float32	Float32	Float32	W1A4	Int8
Accuracy change (vs. baseline)	0%	0%	0%	-0.43%	+0.52%
IoU change (vs. baseline)	0	0	0	-0.0271	+0.0155
Throughput (FPS)	74.6	54.3	2.04	127	46.9
Power (W)	14.6	5.61	1.11	5.46	2.51
Energy efficiency (mJ/image)	195	103	543	43.0	53.5
Memory (MB)	2200	1310	78.70	N/A	31

TABLE XIV: Synthesis of the engineering metrics of the five workflows. Metrics are quantified as high, medium, and low.

Platform	Nvidia Jetson AGX GPU		Xilinx Zynq UltraScale+ MPSoC		
Workflow	TensorFlow	cuDNN	TVM (CPU)	FINN (FPGA)	Vitis-AI (FPGA)
Maturity	High	High	High	Low	High
Ease of Use	High	Low	High	Low	Medium
Documentation and Examples	High	Medium	High	Medium	High
Community Support	High	Low	High	Low	High

the FINN workflow. Its current maturity level is low, with identified bugs, and it presents significant usability challenges. The need to develop custom transformations not available in the FINN library further complicates its usage. Although the community is active, it is relatively small compared to the others. Documentation and examples exist but are dispersed across various websites and GitHub repositories, which complicates the comprehension process. Additionally, FINN’s lack of support for certain operators, such as transposed convolution, necessitates alterations in the neural network architecture.

B. Limitations and Future Works

The conclusions presented in this paper reflect observations from 2021 to 2023. Nevertheless, the field of neural networks is rapidly evolving, and significant changes in these frameworks are anticipated in the near future. For instance, during the course of our project, we observed maturation in both the FINN and Vitis-AI workflows.

On the GPU front, our research focused on the high-level TensorFlow and the low-level cuDNN workflows. Nvidia’s TensorRT, an intermediate, open-source workflow for DNN inference, represents a potential area for future research [23]. Future investigations should also explore quantization to fully leverage the capabilities of Nvidia’s Tensor Cores in embedded GPUs, potentially narrowing the energy efficiency gap with FPGAs.

Further research should evaluate the use of more powerful CPUs, such as those based on Intel x86 architectures, with compilers like TVM or Intel nGraph that have shown effectiveness on these processors [14]. Due to time constraints, this study did not explore ASICs for neural network inference, such as Google’s Edge TPU or Intel’s Movidius VPUs [27], which appear to be promising for embedded applications and warrant future evaluation.

Lastly, the embedded domain poses unique challenges regarding robustness and explainability, aspects not covered in this paper. These topics are currently active research areas

in both academic [21], [34] and industrial spheres [6], [19], deserving attention in future studies.

VI. CONCLUSION

This paper has demonstrated the necessity of adapting advanced neural network architectures to novel datasets within an embedded framework. We introduced a lightweight U-Net that achieves the same accuracy with 16 times fewer parameters and Multiply-Accumulate (MAC) operations, validated on an aerial image segmentation dataset [17]. Furthermore, this study provided an extensive evaluation and comparison of various methods for real-time semantic segmentation of aerial images, employing three contemporary Commercial Off-The-Shelf (COTS) embedded computers across five distinct workflows.

The FPGA target, utilizing Vitis-AI, emerged as the superior choice due to its performance, energy efficiency, and system maturity. However, its implementation necessitates specialized hardware expertise. The ARM CPU target, leveraging TVM, is notable for its user-friendliness and maturity, yet its relatively low energy efficiency and throughput pose significant challenges for embedded system applications. The GPU target, utilizing TensorFlow, is acknowledged for its maturity and ease of use but is more appropriate for rapid prototyping than for actual embedded solutions. Conversely, the GPU target employing cuDNN is better aligned with embedded deployment but suffers from complexity and a lack of support for various neural network layers. Lastly, the FPGA target using FINN shows high potential for energy-constrained applications but necessitates additional development to become a practical option.

VII. ACKNOWLEDGEMENTS

This work was conducted within the SPOC project at the French Institute of Technology (IRT) Saint Exupéry. Funding was provided by the French Research Agency (ANR) and by the industrial partners of the IRT Scientific Cooperation Foundation (FCS).

REFERENCES

- [1] K. Abdelouahab, M. Pelcat, J. Serot, and F. Berry. Accelerating CNN inference on FPGAs: A Survey, May 2018. arXiv:1806.01683 [cs].
- [2] C. Adams, A. Spain, J. Parker, M. Hevert, J. Roach, and D. Cotten. Towards an Integrated GPU Accelerated SoC as a Flight Computer for Small Satellites. In *2019 IEEE Aerospace Conference*, pages 1–7, Mar. 2019. ISSN: 1095-323X.
- [3] M. Blott, T. B. Preußer, N. J. Fraser, G. Gambardella, K. O’Brien, Y. Umuroglu, M. Leeser, and K. Vissers. FINN- R: An End-to-End Deep-Learning Framework for Fast Exploration of Quantized Neural Networks. *ACM Transactions on Reconfigurable Technology and Systems*, 11(3):1–23, Dec. 2018.
- [4] L. Brown. Accelerate Machine Learning with the cuDNN Deep Neural Network Library, Sept. 2014.
- [5] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, H. Shen, M. Cowan, L. Wang, Y. Hu, L. Ceze, C. Guestrin, and A. Krishnamurthy. TVM: An Automated End-to-End Optimizing Compiler for Deep Learning. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 578–594, 2018.
- [6] Confiance AI. Un collectif français d’envergure inédite pour concevoir et industrialiser des systèmes à base d’intelligence artificielle de confiance, 2023. Accessed: 2023-05-01.
- [7] I. Dimitrovski, I. Kitanovski, D. Kocev, and N. Simidjievski. Current trends in deep learning for Earth Observation: An open-source benchmark arena for image classification. *ISPRS Journal of Photogrammetry and Remote Sensing*, 197:18–35, Mar. 2023.
- [8] X. Feng, Y. Jiang, X. Yang, M. Du, and X. Li. Computer vision algorithms and hardware implementations: A survey. *Integration*, 69:309–320, Nov. 2019.
- [9] K. Guo, S. Zeng, J. Yu, Y. Wang, and H. Yang. [DL] A Survey of FPGA-based Neural Network Inference Accelerators. *ACM Transactions on Reconfigurable Technology and Systems*, 12(1):2:1–2:26, Mar. 2019.
- [10] D. Karapetyan, S. Mitrovic Minic, K. T. Malladi, and A. P. Punnen. Satellite downlink scheduling problem: A case study. *Omega*, 53:115–123, June 2015.
- [11] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization, Jan. 2017. arXiv:1412.6980 [cs].
- [12] B. B. Kövari and E. Ebeid. MPDrone: FPGA-based Platform for Intelligent Real-time Autonomous Drone Operations. In *2021 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pages 71–76, Oct. 2021. ISSN: 2475-8426.
- [13] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, May 2015. Number: 7553 Publisher: Nature Publishing Group.
- [14] M. Li, Y. Liu, X. Liu, Q. Sun, X. You, H. Yang, Z. Luan, L. Gan, G. Yang, and D. Qian. The Deep Learning Compiler: A Comprehensive Survey. arXiv:2002.03794 [cs], Aug. 2020. arXiv: 2002.03794.
- [15] Y. Li, Z. Liu, K. Xu, H. Yu, and F. Ren. A GPU-Outperforming FPGA Accelerator Architecture for Binary Convolutional Neural Networks. *ACM Journal on Emerging Technologies in Computing Systems*, 14(2):18:1–18:16, 2018.
- [16] Z. Lv, T. Liu, J. A. Benediktsson, and N. Falco. Land Cover Change Detection Techniques: Very-high-resolution optical images: A review. *IEEE Geoscience and Remote Sensing Magazine*, 10(1):44–63, Mar. 2022. Conference Name: IEEE Geoscience and Remote Sensing Magazine.
- [17] E. Maggiori, Y. Tarabalka, G. Charpiat, and P. Alliez. Can semantic labeling methods generalize to any city? the inria aerial image labeling benchmark. In *2017 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, pages 3226–3229, Fort Worth, TX, July 2017. IEEE.
- [18] S. Mittal. A Survey on optimized implementation of deep learning models on the NVIDIA Jetson platform. *Journal of Systems Architecture*, 97:428–442, Aug. 2019.
- [19] A. Mojsilovic. Introducing AI Explainability 360, Aug. 2019.
- [20] K. Muhammad, S. Khan, M. Elhoseny, S. Hassan Ahmed, and S. Wook Baik. Efficient Fire Detection for Uncertain Surveillance Environment. *IEEE Transactions on Industrial Informatics*, 15(5):3113–3122, May 2019. Conference Name: IEEE Transactions on Industrial Informatics.
- [21] H. Nori, S. Jenkins, P. Koch, and R. Caruana. InterpretML: A Unified Framework for Machine Learning Interpretability, Sept. 2019. arXiv:1909.09223 [cs, stat].
- [22] E. Nurvitadhi, G. Venkatesh, J. Sim, D. Marr, R. Huang, J. Ong Gee Hock, Y. T. Liew, K. Srivatsan, D. Moss, S. Subhaschandra, and G. Boudoukh. Can FPGAs Beat GPUs in Accelerating Next-Generation Deep Neural Networks? In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA ’17, pages 5–14, New York, NY, USA, Feb. 2017. Association for Computing Machinery.
- [23] NVIDIA Corporation. Nvidia tensorrt. <https://github.com/NVIDIA/TensorRT>, 2023.
- [24] A. P érez, A. Rodríguez, A. Otero, D. G. Arjona, A. Jiménez-Peralo, M. A. Verdugo, and E. De La Torre. Run-Time Reconfigurable MPSoC-Based On-Board Processor for Vision-Based Space Navigation. *IEEE Access*, 8:59891–59905, 2020. Conference Name: IEEE Access.
- [25] A. Pappalardo. Xilinx/brevitas, June 2021. original-date: 2018-07-10T22:37:01Z.
- [26] B. Peccerillo, M. Mannino, A. Mondelli, and S. Bartolini. A survey on hardware accelerators: Taxonomy, trends, challenges, and perspectives. *Journal of Systems Architecture*, 129:102561, Aug. 2022.
- [27] A. Reuther, P. Michaleas, M. Jones, V. Gadepally, S. Samsi, and J. Kepner. Survey of Machine Learning Accelerators. *2020 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–12, Sept. 2020. arXiv: 2009.00993.
- [28] O. Ronneberger, P. Fischer, and T. Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation, May 2015. arXiv:1505.04597 [cs].
- [29] S. Sharma, C. Beierle, and S. D’Amico. Pose estimation for non-cooperative spacecraft rendezvous using convolutional neural networks. In *2018 IEEE Aerospace Conference*, pages 1–12, Mar. 2018.
- [30] N. Tijtgat, W. Van Ranst, B. Volckaert, T. Goedeme, and F. De Turck. Embedded Real-Time Object Detection for a UAV Warning System. In *2017 IEEE International Conference on Computer Vision Workshops (ICCVW)*, pages 2110–2118, Venice, Oct. 2017. IEEE.
- [31] F. Wang, X. Luo, Q. Wang, and L. Li. Aerial-BiSeNet: A real-time semantic segmentation network for high resolution aerial imagery. *Chinese Journal of Aeronautics*, 34(9):47–59, Sept. 2021.
- [32] M. Wu, C. Zhang, J. Liu, L. Zhou, and X. Li. Towards Accurate High Resolution Satellite Image Semantic Segmentation. *IEEE Access*, 7:55609–55619, 2019. Conference Name: IEEE Access.
- [33] Y. Xing, J. Weng, Y. Wang, L. Sui, Y. Shan, and Y. Wang. An In-depth Comparison of Compilers for Deep Neural Networks on Hardware. In *2019 IEEE International Conference on Embedded Software and Systems (ICCESS)*, pages 1–8, June 2019.
- [34] F. Xu, H. Uszkoreit, Y. Du, W. Fan, D. Zhao, and J. Zhu. Explainable AI: A Brief Survey on History, Research Areas, Approaches and Challenges. In J. Tang, M.-Y. Kan, D. Zhao, S. Li, and H. Zan, editors, *Natural Language Processing and Chinese Computing*, Lecture Notes in Computer Science, pages 563–574, Cham, 2019. Springer International Publishing.
- [35] R. Zhao, S. Liu, H.-C. Ng, E. Wang, J. J. Davis, X. Niu, X. Wang, H. Shi, G. A. Constantinides, P. Y. K. Cheung, and W. Luk. Hardware Compilation of Deep Neural Networks: An Overview. In *2018 IEEE 29th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pages 1–8, July 2018. ISSN: 2160-052X.

Exploring Neural Network Architectures for Satellite Imagery on FPGA devices

Jean-Baptiste Chaudron and Jacques Gatard

EMBRYA, Toulouse, France

jean-baptiste.chaudron@embrya.io

jacques.gatard@embrya.io

Abstract—Today, Artificial Intelligence (AI) solutions are deployed for various applications in several technological domains. Deep Learning (DL) methods, especially, Artificial Neural Networks (ANN) are considered for space systems to provide new perspectives for complex earth observation or space exploration missions that request in-orbit data processing. However, the inherent complexity of such algorithms in terms of arithmetic operations and associated memory usage limits their integration on on-board components and, usually, requires special accelerator entities dedicated to perform such tasks. For space systems, due to limitations on energy availability, Field Programmable Gate-Array (FPGA) devices are usually preferred over more power-consuming Graphical Processing Units (GPU). Nonetheless, the design and implementation processes are more complex for FPGA and must be carefully analyzed. In this paper, we describe our approach from initial prototyping to implementation for an industrial test-case about satellite imagery: the Airbus Ship Detection Challenge (ASDC). We discuss the applications considerations for classification and semantic segmentation and describe a set of selected ANN architectures together with the training environment. We conduct an evaluation strategy to select small and efficient architectures that provide good trade-off in terms of accuracy and performance. Finally, we detail optimization techniques and experiment on-board performances of our EMBRYA’s Enki core-ip on a selection of FPGA based embedded devices.

Index Terms—FPGA, Classification, Semantic Segmentation, CNN

I. INTRODUCTION AND BACKGROUND

Artificial Neural Networks (ANN) algorithms have a very long history which started in 1943 with the invention of the *perceptron* concept [40] followed by its first implementation in 1957 [54] and extension to multi-layers structures in 1958 [55]. These first approaches had limited learning capabilities and several initiatives have been proposed over the years to fill this gap [30] [4] [35]. In 1982, the *back-propagation* method was formulated [68] and then experimented [56] providing the baselines of the key feature of ANN: their ability to learn. Based on this foundation, an immense amount of research has been conducted to refine and expand the ANN concepts. Thus, a large variety of architectures have been proposed in the literature which can be classified in three main types:

- The Multi-Layer Perceptrons (MLP), or Fully Connected (FC) networks, regroups the extensions to multiple layers of the original perceptron structure [29].

- The Recurrent Neural Networks (RNN) are extensions of MLP which integrate a feedback loop [31] or an internal memory [11]. These adds-on address a limitation inherent to the MLP structure, which is its reliance solely on inputs for predicting the output. Thus, it enables the capture of dependencies between the previously processed output and the subsequent output. These days, Long-Short Term Memory (LSTM) [17] and Gated Recurrent Unit (GRU) [7] are the most used architectures of this kind.
- The Convolutional Neural Networks (CNN) are specialized for computer vision applications. Their architectures, derived from the *neocognitron* concept invented in 1980 [14], are capable of processing images by regions to capture geometric 2-Dimensional (2D) relationships. The first CNN, based on matrix convolution operations, was implemented in 1998 [33] for the MNIST¹ handwritten digit recognition task. The differences between the three ANN types mentioned above are illustrated in Figure 1.

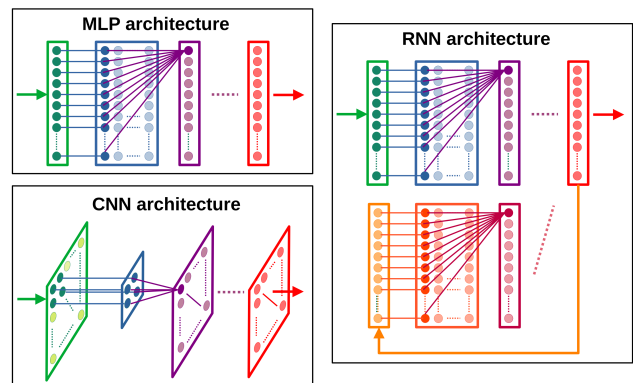


Fig. 1: MLP, CNN of RNN neural network types

Over the years, the expansion of ANN has marked a significant milestone in various fields, consistently outperforming more traditional methods such as image and speech recognition. For space applications as well, ANN provide substantial opportunities to improve the functionality and efficiency of space systems across a wide range of uses and applications. [58]. More specifically, ANN can contribute to

¹<http://yann.lecun.com/exdb/mnist/>

autonomous navigation and control of spacecraft which is crucial for space missions beyond direct human control due to distance or complexity. By processing vast amounts of data from various sensors, neural networks can help in real-time decisions for path correction, navigation and obstacle avoidance [61]. Also, ANN can help to improve the efficiency and reliability of space communication systems [13], for example, with noise reduction [52] or interference mitigation [8]. This is particularly important for deep space missions where communication delays and signal degradation are significant challenges. Finally, in Earth observation missions, ANN are used for processing and analyzing data collected from satellites [59] to enhance the ability to monitor climate change, natural disasters and urban development. By processing data directly on the satellite, only relevant information needs to be transmitted to Earth such as cloud detection for image quality [19]. This significantly reduces the bandwidth requirements and data transmission costs, a critical consideration given the high volume of data generated by Earth observation satellites. The integration of such resource-intensive computer vision algorithms into on-board embedded systems requires careful consideration and analysis, especially considering the limited memory and processing power of these systems.

This paper introduces a first step forward for the development of our EMBRYA's Enki core-ip prototype, deployed on Field Programmable Gate-Array (FPGA) devices, by applying it to a comprehensive study on ASDC satellite imagery dataset covering aspects from initial conceptualization to on-board implementation. We will detail the dataset and explore the associated tasks of classification and semantic segmentation associated with it. Our work investigates various CNN architectures, examining their original designs and exploring their reductions to enhance on-board performance. Specific optimization details and experimental results are also presented. Furthermore, a thorough bibliography is provided throughout the document to substantiate our methodology. The structure of this paper is organized as follows:

- Section II explains the background with the emergence of hardware accelerators for AI inference and the motivation behind our new generation Enki technology.
- Section III describes the application in focus. We will present the ASDC dataset, detailing its characteristics and features. Following this, we will delve into the binary classification and semantic segmentation tasks that will be applied to this dataset.
- Section IV provides a description of the CNN architectures under investigations and associated training environments and presents results for our complete exploration strategy. We select different architectures that we analyze and experiment for effective on-board implementations.
- Section V explains the optimization strategy and shows experimental results of ENKI on a set of selected SoC FPGA hardware devices and, finally, Section VI concludes and proposes some perspectives.

II. BACKGROUND

A. ASIC IA accelerators

Nowadays, due to the raise of AI based algorithms, several Application-Specific Integrated Circuits (ASIC) based accelerators are proposed to support these computing intensive tasks [51]. GPU based solutions are the mainstream approach to process ANN and other AI based algorithms in particular with the expansion of NVIDIA and the development of CUDA, an open source language similar to C++ used to directly program low level functions on NVIDIA GPUs. In addition, NVIDIA devices (and CUDA) supports many deep learning frameworks widely used nowadays [23]. For embedded systems, NVIDIA also offers several scaled-down GPU versions which are integrated with a CPU on a System-on-Chip (i.e. SoC with integrated GPUs or iGPUs). These small GPU devices, such as Jetson Nano, have shown that they can operate in a space environment [62]. However due to peak power consumption (associated to heat dissipation) the integration of such hardware is still problematic for satellite [18]. Therefore, for space applications, other ASIC based solutions such as Google Coral TPU (Tensor Processing Unit) or Intel Myriad Vision Processing Units (VPU) are preferred [15] offering good performances with low power consumption [49]. Recently, the Intel Myriad VPU was the first AI accelerator to be integrated on-board in a satellite [20].

B. FPGA Solutions

Despite their efficiency, ASIC based systems have several drawbacks. Especially, ASIC are purpose-built for specific capabilities and therefore cannot be reprogrammed. In addition, the design of a dedicated ASIC solution requires a long and costly development cycle which might not be suitable for some projects and applications. In this context, FPGA technology provides elegant alternative with a no hard etched-it circuitry. Thus, it can be reprogrammed and require low power consumption which makes it an excellent alternative to ASIC for development. The use of FPGA based devices has been analyzed for years for integration on space applications [22]. Therefore, the raise of AI algorithms has also pushed a lot of initiatives and to develop and deploy AI chip-ips on FGPA [67]. Currently, there are two main approaches to deploy AI/ML based applications on FPGA-based systems [47] :

- 1) The design of specialized custom chip-ips for executing specific neural networks [21]. Currently, the main approach is to use automatic HDL generators frameworks which ease the development of hardware chip-ip by translating machine learning algorithms and generating the design of dedicated chip-ip solutions for integration into FPGA such as FINN [65], HLS4ML [12], MATLAB HDL Coder [38]. Recently, FPG-AI framework has been characterized on NanoXplore FPGAs [34].
- 2) The use of stand-alone generic AI chip-ip accelerators such as Microchip CoreVector Blox Neural Network Engine [41], Intel FPGA AI Suite IP [3] or Xilinx Deep Processing Unit (DPU) [73]. The performance and energy

efficiency of the Xilinx DPU accelerator is among the most competitive nowadays [2], it has already been used for space applications [48] and analyzed for radiation-tolerance [1]. These generic chip-ips are usually designed per FPGA providers and dedicated to their edge devices which makes it difficult to deploy on custom hardware or new technologies. Also, these chip-ips usually target SoC technology combining the FPGA with a CPU. Thus, the chip-ip accelerator is used as a co-processor controlled by means of dedicated instructions sent per the CPU.

C. Why ENKI?

This paper presents the first deployment of our Enki prototype based on a technology under patent review. It offers a new concept for a generic full AI ASIC processor (and not only a co-processor). Currently, for development purpose, Enki is being deployed on FPGA devices and therefore can be considered as one of these generic chip-ip AI accelerators described in Section II-B. From the conceptual point of view, Enki addresses some restrictions compared to current stand-alone generic AI chip-ip solutions:

- *Genericity*: Current solutions are generally oriented toward pure performance of CNN within very strict genericity bounds (type of layers, activation, ...) and are not usable outside these bounds. For example, CNN can't be directly combined with RNN on the same generic chip-ip. Also, it is not possible to integrate some other feature extraction techniques, such as the Hough transform [9] or Non-Maximum Suppression (NMS) useful to complement CNN based image processing solutions. Also, on Enki, you can load multiples neural network configurations.
- *Precision*: Due to the limitation of FPGA logic and memory, chip-ip accelerators use integer datatypes such as Xilinx DPU using `int8` representation. The migration process from floating point model must be handled using dedicated tool to ensure a proper conversion with a limited loss in precision. Following the recent standardization extension of Open Neural Network Exchange (ONNX) to quantized version QONNX [44], Enki use the scaled integer quantization combining floating point datatypes for scales (costly in term of resource nonetheless) with integer datatypes (that can be bit-wise fine tuned). This principle combined with direct quantized training ensures higher accuracy of the quantized models. The migration details are described in Section V-A and Enki is, to the best of our knowledge, the first QONNX compliant generic FPGA chip-ip accelerator.
- *Deployment*: Enki HDL code can be synthesized to any technology-dependent netlist and then implemented (optimization, placement and routing) on a dedicated FPGA. In addition, the concepts behind Enki are not restricted to SoC architecture and can be deployed on a non intrusive FPGA only solution (such as an FPGA connected an Ethernet network).

- *Learning*: Finally, all AI chip-ip accelerators are optimized for inference following the *offline training, online learning* paradigm which is very well suited for supervised learning models. However, nowadays, some more advanced AI concepts are emerging and offering new perspectives such as Federated Learning (FL) for satellites constellations [39] or Reinforcement Learning (RL) applicable in the context of spacecraft control [64]. These new algorithms requires the *learning* capability non existing nowadays on state of the art accelerators. A comparison overview is summarized in Table I.

TABLE I: Comparison of Enki (Overview)

	Enki	Xilinx DPU	HLS4ML/FINN
Inference	✓	✓	✓
External memory	✓	✓	X
Genericity (Chip-ip)	✓✓	✓	X
Performance	✓	✓✓	✓✓
Online Training	✓✓	X	X

III. APPLICATION DESCRIPTION

A. The ASDC dataset

The rise in maritime incidents such as illegal fishing, cargo theft, AIS spoofing, drug trafficking and shipwrecks, has made it crucial to detect ships to oversee and curb maritime crime effectively. Researchers are continually working to overcome the challenges of automatically detecting ships, especially in remote sensing. This effort has significantly contributed to ensuring safety, identifying illegal activities, monitoring pollution, and tracking oil spills. In the field, ASDC is a well-known satellite imagery data-set [28] hosted on Kaggle², a popular website for data science competitions. The ASDC serves as a tangible benchmark for the development and comparison of various computer vision and machine learning algorithms. Its large dataset of 31.4 GB is composed of a set of 208162 RGB images (3 channels) with a resolution of 768×768 pixels, 192556 images that can be used for training (labelled) and 15606 images that are used for testing (non labelled). The final goal of the competition was to accurately identify and locate ships within the images of the test set. It presents unique challenges, including the detection of small or partially hidden ships, distinguishing ships from other objects or natural formations, and adapting to diverse lighting and weather conditions. Satellite imagery in the dataset can feature either a single ship, multiple ships, or no ships at all, as illustrated in Figure 2. The breakdown of how ships are distributed across the images in the training dataset is detailed in Table II and shows a very unbalanced distribution which might be considered for properly training the selected CNN models (see Section IV).

²<https://www.kaggle.com/competitions/airbus-ship-detection>

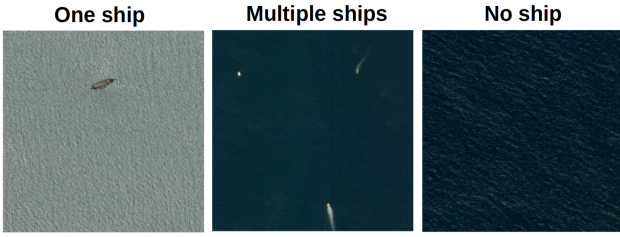


Fig. 2: ASDC images with and without ship

TABLE II: Ships distribution across the training dataset

Ship(s)	None	1	2	3	4 (and more)
Number of images	149999	27104	7674	2954	4824
Percentage	77,9%	14,1%	4,0%	1,5%	2,5%

B. Classification and Segmentation

The ASDC challenge can be tackled using two synergistic methods: classification and semantic segmentation. Focusing initially on classification, we opted for a binary classification approach to determine whether the input image contains a ship (one or many) or none as depicted in Figure 3.

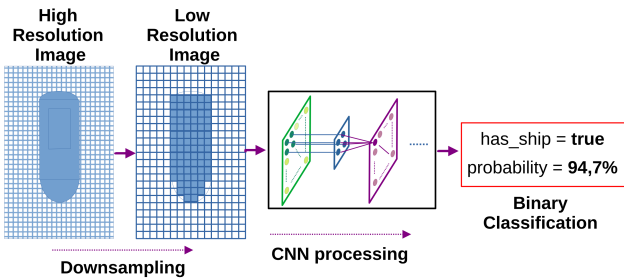


Fig. 3: Binary classification for ASDC

Since the inception of the first CNN initiative [33], the CNN architectures have expanded to address more intricate image classification challenges, such as those presented by the CIFAR³ or ILSVRC⁴ [57] datasets. Consequently, sophisticated CNN architectures like AlexNet [32] or ResNet [27] have been developed. These architectures are highly effective in solving image classification problems, which involve determining whether an image belongs to a given specific class. In the top-performing solutions on the ASDC challenge leaderboard on Kaggle⁵, common models used to address the classification problem are standard variants of ResNet [27] as well as more advanced variants such as ResNeXt [69]. Considering the semantic segmentation aspect of ASDC, traditional

³<https://www.cs.toronto.edu/~kriz/cifar.html>

⁴<https://www.image-net.org/challenges/LSVRC/>

⁵<https://www.kaggle.com/competitions/airbus-ship-detection/leaderboard>

CNN architectures for classification cannot be directly applied. As depicted in Figure 4 for ASDC, semantic segmentation enhances classification to operate at a pixel level. In this approach, the model is required to determine the class for each pixel in the input image. The model's output is a probability distribution across various classes for every pixel of the input image. It is important to note that for both classification and segmentation tasks, it is often necessary to down-sample the original image to manage memory usage and reduce the number of arithmetic operations required.

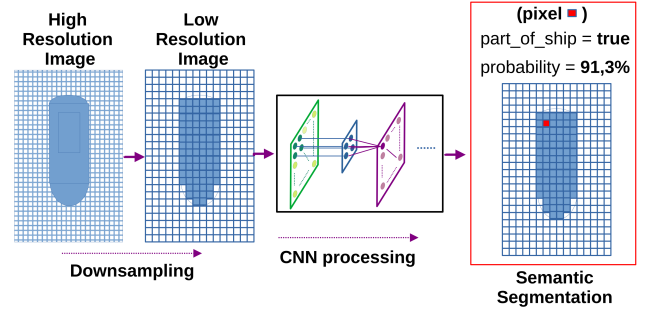


Fig. 4: Semantic Segmentation for ASDC

Many of the leading solutions for semantic segmentation in the ASDC challenge, as seen on the Kaggle leaderboard, utilize primarily the UNet architecture [53]. For identifying ships, some solutions opt for instance segmentation, often employing Mask-RCNN [24] or You Only Look Once (YOLO) [46], instead of semantic segmentation.

C. Our application design

For efficiency purpose, in our application design we used a combination of detection and semantic segmentation to avoid the unnecessary runs of a deep semantic segmentation model directly on the input image if it isn't containing a ship. Therefore, our application first use a detection algorithm on down-sampled 64×64 image issued from the 768×768 original input image. Then, if a ship is detected, the application runs the segmentation model to create the expected segmentation matrix to locate the ships on a bigger image 224×224 (still downsampled from the original input one). This efficient implementation concept is illustrated in Figure 5.

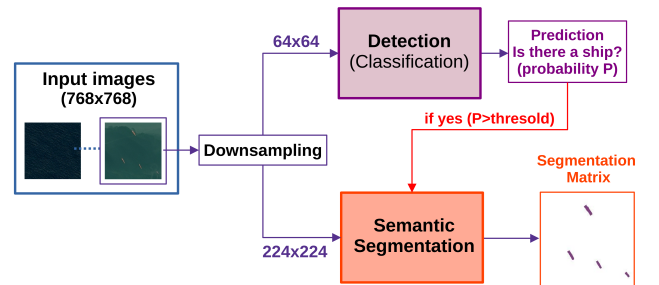


Fig. 5: Optimizing Segmentation with preliminary detection

IV. EXPLORING CNNs ARCHITECTURES

A. Architectures under investigation

For the classification task, we have selected 2 types of CNN for evaluation: VGG [36] based on a traditional architecture and the more sophisticated ResNet [27] containing parallel residual connections in its structure. For the semantic segmentation, we have selected 2 types of *Encoder-Decoder* CNN architectures: Unet [53] and Segnet [5]. These have very similar architectures, Unet shares feature maps between the encoder and the decoder while Segnet shares max-pooling indexes⁶ [6]. As mentioned in III-C, because of the resource constrained systems of space systems, our goal is to find the smallest architectures that perform well and offer a good balance between accuracy and performance. Therefore, we explore the internal designs of these architectures by reducing the internal number of channels in order to evaluate the balance between accuracy and performance (in terms of memory usage). We ended up with 4 architectures from big model size (>50 MB) to low size (<1 MB) more suitable for FPGA deployment. In addition, for each of these architectures, we experimented 4 different activation functions: well known Relu, PRelu [25], Silu [10] and Gelu [50]. These CNN architectures and their sizes are resumed in Table VI.

TABLE III: Size (MB) for model architectures

Type	Arch. 1	Arch. 2	Arch. 3	Arch. 4
Resnet	50.3	12.6	4.6	0.15
Vgg	54.5	13.9	4.0	0.14
Unet	124.3	11.5	4.6	0.8
Segnet	117.9	13.1	5.2	0.7

B. Training Environment

Based on the original ASDC dataset, we built two test cases: one for the detection (i.e. classification) with down-sampled 64×64 images and one for the semantic segmentation with down-sampled 224×224 images. Considering that the detection algorithm would be first used to detect if the image contains a boat we considered the images of ASDC that are containing one (or many) boat(s) for semantic segmentation. We divided the official training set containing ship into new training and validation sets with a ratio of roughly 85% and 15% respectively. For classification, we kept the same ratio of 85% and 15% for training/testing sets and integrated images without boat to the training set and validation set as illustrated in Figure 6.

We trained our architectures using Pytorch framework [45]. For classification, we used weighted Adam optimizer [37]

⁶Note that Segnet is an example of CNN that can't be deployed as if on Xilinx DPU due to its max unpool layers

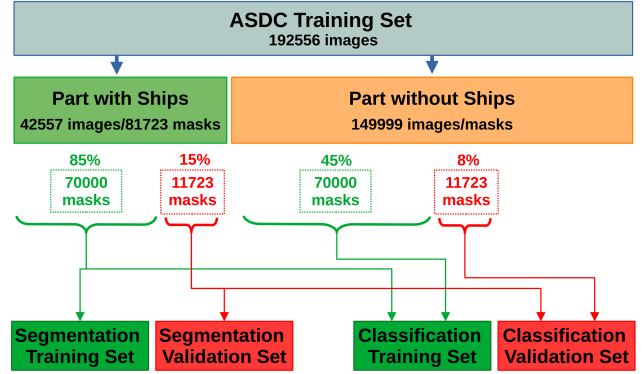


Fig. 6: ASDC handling for Classification and Segmentation

over 50 epochs with a learning rate of 0.005 and momentum values (betas) equals to 0.9 and 0.999 respectively and training batches with a size of 256 and a simple Binary Cross Entropy error. For segmentation, we used the same optimizer over 100 epochs with a learning rate of 0.001 (with the same 0.9 and 0.999 for momentum values) and training batches with a size of 16. We normalized the RGB format of the input image pixel from 0-255 range to 0-1 range and we initialized our weights of our architecture using Kaiming methods [26]. As the choice of loss function is crucial in deep learning tasks [63], especially in imbalanced datasets like the ASDC. We selected a combination of Dice and Focal Losses to provide a more balanced approach. The Dice Loss is suitable for data with class imbalance, as it measures the overlap between the predicted segmentation and the ground truth and makes it more sensitive to the pixels of the minority class (ships in this case). On the other hand, the Focal Loss is designed to address class imbalance by down-weighting the loss for well-classified examples. This allows the model to focus on hard, misclassified examples. The combination of these two loss functions is very efficient as shown in Section IV-D.

C. Classification Results

For classification, the accuracy representing the percentage of correct prediction compared to label is used to evaluate the correctness of the model. The results for the accuracy on the validation set are synthesized in Table IV.

As expected, the biggest models are performing better than the small ones. However, we can observe that the very small models still provide a very good accuracy and being extremely smalls (above 0.9 for Vgg and around 0.9 for Resnet). Overall, in general, the Vgg architecture is the one performing the best for all sizes. For these detection architectures, in the general case, the use of other Gelu and Silu activation functions doesn't bring significantly better results than Relu while adding more complexity for the on-board process of these models. The Prelu activation function can thus be considered for implementation providing good results, as this is depicted in Figure 7 for the biggest (Arch.1) and the smallest (Arch.4) Vgg architectures where we can observe that

TABLE IV: Classification accuracy (Val.) for 64×64 images

Type	Act	Arch. 1	Arch. 2	Arch. 3	Arch. 4
Resnet	1 (relu)	0.930	0.925	0.924	0.898
Vgg	1 (relu)	0.939	0.933	0.932	0.920
Resnet	2 (prelu)	0.934	0.931	0.929	0.898
Vgg	2 (prelu)	0.941	0.939	0.933	0.919
Resnet	3 (silu)	0.926	0.925	0.924	0.905
Vgg	3 (silu)	0.934	0.935	0.933	0.917
Resnet	4 (gelu)	0.928	0.927	0.927	0.901
Vgg	4 (gelu)	0.937	0.935	0.931	0.917

Prelu outperforms the other activation functions for the biggest architecture (conf11 and conf41) while the Relu is the best one for the smallest architecture (conf12 and conf42).

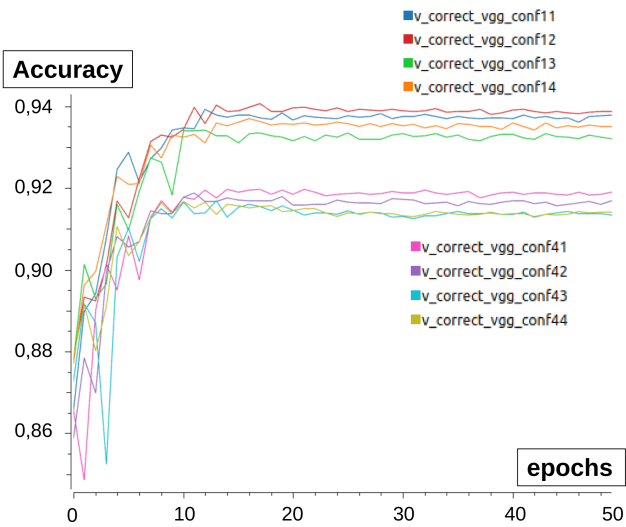


Fig. 7: Evolution of accuracy for Vgg architectures

D. Segmentation Results

To measure the accuracy of our models, we use Intersection over Union (noted IoU), one of the most used metrics for the evaluation of semantic segmentation models [42]. IoU considers the number of common pixels between the label and prediction masks (the intersection of the two sets) divided by the total number of pixels present in the two masks (the union of the two sets). The results, obtained on the validation set, are synthesized in Table V. We can observe that we are obtaining very good results with high IoU for all the architectures showing the efficiency of our combined Dice/Focal Loss function. As expected, similarly to classification architectures, the biggest models are better than the small ones, still offering

very good results. The most efficient CNN in our experiment is Unet which outperforms Segnet. Table V also shows that the use of other activation functions might be relevant to increase the accuracy of semantic segmentation models especially for small architectures. In general, Silu and Gelu are the most performing activation functions in our tests. To illustrate this, Figure 8 presents the evolution of the IoU for the validation set over the training epochs for Unet, the best performing CNN, taking its biggest architecture (Arch.1) and its smallest one (Arch.4). We can see that the validation IoU for Silu (conf13 and conf43) and Gelu (conf14 and conf44) are the ones that have the best evolution curves over Relu (conf11 and conf41) thus these activation functions might be considered to enhance the ANN accuracy.

TABLE V: Segmentation IoU metrics for 224×224 images

Type	Act	Arch. 1	Arch. 2	Arch. 3	Arch. 4
Unet	1 (relu)	0.874	0.852	0.837	0.774
Segnet	1 (relu)	0.779	0.762	0.754	0.710
Unet	2 (prelu)	0.884	0.842	0.840	0.772
Segnet	2 (prelu)	0.775	0.756	0.753	0.711
Unet	3 (silu)	0.891	0.871	0.850	0.792
Segnet	3 (silu)	0.784	0.763	0.753	0.735
Unet	4 (gelu)	0.894	0.875	0.843	0.783
Segnet	4 (gelu)	0.783	0.770	0.753	0.715

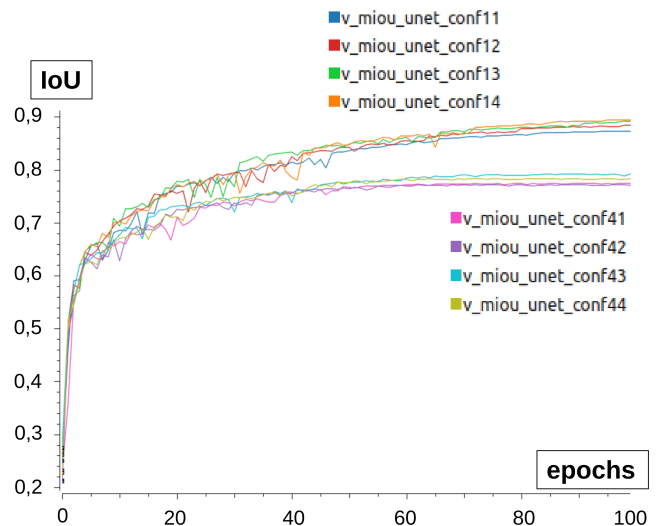


Fig. 8: Evolution of IoU for Unet architectures

V. ON-BOARD IMPLEMENTATIONS

A. Discussion and optimization strategy

The high constraints on FPGA devices to deploy a neural network usually induces a lot of step to migrate, to highly optimize (pruning, quantization, ...), to generate FPGA bit-stream and deploy the system on-board. Unlike the other approaches, we want to deploy target networks with a simple optimization strategy. Moreover, the results from Section IV-C and IV-D show that, by using the Pytorch framework with proper training settings (loss function, ...), we can design and train small and accurate models. We have selected Vgg 41 (Relu) and 42 (Prelu) for classification and Unet 41 (Prelu) and 43 (Silu) for segmentation on which we applied two optimization steps:

- 1) *Remove Batch Normalization (BN)*: BN layers are mainly introduced to improve training, by increasing convergence speed and avoiding local minima. In order to improve the execution time while preserving semantic performance batch normalization parameters are folded into the previous convolutional or dense layers weights and biases⁷.
- 2) *Migrate to QONNX*: In order to alleviate storage requirements and limit floating point operations, quantization is applied. On our side, we use QONNX representation based on scaled integer quantization which works by shifting and scaling numbers to fit within an integer range. To do so, we used Brevitas library [43] compliant with Pytorch and directly integrated in our environment. Brevitas supports both Post-Training Quantization (PTQ) and Quantization-Aware Training (QAT) techniques. We applied a merge of these two techniques by retrain the quantized Brevitas *twins* after the batch normalization fusion. This simple optimization process is illustrated in Figure 9 and Figure 10.

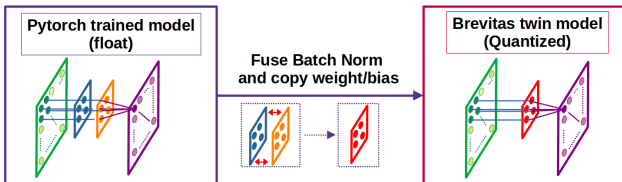


Fig. 9: Batch Normalization fusion and weight copy (Step 1)

B. Hardware platforms

To deploy our scaled integer quantized models, we have chosen 3 Xilinx SoC architectures:

- A small device: Zedboard (XC7Z020 SoC) [70]
- A medium device: Kria KR260 (XCK26 MPSoC) [72]
- A large device: ZCU 102 (XCZU9EG MPSoC) [71]

Table VII summarizes the data issued from the three aforementioned documents regarding the internal characteristics of the devices. It is important to note that Block RAM is a dedicated type of RAM that does not require extra Look-Up

⁷<https://nenadmarkus.com/p/fusing-batchnorm-and-conv/>

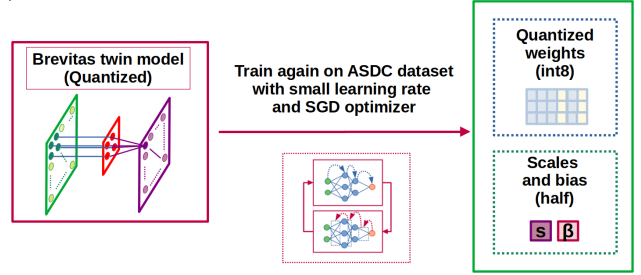


Fig. 10: Quantization-Aware (re)training (Step 2)

TABLE VI: Accuracy and IoU results for quantized models

Type	Original	Quantized
Vgg 41 (<i>relu</i>)	0.920	0.916
42 (<i>prelu</i>)	0.919	0.915
Unet 41 (<i>relu</i>)	0.774	0.758
Unet 43 (<i>silu</i>)	0.792	0.776
Vgg size (MB)	0.14	0.04
Unet size (MB)	0.8	0.22

Tables (LUTs), in contrast to distributed RAM, which is using LUTs.

TABLE VII: Xilinx SoC FPGA under consideration

ID	Zedboard	Kria KR260	ZCU 102
System Logic Cells	85000	256200	599550
CLB Flip-Flops (FF)	106400	234240	548160
CLB LUTs	53200	117120	274080
Distributed RAM (Mb)	-	3.5	8.8
Block RAM (36Kb)	140	144	912
Block RAM (Mb)	4.9	5.1	32.1
UltraRAM Blocks	0	64	0
DSP Slices	220	1248	2520
PC-PL DDR	512MB (3)	4GB (4)	4GB (4)

C. Enki results

We synthesized and implemented our Enki prototype for each of those SoC FPGA targets. We have been able to integrate the full version of Enki on the ZCU 102 and Kria 260 while we had to set up a reduced version, using less BRAM memory for feature map intermediary data and with less features, for the Zedboard. The full chip-ip of this first version includes features to perform Convolution, Transposed Convolution, and Linear, Max and Average Pooling layers. Also, it supports full float Sigmoid and Hyperbolic Tangent

(Tanh) as well as Relu, Silu and Prelu. The full version also has the capability to load 4 configurations for 4 Networks. Thus, Vgg 11, Vgg 12, Unet 11 and Unet 13 are loaded within Enki and can be executed. Note that these configurations can be easily changed without modifying the bitstream. The reduced version is restricted by only supporting a single NN configuration for execution and removing the support to full float activation functions. The resource utilization of the Enki chip-ip for each device is summarized in Table VIII and the on-board latency results are shown in Table IX. Execution times are expressed in milliseconds (ms) and represent the duration of one execution step to process one input image.

TABLE VIII: Enki FPGA ressource usage

ID	Zedboard <i>reduced</i>	Kria KR260 <i>full</i>	ZCU 102 <i>full</i>
CLB Flip-Flops (FF)	31654 30%	37813 16%	43991 8%
CLB LUTs	32921 62%	42434 26%	41637 15%
Block RAM (36Kb) (36Kb)	48 34%	36 25%	168 18%
UltraRAM Blocks	NA	52 81%	NA
DSP Slices	182 83%	234 19%	237 9%

TABLE IX: Enki latencies

ID	Zedboard <i>50 Mhz</i>	Kria KR260 <i>125 Mhz</i>	ZCU 102 <i>250 Mhz</i>
Vgg 41 (<i>relu</i>)	246 ms	88 ms	43 ms
Vgg 42 (<i>prelu</i>)	247 ms	88 ms	43 ms
Unet 41 (<i>relu</i>)	NA	911 ms	473 ms
Unet 43 (<i>silu</i>)	NA	918 ms	477 ms

D. Discussion

The first above-mentioned development results of our technology highlighted the successful deployment capacity of Enki and several points may be highlighted:

- The latency results are lower than we expected and to what can be achieved on the Xilinx DPU⁸. This first version of the full chip-ip Enki is not optimized yet. On the Kria and the ZCU 102, the footprint is relatively small which open the path for plenty of adjustments and optimizations. Moreover, we selected some clock settings that are below the maximum achievable on these card.
- Only latencies have been measured, since reconfiguration is part of Enki technology. Each model already configured

⁸Additional optimization steps using Xilinx software library might be required to simple optimization steps described in this document

on Enki can be executed instantaneously via a simple command. The reconfiguration with new external models can be summarized to the cost of a copy into the correct DDR area (so few microseconds) and therefore negligible compared to inference latency execution time.

- The results from the reduced IP needs to be further analyzed and compared to the full IP on the same board.
- It seems that there are some improvements with the use of URAM memories on the Kria. The latency is a little less than twice the latency of the ZCU 102. Again, it must be confirmed per execution with and without URAM usage on the same board (using the reduced IP for example).
- The Prelu execution compared to Relu is negligible however, the use of the Silu has an impact on the execution time using the full float (taylor approximated) sigmoid activation function. We are currently replacing these per interpolation tables to speed up the process.

VI. CONCLUSION AND PERSPECTIVES

In this paper, we have successfully validated our bottom up approach for an ship detection and segmentation application. We have investigated several architectures from literature and have set up a complete evaluation campaign to complete the ASDC dataset. As expected, deeper and complex model architecture provides the best accuracy but are not suitable for on-board deployment. We have highlighted that small models with simple optimization strategy can be deployed and offer a good compromise between accuracy and on-board performance. This work required the mastering on several aspects from the early phase of the application design to the on-board testing on SoC FPGA devices. A complete bibliography is also presented along this document to illustrate our argumentation and statements. Last but not least, we successfully developed and tested the first prototype of our Enki technology. This first prototype doesn't include optimization to enhance its performance, the latency results are lower than expected. On the other hand, the current resource usage on the selected devices shows that we have a lot of available hardware to do so. Many perspectives are under discussion to follow up this work. We first want to instrument and explore a complete comparison analysis with the Xilinx DPU including some energy consumption measurements. Even if Enki offers limited latency compared to Xilinx DPU, we believe that the overall pixel per Watt ratio is still good due to our resource usage and clock settings. In addition, we must complement this with a thorough analysis on the impact of the configuration parameters for ENKI for energy efficiency (such has been done on the Xilinx DPU [66]). From the application point of view, we are currently extending our work on to panoptic segmentation strategies and applying it to PASTIS dataset [16]. Our first results are very encouraging and we may combines it the use of CNN with RNN concepts in ConvLSTM [60] to extract properly the temporal relation in the image sequences.

REFERENCES

- [1] *Space DPU: Constructing a Radiation-Tolerant, FPGA-based Platform for Deep Learning Acceleration on Space Payloads*. Zenodo, November 2021.
- [2] *Survey of High-Performance Processors and FPGAs for On-Board Processing and Machine Learning Applications*. Zenodo, November 2021.
- [3] Intel (Altera). Fpga ai suite - ip reference manual. Technical Report R2023a-R2023b, 2023.
- [4] Shunichi Amari. A theory of adaptive pattern classifiers. *IEEE Transactions on Electronic Computers*, EC-16(3):299–307, 1967.
- [5] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(12):2481–2495, 2017.
- [6] Jean-Baptiste Chaudron and Alfonso Mascarenas-Gonzalez. Exploring segnet architectures for igpu embedded devices. In *NCTA 2023 - 15th International Joint Conference on Neural Computation Theory and Applications*, 2023.
- [7] Kyunghyun Cho, Bart van Merriënboer, Çağlar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In Alessandro Moschitti, Bo Pang, and Walter Daelemans, editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1724–1734. ACL, 2014.
- [8] Saed Daoud, Geoffrey Eappen, Flor Ortiz, Eva Lagunas, Wallace Martins, and Symeon Chatzinotas. Cnn-based on-board interference detection in satellite systems: An analysis of dataset impact on performance. In *2023 IEEE International Conference on Acoustics, Speech, and Signal Processing Workshops (ICASSPW)*, pages 1–5, 2023.
- [9] Richard O Duda and Peter E Hart. Use of the hough transformation to detect lines and curves in pictures. *Communications of the ACM*, 15(1):11–15, 1972.
- [10] Stefan Elfving, Eiji Uchibe, and Kenji Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural Networks*, 107:3–11, 2018. Special issue on deep reinforcement learning.
- [11] Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990.
- [12] Farah Fahim, Benjamin Hawks, Christian Herwig, James Hirschauer, Sergio Jindariani, Nhan Tran, Luca P. Carloni, Giuseppe Di Guglielmo, Philip C. Harris, Jeffrey D. Krupa, Dylan S. Rankin, Manuel Blanco Valentin, Josiah D. Hester, Yingyi Luo, John Mamish, Seda Ogrenci Memik, Thea Aarrestad, Hamza Javed, Vladimir Loncar, Maurizio Pierini, Adrian Alan Pol, Sioni Summers, Javier M. Duarte, Scott Hauck, Shih-Chieh Hsu, Jennifer Ngadiuba, Mia Liu, Duc Hoang, Edward Kreinar, and Zhenbin Wu. hls4ml: An open-source codesign workflow to empower scientific low-power machine learning devices. *CoRR*, abs/2103.05579.
- [13] Fares Fourati and Mohamed-Slim Alouini. Artificial intelligence for satellite communication: A review. *Intelligent and Converged Networks*, 2(3):213–243, 2021.
- [14] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36:193–202, 1980.
- [15] Gianluca Furano, Gabriele Meoni, Aubrey Dunne, David Moloney, Veronique Ferlet-Cavrois, Antonis Tavoularis, Jonathan Byrne, Lonie Buckley, Mihalis Psarakis, Kay-Obbe Voss, and Luca Fanucci. Towards the use of artificial intelligence on the edge in space systems: Challenges and opportunities. *IEEE Aerospace and Electronic Systems Magazine*, 35(12):44–56, 2020.
- [16] Vivien Sainte Fare Garnot and Loic Landrieu. Panoptic segmentation of satellite image time series with convolutional temporal attention networks, 2022. <http://arxiv.org/abs/2107.07933> arXiv:2107.07933.
- [17] Felix A. Gers, Jürgen A. Schmidhuber, and Fred A. Cummins. Learning to forget: Continual prediction with lstm. *Neural Comput.*, 12(10):2451–2471, 2000.
- [18] Max Ghiglione and Vittorio Serra. Opportunities and challenges of ai on satellite processing units. In *Proceedings of the 19th ACM International Conference on Computing Frontiers, CF '22*, page 221224, New York, NY, USA, 2022. Association for Computing Machinery.
- [19] Gianluca Giuffrida, Luca Fanucci, Gabriele Meoni, Matej Bati, Lonie Buckley, Aubrey Dunne, Chris van Dijk, Marco Esposito, John Hefe, Nathan Vercruyssen, Gianluca Furano, Massimiliano Pastena, and Josef Aschbacher. The -sat-1 mission: The first on-board deep neural network demonstrator for satellite earth observation. *IEEE Transactions on Geoscience and Remote Sensing*, 60:1–14, 2022.
- [20] Gianluca Giuffrida, Luca Fanucci, Gabriele Meoni, Matej Bati, Lonie Buckley, Aubrey Dunne, Chris van Dijk, Marco Esposito, John Hefe, Nathan Vercruyssen, Gianluca Furano, Massimiliano Pastena, and Josef Aschbacher. The -sat-1 mission: The first on-board deep neural network demonstrator for satellite earth observation. *IEEE Transactions on Geoscience and Remote Sensing*, 60:1–14, 2022.
- [21] Kaiyuan Guo, Shulin Zeng, Jincheng Yu, Yu Wang, and Huazhong Yang. [dl] a survey of fpga-based neural network inference accelerators. *ACM Transactions on Reconfigurable Technology and Systems*, 12(1), mar 2019.
- [22] Sandi Habinc. Suitability of reprogrammable fpgas in space applications. 2002.
- [23] William Grant Hatcher and Wei Yu. A survey of deep learning: Platforms, applications and emerging research trends. *IEEE Access*, 6:24411–24432, 2018.
- [24] Kaiming He, Georgia Gkioxari, Piotr Dollr, and Ross Girshick. Mask r-cnn. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2980–2988, 2017.
- [25] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, 2015.
- [26] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034, 2015.
- [27] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.
- [28] Jeff Inversion and Martin Faudi. Airbus ship detection challenge, 2018. URL: <https://kaggle.com/competitions/airbus-ship-detection>.
- [29] Irie and Miyake. Capabilities of three-layered perceptrons. In *IEEE 1988 International Conference on Neural Networks*, pages 641–648 vol.1, 1988.
- [30] A.G. Ivakhnenko and V.G. Lapa. Cybernetic predicting devices. Technical Report 37803, U.S. Department of Commerce, 1966.
- [31] M I Jordan. Serial order: a parallel distributed processing approach. technical report, june 1985-march 1986.
- [32] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.
- [33] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- [34] Vasileios Leon, Ioannis Stamoulias, George Lentaris, Dimitrios Soudris, David Gonzalez-Arjona, Ruben Domingo, David Merodio Codinachs, and Isabelle Conway. Development and testing on the european space-grade brave fpgas: Evaluation of ng-large using high-performance dsp benchmarks. *IEEE Access*, 9:131877–131892, 2021.
- [35] Seppo Linnainmaa. Taylor expansion of the accumulated rounding error. *BIT Computer Science and Numerical Mathematics*, 16(2):146160, june 1976.
- [36] Shuying Liu and Weihong Deng. Very deep convolutional neural network based image classification using small training sample size. In *2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR)*, pages 730–734, 2015.
- [37] Ilya Loshchilov and Frank Hutter. Fixing weight decay regularization in adam. *CoRR*, abs/1711.05101, 2017.
- [38] Inc MathWorks. Hdl coder - evaluation reference guide. Technical Report R2023a-R2023b, 2023.
- [39] Bho Matthiesen, Nasrin Razmi, Israel Leyva-Mayorga, Armin Dekorsy, and Petar Popovski. Federated learning in satellite constellations. *IEEE Network*, pages 1–16, 2023.
- [40] Warren McCulloch and Walter Pitts. A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:127–147, 1943.

- [41] Microchip (Microsemi). Corevectorblox handbook. Technical Report HB0919, November 2020.
- [42] Shervin Minaee, Yuri Boykov, Fatih Porikli, Antonio Plaza, Nasser Kehtarnavaz, and Demetri Terzopoulos. Image segmentation using deep learning: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(7):3523–3542, 2022.
- [43] Alessandro Pappalardo. Xilinx/brevitas, 2023.
- [44] Alessandro Pappalardo, Yaman Umuroglu, Michaela Blott, Jovan Mitrevski, Ben Hawks, Nhan Tran, Vladimir Loncar, Sioni Summers, Hendrik Borras, Jules Muhizi, Matthew Trahms, Shih-Chieh Hsu, Scott Hauck, and Javier Duarte. Qonnx: Representing arbitrary-precision quantized neural networks, 2022. <http://arxiv.org/abs/2206.07527> arXiv:2206.07527.
- [45] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. *CoRR*, abs/1912.01703, 2019. <http://arxiv.org/abs/1912.01703> arXiv:1912.01703.
- [46] Krishna Patel, Chintan Bhatt, and Pier Luigi Mazzeo. Deep learning-based automatic detection of ships: An experimental study using satellite images. *Journal of Imaging*, 8(7), 2022.
- [47] Michael Petry, Patrick Gest, Andreas Koch, Max Ghiglione, and Martin Werner. Accelerated deep-learning inference on fpgas in the space domain. In *Proceedings of the 20th ACM International Conference on Computing Frontiers*, CF '23, page 222228, New York, NY, USA, 2023. Association for Computing Machinery.
- [48] Michael Petry, Patrick Gest, Andreas Koch, Max Ghiglione, and Martin Werner. Accelerated deep-learning inference on fpgas in the space domain. In *Proceedings of the 20th ACM International Conference on Computing Frontiers*, CF '23, page 222228, New York, NY, USA, 2023. Association for Computing Machinery.
- [49] Tobiasz Rafal, Wilczynski Grzegorz, Graszka Piotr, Czechowski Nikodem, and Luczak Sebastian. Edge devices inference performance comparison. *Journal of Computing Science and Engineering*, 17(2):5159, June 2023.
- [50] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for activation functions, 2017. <http://arxiv.org/abs/1710.05941> arXiv:1710.05941.
- [51] Albert Reuther, Peter Michaleas, Michael Jones, Vijay Gadepally, Siddharth Samsi, and Jeremy Kepner. Survey and benchmarking of machine learning accelerators. In *2019 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–9, 2019.
- [52] Brianna I. Robertson and Aaron Smith. Optimizing space communications using deep learning. In *2021 IEEE Cognitive Communications for Aerospace Applications Workshop (CCAAW)*, pages 1–8, 2021.
- [53] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In Nassir Navab, Joachim Hornegger, William M. Wells, and Alejandro F. Frangi, editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pages 234–241, Cham, 2015. Springer International Publishing.
- [54] Frank Rosenblatt. The Perceptron - A Perceiving and Recognizing Automaton. Technical Report 85-460-1, Cornell Aeronautical Laboratory, 1957.
- [55] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–408, 1958.
- [56] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. *Learning Internal Representations by Error Propagation*, page 318362. MIT Press, Cambridge, MA, USA, 1986.
- [57] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.
- [58] Antonia Russo and Gianluca Lax. Using artificial intelligence for space challenges: A survey. *Applied Sciences*, 12(10), 2022.
- [59] S. Salcedo-Sanz, P. Ghamisi, M. Piles, M. Werner, L. Cuadra, A. Moreno-Martnez, E. Izquierdo-Verdiguier, J. Muñoz-Mar, Amirhosein Mosavi, and G. Camps-Valls. Machine learning information fusion in earth observation: A comprehensive review of methods, applications and data sources. *Information Fusion*, 63:256–272, 2020.
- [60] Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-kin Wong, and Wang-chun Woo. Convolutional lstm network: A machine learning approach for precipitation nowcasting. NIPS'15, page 802810, Cambridge, MA, USA, 2015. MIT Press.
- [61] Stefano Silvestrini and Michle Lavagna. Deep learning and artificial neural networks for spacecraft dynamics, navigation and control. *Drones*, 6(10), 2022.
- [62] Windy S. Slater, Nayana P. Tiwari, Tyler M. Lovelley, and Jesse K. Mee. Total ionizing dose radiation testing of nvidia jetson nano gpus. In *2020 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–3, 2020.
- [63] Juan Terven, Diana M. Cordova-Esparza, Alfonso Ramirez-Pedraza, and Edgar A. Chavez-Urbiola. Loss functions and metrics in deep learning, 2023. <http://arxiv.org/abs/2307.02694> arXiv:2307.02694.
- [64] Massimo Tipaldi, Raffaele Iervolino, and Paolo Roberto Massenio. Reinforcement learning in spacecraft control applications: Advances, prospects, and challenges. *Annual Reviews in Control*, 54:1–23, 2022.
- [65] Yaman Umuroglu, Nicholas J. Fraser, Giulio Gambardella, Michaela Blott, Philip Heng Wai Leong, Magnus Jahre, and Kees A. Vissers. FINN: A framework for fast, scalable binarized neural network inference. *CoRR*, abs/1612.07119, 2016. <http://arxiv.org/abs/1612.07119> arXiv:1612.07119.
- [66] Jurgen Vandendriessche, Bruno Da Silva, and Abdellah Touhafi. Frequency evaluation of the xilinx dpu towards energy efficiency. In *IECON 2022 48th Annual Conference of the IEEE Industrial Electronics Society*, pages 1–6, 2022.
- [67] Teng Wang, Chao Wang, Xuehai Zhou, and Huaping Chen. An overview of fpga based deep learning accelerators: Challenges and opportunities. In *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pages 1674–1681, 2019.
- [68] Paul J. Werbos. Applications of advances in nonlinear sensitivity analysis. In R. F. Drenick and F. Kozin, editors, *System Modeling and Optimization*, pages 762–770, Berlin, Heidelberg, 1982. Springer Berlin Heidelberg.
- [69] Saining Xie, Ross Girshick, Piotr Dollr, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5987–5995, 2017.
- [70] Xilinx. Zynq-7000 soc data sheet: Overview. Technical Report DS190, v1.11.1, July 2018.
- [71] Xilinx. Zynq ultrascale+ mpsoe data sheet: Overview. Technical Report DS891, v1.10, February 2022.
- [72] Xilinx. Kria k26 som data sheet. Technical Report DS987, v1.4.1, July 2023.
- [73] Xilinx. Dpucahx8l for convolutional neural networks - product guide. Technical Report PG366, v1.0, March 2024.

Session Tu.1.B

Realtime System 1

Tuesday 11th June

11:30

–

Lauragais Room

Runtime Performance Evaluation of a Non-Preemptive Cooperative Multithreading Framework Through Tracing

Lea Jungmann, Zain A. H. Hammadeh, Jan Sommer, Daniel Lüdtkke

*Institute for Software Technology
German Aerospace Center (DLR)*

Braunschweig, Germany

lea.jungmann@dlr.de, zain.hajhammadeh@dlr.de, jan.sommer@dlr.de, daniel.luedtke@dlr.de

Abstract—In the aerospace and automotive domains, there is a growing trend towards delegating more tasks to embedded software, employing sophisticated algorithms and machine learning-based solutions. As a result of this trend, the complexity of embedded software is escalating rapidly. Classical performance analysis methods, such as static worst-case execution time analysis, struggle to cope with this complexity without providing prohibitively over-approximated upper bounds.

In this paper, we introduce a tracing-based performance analysis approach tailored to data flow space applications. We illustrate how traces are leveraged to extract arrival curves, minimum distance functions, and execution times. We showcase the utility of tracing in design decisions using an aerospace use case, e.g., optimising the number of cores to reduce end-to-end latency. Furthermore, we extracted and presented debugging information graphically. While our tracing-based performance analysis may introduce overhead on the extracted timing properties, such as worst-case execution time, this overhead is bounded by 6.5%. Finally, we demonstrated the efficacy of our proposed tracing-based analysis approach through its application in a space application scenario.

I. INTRODUCTION

Modern space applications, including Earth observation, in-orbit servicing, and autonomous spacecraft and rover missions on distant celestial bodies, entail intensive on-board data processing and sophisticated control algorithms. These applications can become very complex, with high requirements for reliability and performance. The high demand for small satellites, such as cube-sats, necessitates more modular and reusable software that meets mission requirements, including timing requirements.

Multi-core platforms can offer high performance with low power consumption compared to single-core platforms. However, the parallel execution and simultaneous access to shared resources on multi-core platforms introduce additional complexity to embedded software. Furthermore, reading from sensors involves a significant time delay relative to computing time. Although self-suspending processes are employed for sensor reading, they contribute to more intricate and less predictable timing models. Event-driven execution models,

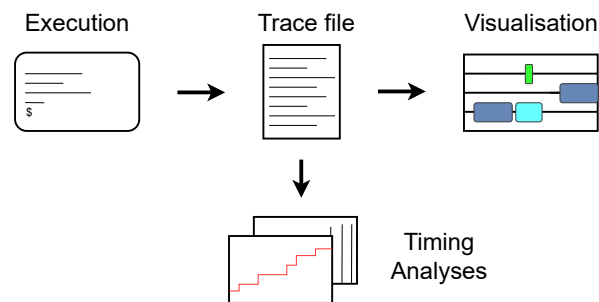


Fig. 1. Workflow of the tracing discussed in this paper: the execution of a program equipped with tracing results in a trace file which can either be graphically displayed or further analysed.

such as the publisher-subscriber model in Robotic Operating System version 2 (ROS2) [1], are also commonly employed for improved data predictability. In this case, a common industry practice is to assign tasks to a pool of threads where the threads cooperate to execute the tasks under a non-preemptive manner [1], [2]. Nevertheless, these models do not simplify timing considerations; instead, they introduce their own complexities.

Measurement-based performance analyses are widely utilized in the industry; however, they cannot guarantee complete coverage of all corner cases. Static methods, on the other hand, can offer formal guarantees on performance and are primarily employed for safety-critical applications. Despite providing over-approximated results, static methods struggle to smoothly scale with the complexity of modern hardware/software. Contemporary research often leans towards proposing hybrid approaches to address the heightened complexity of modern hardware/software and compute reliable guarantees. Tracing emerges as a versatile approach to extract crucial runtime information from complex embedded software to enhance formal methods. For instance, tracers have been used to define the activation pattern of tasks in the Real-Time Calculus (RTC) [3] approach and the Symbolic Timing Analysis for Systems

(SymTA/S) [4] approach. This technique finds application in various domains, such as robotics [5] and automotive [6], where it aids in extracting timing properties and establishing precedence relations between software components. Tracing is preferred over regular debugging, as the latter may lead to breakpoints violating timing requirements or skewing the observed performance.

In this work, we aim to extract timing properties of applications with complex timing behaviour, namely applications executed by cooperating thread pools. For that end, our proposal utilizes a tracing mechanism. Also, we present how to visualise these traces, and how to extract debugging information from them, using open-source tools. An overview of our proposed workflow is summarised in Fig.1. Our work employs the Common Trace Format (CTF) [7] to write traces, and the TraceCompass [8] to visualise the traces. Also, we use Babel-trace [9] to extract timing properties from the traces. We implemented our proposed tracing mechanism on an event-driven multithreading framework, namely Tasking Framework [2]. As our applications are intended to run on different operating systems, primarily Linux and RTEMS, we are focused on developing a cross-platform tracing mechanism.

In the following section, we explore the related work. In Section III, we briefly introduce Tasking Framework and its main features. In Section IV, we elaborate on the implementation of the tracing mechanism. Section V presents our approach to extract timing properties. The overhead of the proposed tracing mechanism is discussed in Section VI. We demonstrate the applicability of the proposed approach on a realistic case study in Section VII. Finally, Section VIII concludes the paper.

II. STATE OF THE ART

The extraction of runtime information is vital for the developing process. Knowing the execution behaviour is key to debugging and, later on, the optimisation of a system. An established way of extracting runtime information is tracing. Tracing records the behaviour of a system during its execution by placing hooks, called tracepoints, in the code [10]. At its core, tracing produces a trace file that can be read and analysed after it is produced [11]. Depending on the tracepoints and their eventual content, the runtime information gained with tracing can vary, depending on the observed system. If the system structure is not known or only known partially, the focus when extracting runtime information may lie in getting a more complete system model such as in [12], [13]. In other cases, such as in [5], runtime information, such as response times, is collected to aid in analysing timing behaviour.

Facing unknown behaviour in a real-time system, [12]’s approach uses execution traces and a task definition to model the system’s runtime behaviour as a set of independent periodic tasks. All tasks occurring in a given execution trace are categorised as either periodic or non-periodic. Furthermore,

they extract additional information on the periodic tasks such as their period and response time profile.

Usually, traces are extracted using Tracers, tools that use already existing or custom hooks to instrument the code for recording during runtime. A popular tracer for Linux applications is the Linux Trace Toolkit: next Generation (LTTng) [14], that is capable of tracing processes both in the kernel and user space. For kernel tracing, it uses tracepoints already embedded in the Linux kernel. For user space tracing, LTTng needs the application to be instrumented using either LTTng-style tracepoints or Java or Python logging statements that are then fed to a LTTng handler. LTTng uses a binary format called Common Trace Format (CTF) [7] to write its traces in a compact manner. LTTng is also used as a basis for other tracing tools, such as in [5], which presents a range of multi-purpose tracing tools for the ROS 2 that use LTTng as their tracing backend. LTTng is used because it has both user and kernel space tracing capabilities, making the trace as comprehensive as possible for Linux applications, as well its low overhead and real-time compatibility.

[13] employs the extended Berkeley Packet Filter (eBPF) for tracing in ROS 2. Other than LTTng, it does not require direct instrumentation, which would lead to having to recompile ROS2 standard libraries. Tracing is used in [13] to extract the flow of information within the system, since this information may not be directly accessible in industry scenarios. It does so by identifying and tracking ROS 2 nodes and callbacks during execution.

LTTng only runs on Linux systems, making it unsuitable for cross-platform applications. However, the format that LTTng uses, CTF, is an open standard that is intelligible to both industry tools such as Tracealyzer [15] and open-source solutions like TraceCompass [8]. Its binary nature makes CTF a very compact format already, its high flexibility regarding form and content of individual events, instances of tracepoints being passed, allows to control the amount of overhead and size of the resulting trace file.

To cope with the emerged challenges from using new programming language like Rust, Wang et al. proposed in [16] a context aware tracing for estimating the execution time of asynchronous tasks. The main concern of [16] is the Rust programs that are implemented as coroutines. Hence, the applicability of [16] is limited to Rust asynchronous programs.

III. TASKING FRAMEWORK

The Tasking Framework is an open-source¹ non-preemptive, cooperative multithreading C++ framework and execution platform, mainly used in the development of space applications [2]. It is being developed by the German Aerospace Center. While it supports different platforms such as Linux,

¹<https://github.com/DLR-SC/tasking-framework>

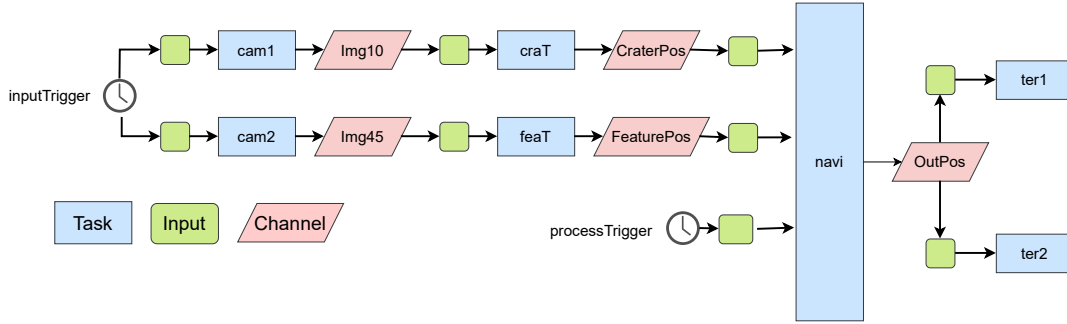


Fig. 2. The optical navigation subsystem in ATON [17] in Tasking Framework.

RTEMS and FreeRTOS, it can also be run on bare-metal. Tasking Framework has been used in several space projects, including Autonomous Terrain-based Optical Navigation (ATON) [17], Euglena Combined Organic food Production In Space (Eu:CROPIS) [18], and Scalable On-Board Computing for Space Avionics (ScOSA) [19].

In Tasking Framework, applications are modelled as a graph of tasks, channels and inputs, as can be seen in Fig.2, which models the optical navigation subsystem of ATON. This task and channel model is modelled after Petri nets, with tasks analogous to transitions and channels likened to places. As with Petri nets, channels and tasks are connected through inputs. Channels can be thought of as data storage while tasks are processing units that take their input from and push their output to channels. Once data is pushed on a channel, the inputs that connect tasks to the channel are notified of the new data on the channel. This may lead to the activation of the connected tasks. In addition, there are also events. Events are used to either periodically trigger a task or to trigger the task after a time-out.

The point in time at which a task is activated, that means marked as ready to be executed, depends on the activation model used for this task. Tasking Framework supports different activation models, meaning the conditions of activation can be chosen individually for each task. Thus, a task may wait for a push on all, one or some channels that it is connected to or may require multiple pushes on a channel before activation. The activation model for a task is chosen at compile time, however, the task barrier structure, a specialised kind of channel, may be used if the amount of pushes required for task activation has to be changed during runtime. The default call semantic for tasks is asynchronous, however, the task group structure can be used to implement synchronicity among a group of tasks, meaning that a task once executed can only be executed again after all other tasks in the group have also executed regardless of its own activation status.

When a task is activated, it will be queued for execution. Tasks are executed using a pool of threads, called *executors*, that collaborate on the execution of tasks. Tasks are executed

by Tasking Framework in *non-preemptive* manner. Fig. 3 illustrates the execution model in Tasking Framework. There are three scheduling policies supported in Tasking Framework, namely First-In First-Out (FIFO); Last-In First-Out (LIFO); Fixed Priority. The scheduling is work-conservative, i.e., there is no idle executor as long as the ready queue is not empty. Executors collaborate in a load-balancing manner and every task can be executed by any available executor.

Currently, the application programming interface (API) of Tasking Framework supports only C++. Developing applications using different programming languages is not supported up to now.

IV. IMPLEMENTATION

This section outlines how a tracing mechanism was integrated into Tasking Framework and how it can be displayed with TraceCompass [8]. Tracing the Tasking Framework is reliant on the instrumentation of its code, i.e. the hooks placed in the source code to record a Tasking Framework application. We use the Common Trace Format (CTF) [7], which is a flexible and lightweight binary format, to write traces. The Tasking Framework is large enough that it would cause too much overhead to record every single action that is executed during the runtime. Thus, one must identify a configuration of points within Tasking Framework that give an accurate picture of the inner happenings of the framework. Preferably, with as few points as possible as to avoid causing too much overhead. These tracepoints are:

- 1) A push on a channel. The push on a channel happens whenever new data is made available to the channel. In turn, all connected inputs are notified informing the connected tasks that new data is available on the channel.
- 2) Activation of a task. The activation of a task signals that a task is ready to be executed and has been queued by the scheduler to wait for the next free execution slot.
- 3) Task starts & stops executing. This shows how long the task had to wait before being executed and how long it was executed.

Pushes are triggers for task activation and their presence or absence in a trace can contribute to error searches. The timing

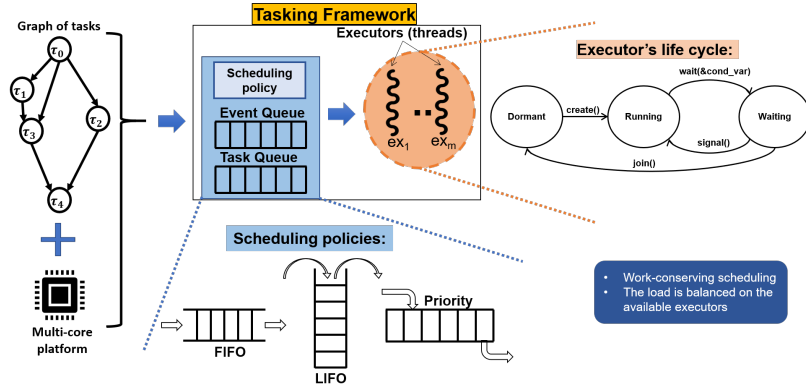


Fig. 3. The execution model in Tasking Framework.

information on the activation, start and stop of the execution of a task can provide information on execution and queuing wait times. The tracer class is implemented as a singleton to prevent conflicting write processes, especially when multiple threads are running and generating events at the same time. Customising the trace format also gives control over the amount of overhead produced by the tracer because custom events can use only the exact amount of data needed and do not have to fill fields with empty data to conform to standards. Custom events pose the question of how exactly their payloads are supposed to look like. Keeping the payload small is imperative to keeping the overhead small. Each tracepoint generates an event upon execution. Hence, the four custom events are:

- push on a channel λ_l , denoted by π_l
- activation of a task τ_i , denoted by α_i
- start of a task execution, denoted by $\sigma_{\uparrow i}$
- stop of a task execution, denoted by $\sigma_{\downarrow i}$

We introduce the trace θ as a finite set of these events.

Each of these events requires at least the identification number of the relevant task or channel in order to match the events to their corresponding tasks or channels. While this is enough to complete necessary calculations and calculate corresponding graphs, these graphs are not particularly readable for humans. The number associated to each task is not a speaking name and would require the developer or user to look up the numbers in Tasking Framework in a time-inefficient manner. To prevent this, the payload of each custom event includes not only the identification number of a task but also its four-character name that is used for display purposes and for user interaction.

We use TraceCompass [8] to visualise our traces. TraceCompass is an Eclipse Rich Client Platform (RCP) tool to read, visualise and analyse traces. TraceCompass provides a variety of charts. These charts allow for inspecting, measuring and analysing the opened trace. Like all Eclipse RCPs, TraceCompass can be modularly expanded with the help of

plug-ins to add more functionality. Plug-ins for TraceCompass include plug-ins for additional analyses, scripting, global filters and support for additional trace types by different tracers and profilers. Since TraceCompass is built and specialised on Linux kernel and user space traces, it does not include many charts for custom traces. In fact, a completely custom CTF trace imported into TraceCompass will get two charts generated by TraceCompass, the Statistics chart, that shows the absolute and relative frequencies of the events in the trace, and a list of all events and their payloads contained in the trace, that is by default chronologically ordered. While manageable for very small traces, these two charts are not very helpful when used with traces that contain more than a handful of events. Additionally, while CTF does not have a limitation on the length of individual traces, there is a limit of about 1.6 million events that can be loaded into and displayed in TraceCompass. For traces containing more events, there are other tools such as Babeltrace [9], that are able to handle trace files of that size.

Making a custom graph is possible using a Python script and the EASE scripting module integrated into TraceCompass. The script takes the currently opened trace as input and iterates over the events. Each event whose name can be matched to one of the defined custom events is used to extract its *quark*, which in TraceCompass stands for a unique identifier for an object. In this case, the quark is generated from the event name, meaning the events are sorted by task or channel name. The event is then added to the state system using the quark and the timestamp to sort it to the right position. Once all eligible events are added to the state system, it is used to create a *TimeGraph*. TimeGraph lists all states, in this case tasks and channels, on the left side of the diagram while using a timeline as x-axis. This custom TimeGraph will be referred to as *Tasking Graph*. An example of a Tasking Graph can be seen in Fig.4, which displays the Tasking Graph of a trace of a skeleton implementation of the ATON optical navigation subsystem shown in Fig.2. This means, each task or channel

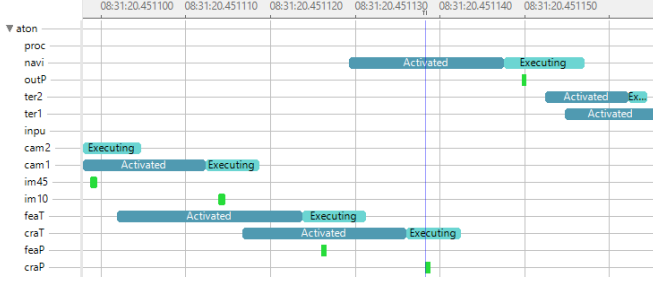


Fig. 4. Tasking Graph of the use case shown in Fig.2.

gets its own individual timeline that displays how the object changes states during the runtime. Activation and Execution periods for each task (light and dark blue respectively) are displayed on task timelines, while channel timelines display pushes (green).

V. EXTRACTING TIMING PROPERTIES

Next to the graphical representation of the trace, which can be useful for debugging, the generated trace can give more insights into the system behaviour. This information can then be used to decide whether a system needs traffic shaping or reconfiguring. The tracepoints on task activation and start of execution help to determine how long a task has to wait in each instance before it gets executed while the start and end times give insight to the execution time of a task. Let t^{event} represent the timestamp of the *event*.

The instance k of τ_i experiences a queuing time q_i^k :

$$q_i^k = t^{\sigma\tau_i} - t^{\alpha_i} \quad (1)$$

Hence, the maximum queuing time that τ_i suffers is:

$$Q_i = \max\{q_i^k | \forall k \in \theta\} \quad (2)$$

The instance k of τ_i experiences an execution time c_i^k :

$$c_i^k = t^{\sigma\downarrow i} - t^{\sigma\tau_i} \quad (3)$$

The longest observable execution time of τ_i is:

$$C_i = \max\{c_i^k | \forall k \in \theta\} \quad (4)$$

When tracing the activations and executions of a task, one can also analyse the trace to study possibly emerging patterns in the task behaviour and use them to predict system behaviour. For systems that run, ideally, in perpetuity or for very long stretches of time, tracing can only offer a snapshot of the system behaviour. However, graphical analysis is not the only analysis that can be applied to a trace. The recorded events of a Tasking Framework trace allow for the extraction of the following information: execution time of every instance of a particular task, activation times and queuing times, as well as push behaviour.

In practice, we use Babeltrace and its python bindings to iterate over the trace. This allows us to extract execution and

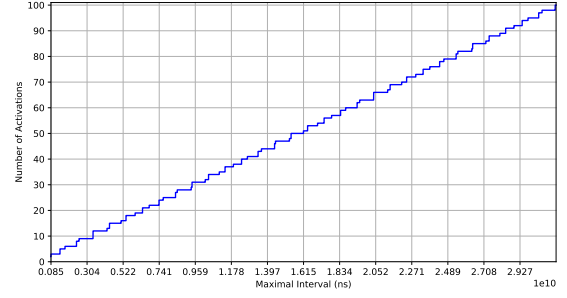


Fig. 5. Arrival curve $\eta^+(\Delta t)$ of the navigation task (navi) of ATON.

queuing times, as well as compute arrival curves and distance functions.

A. Arrival Curves

An arrival curve is a function that can be applied to a trace or any other timeline of events. The minimum and maximum arrival curves $\eta^-(\Delta t)$ and $\eta^+(\Delta t)$, are defined as functions on $\mathbb{R}^+ \rightarrow \mathbb{N}^+$, so that for any half-open time interval $[t, t + \Delta t)$ they return respectively either the minimum or maximum number of task activations α that can occur within the interval [20], [21]. An example of a maximum arrival curve can be seen in Fig.5. Arrival curves are *non-decreasing*, with $\eta^+(\Delta t)$ being *sub-additive*, meaning that the following is always true for $\eta^+(\Delta t)$:

$$\forall \Delta t, \Delta t' \in \mathbb{R}^+ : \eta^+(\Delta t + \Delta t') \leq \eta^+(\Delta t) + \eta^+(\Delta t') \quad (5)$$

B. Distance Functions

Distance functions are the pseudo-inverse of arrival curves. The minimum (*maximum*) distance function $\delta^-(n)$ (respectively $\delta^+(n)$) is defined on $\mathbb{N}^+ \rightarrow \mathbb{R}^+$ and returns the smallest (*largest*) time interval Δt that contains at least (*at most*) n events. An example for a minimum distance function can be seen in Fig.6. Minimum distance functions are non-decreasing and super-additive [21], meaning that every minimum distance function fulfils the following:

$$\forall n, n' \in \mathbb{N}^+ : \delta^-(n) + \delta^-(n') \leq \delta^-(n + n') \quad (6)$$

C. Extrapolating Trace Data

Exploiting the sub-additive and super-additive properties of $\eta^+(\Delta t)$ and $\delta^-(n)$, it is possible to extrapolate data for the behaviour of a traced system under the following assumption: the $\delta^-(2)$, which was observed within the trace, is also the global minimum. With this assumption in mind, distance functions can be extrapolated as follows:

$$\delta^-(n + 1) = \delta^-(n) + \delta^-(2). \quad (7)$$

While arrival curves can be extrapolated with:

$$\eta^+(\Delta t + \Delta t') = \eta^+(\Delta t) + \eta^+(\Delta t') \quad (8)$$

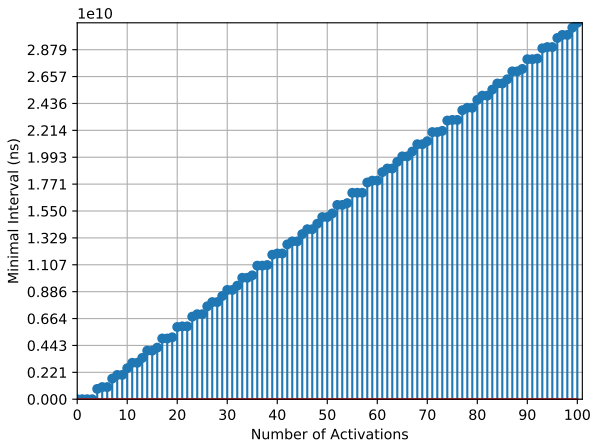


Fig. 6. Distance function $\delta^-(n)$ for the navigation task (navi) of ATON.

When using this extrapolation, it has to be noted that this is a safe extrapolation. Meaning, the extrapolated data has to be treated as lower or upper bound for the distance function and arrival curve respectively, meaning in case of a distance function that the extrapolated values represent the smallest possible value with no indication of an upper limit. Meanwhile, for the arrival curve the extrapolated data represents an upper bound with no indication for a lower bound.

VI. OVERHEAD OF TRACING

To measure the overhead caused by the tracing mechanism, example applications were profiled, resulting in a detailed overview of how much time the applications spent in which function. These example applications use dummy tasks, which are tasks with negligible computational effort, to replicate real-world applications in their structure. The offline overhead, i.e., the overhead of all functions that are involved in tracing, lies at around 15%-20%, depending on the profiler that is used. More relevant, however, is the overhead that is actively interfering with the timing behaviour of the tasks. The overhead of the tracing functions that affect the timing behaviour is caused by three functions, namely the ones that record pushes on a channel and the start and stop of a task executing. Their overhead sums up to 5.8%-6.5%, depending on the used profiler. Note, that the execution times of the tracing function are mostly fixed, since they always write the same amount of data. That means, if a task executes longer, the percentage of the overhead goes down accordingly. We computed the overhead using almost empty dummy tasks. Hence, the overhead shown here is likely to be an upper boundary of the possible overhead.

VII. USE CASE

We are considering a use case inspired by the optical navigation subsystem of the ATON project, utilizing the Linux operating system and ARM Cortex-A53 (1.2 GHz) as an

embedded quad core processor. The aim of the experiments in this section is to showcase the effectiveness of our proposed tracing mechanism as a performance analysis approach, focusing on extracting the timing properties of our use case. We conducted three experiments to exemplify how our tracing-based performance analysis is seamlessly integrated into the design process of safety-critical applications.

We have recorded a trace of 83523 events and extracted arrival curves and distance functions for event-driven tasks. Figures 5 and 6 illustrate $\eta^+(\Delta t)$ and $\delta^-(n)$ respectively for the navigation task (navi). Since the task is event-driven, its activation pattern is non-periodic. Our use case represents a graph of tasks. Therefore, we are interested not only in the timing properties of each task but also in the timing properties of different *chains* defined within the graph, specifically the *end-to-end latency*. We define a *chain* as the execution of a sequence of tasks from a *source* to a *sink*. In our use case, cam1 and cam2 are sources, and ter1 and ter2 are sinks. Hence, there are four chains. Let χ denote a chain, thus:

$$\begin{aligned} \chi_1 &: \text{cam1} \rightarrow \text{craT} \rightarrow \text{navi} \rightarrow \text{ter1} \\ \chi_2 &: \text{cam1} \rightarrow \text{craT} \rightarrow \text{navi} \rightarrow \text{ter2} \\ \chi_3 &: \text{cam2} \rightarrow \text{feaT} \rightarrow \text{navi} \rightarrow \text{ter2} \\ \chi_4 &: \text{cam2} \rightarrow \text{feaT} \rightarrow \text{navi} \rightarrow \text{ter1} \end{aligned}$$

A. Design decision 1: Platform

The goal of this experiment is to demonstrate the capability of our tracing mechanism to be cross-platform. Hence, beside the above mentioned settings (Linux + Cortex-A53), we compiled our case study to run on RTEMS using the GR712RC board with LEON3 processor (40 MHz), which is the default radiation-hardened processor for space systems. We present the execution time and queuing time experienced by each task considering FIFO scheduling with one executor in Fig. 7 for Linux + Cortex-A53 platform. Also, in Fig. 8 we consider FIFO scheduling with one executor for RTEMS + LEON3 platform. The end-to-end latency for both platforms is presented in Fig. 9. RTEMS, as an RTOS, produces results with less variation and suffering from less interference thanks to the RTOS kernel. Therefore, the range between the minimum and maximum values is smaller compared to the results obtained from the Linux OS. However, the maximum observable execution times are not improved, nor the queuing times. In fact, the third quartile values using the RTEMS + LEON3 platform for both the execution time and the queuing time are about 100 times larger than the third quartile values using Linux + Cortex-A53. The main reason is the very slow radiation-hardened processor (LEON3 with 40 MHz) which is about 30 times slower than the high-performance platform (Cortex-A53 with 1.2GHz). The need for more on-board processing power is a major concern for researchers and space companies. Many missions aim to integrate high-performance commercial off-the-shelf (COTS) processors alongside the radiation-hardened

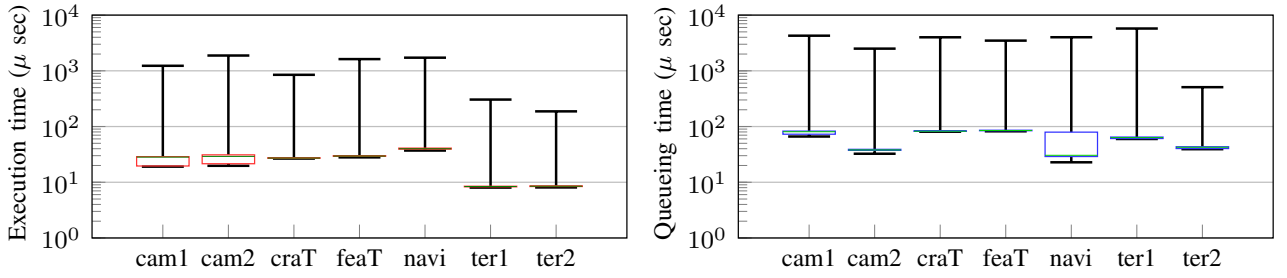


Fig. 7. The execution time and queuing time of the tasks in the use case under the FIFO scheduling using one executor on the Linux + Cortex-A53 platform.

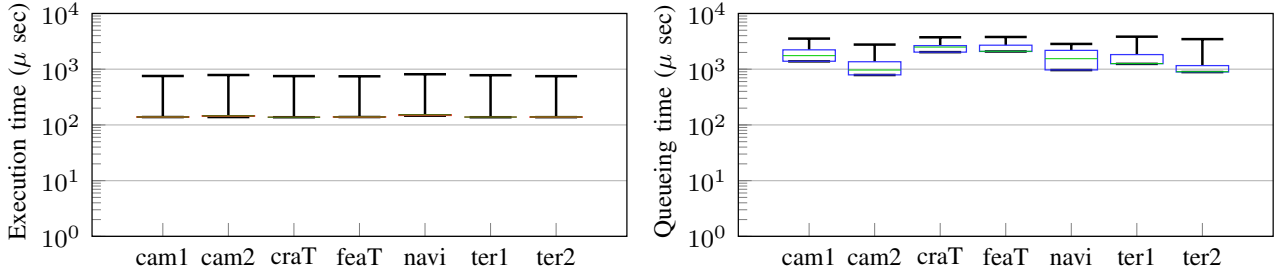


Fig. 8. The execution time and the queuing time of the tasks in the use case under the FIFO scheduling using one executor on the RTEMS + LEON3 platform.

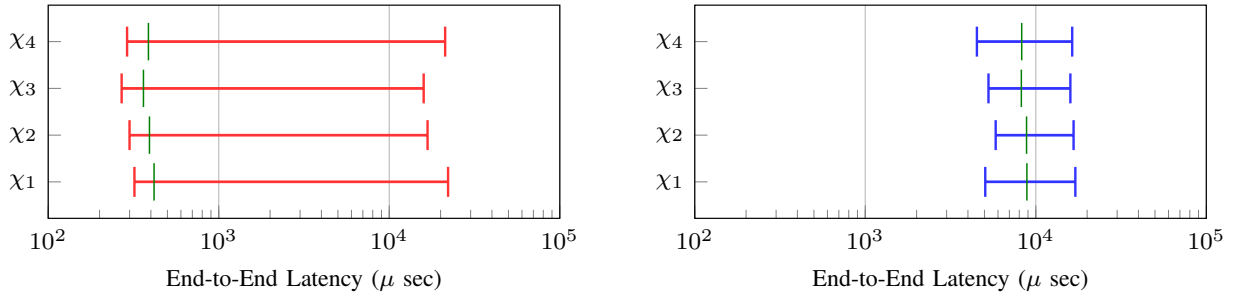


Fig. 9. The end-to-end latency of the chains under the FIFO scheduling considering one executor. In the left figure (in red), we consider the Linux + Cortex-A53 platform, and we consider the RTEMS + LEON3 platform for the right figure (in blue).

processors [22]–[24] to meet the required on-board processing power.

B. Design decision 2: Scheduling policy

In this experiment, we utilize our tracing mechanism to study the impact of scheduling policies and priority assignments on the timing behavior of tasks. Accordingly, we generated a new trace on Linux + Cortex-A53 architecture considering fixed priority scheduling, with priorities assigned as outlined in Table I². The new results are presented in Fig. 10. As the Tasking Framework executes tasks in a non-preemptive manner, the queuing time of tasks may exceed their execution time. However, employing priority scheduling reduces the queuing time and improves task execution times.

²We refer here to the FIFO and priority scheduling implemented in the Tasking Framework, as depicted in Fig. 3

TABLE I
PRIORITY ASSIGNMENTS WHERE 1 IS THE HIGHEST PRIORITY

Task	cam1	cam2	craT	feaT	navi	ter1	ter2
Priority	1	1	3	2	4	5	5

Consequently, the end-to-end latency is enhanced. Fig. 11 illustrates the end-to-end latency for FIFO scheduling (depicted in red on the left) and priority scheduling (shown in blue on the right). Under FIFO scheduling, data processed from cam1 to ter1, in χ_1 , experience the longest end-to-end latency. Conversely, under priority scheduling, χ_2 exhibits the longest latency.

With this experiment we show that it is possible to extract enough data using tracing to come to a sound decision regarding scheduling policy. In this paper, unless stated otherwise, we use FIFO scheduling for all other experiments.

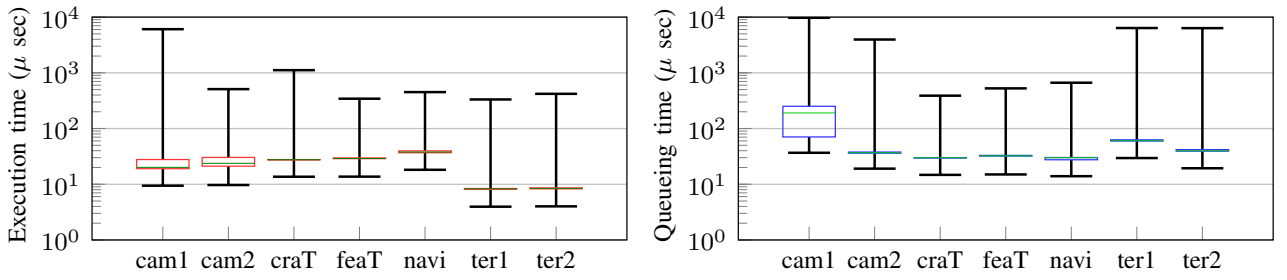


Fig. 10. The execution time and the queuing time of the tasks in the use case under the fixed priority scheduling using one executor on Linux + Cortex-A53 platform.

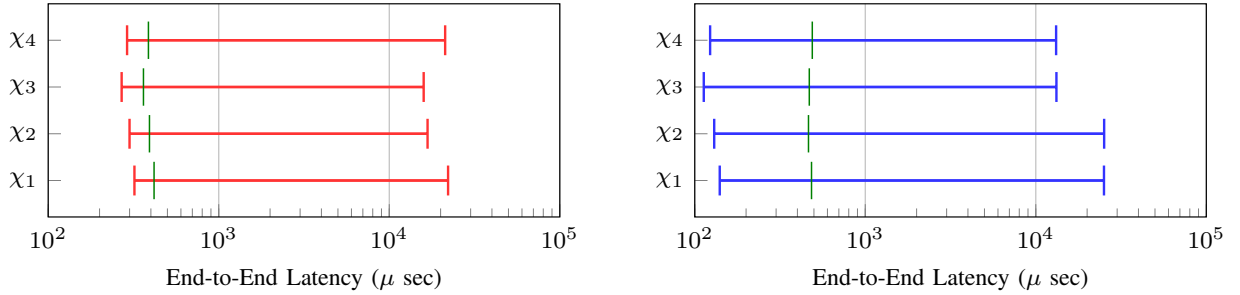


Fig. 11. The end-to-end latency of the chains considering one executor on the Linux + Cortex-A53 platform. In the left figure (in red), we consider the FIFO scheduling, and we consider the priority scheduling for the right figure (in blue).

C. Design decision 3: Number of executors

We aim to address a design question: how many cores should be allocated to the ATON optical navigation subsystem to strike a balance between delay and the number of cores? To achieve this, we executed our code under FIFO scheduling, considering two and three executors, respectively. The results are presented in Fig. 12 and Fig. 13. Comparing the results of one executor (Fig. 7) with two executors (Fig. 12), we observed that the execution times were better with two executors, but the maximum values increased. This can be attributed to the increased ratio of *cache misses* that tasks may experience when executed by two different executors on different cores. Consequently, the maximum queuing time of the tasks also increased significantly, although the minimum values improved. Fig. 14 demonstrates that the minimum end-to-end latency improved compared to the scenario with one executor under FIFO and priority scheduling. However, the maximum end-to-end latency increased significantly.

D. Comparing with static methods

Industrial embedded software are complex and formal methods cannot cope with it. Using languages like C, C++, RUST makes the analysis even more complicated. For instance, using virtual methods in C++ leads to indirect jumps, beside the indirect jumps caused by the switch-case statements and functions pointers in C and C++. Also, the objected-oriented programming in C++ makes bounding the loop more challenging for

tools depend on the source code like oRange [25]. Solutions that use dynamic symbolic execution, like e.g. DELOOP [26], can help us to resolve indirect jumps and compute safe bounds on the bounded loops. In [26], the dynamic symbolic execution was used to compute flow facts for Tasking Framework. These flow facts were forwarded to OTAWA [27] to compute the WCET of the Tasking Framework functions, for instance, the push function. The main drawbacks of dynamic symbolic execution based solutions that they are platform dependent. As DELOOP was developed for armv7 architecture, it cannot be used out of the box to compute the execution time for, e.g., X86 architecture. Using portable tracing based solution like our proposed solution can overcome the challenges emerged from different programming languages and it is platform independent. Table II shows the WCET of the push function in all tasks of our use case for ARM Cortex-M3. The results in Table II are more pessimistic than the results computed using the traces because they consider the longest execution path in the push function, which may not observable in the trace.

TABLE II
RESULTS OF THE WCET ANALYSIS FOR THE PUSH FUNCTION IN THE USE CASE

Task	WCET (cycles)
cam1	2435
cam2	2435
craT	3635
feaT	3635
navi	4800

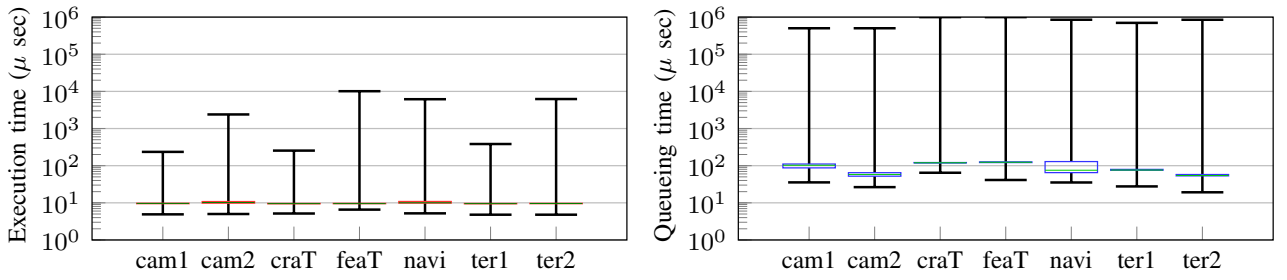


Fig. 12. The execution time and queuing time of the tasks in the use case under the FIFO scheduling using two executors on the Linux + Cortex-A53 platform.

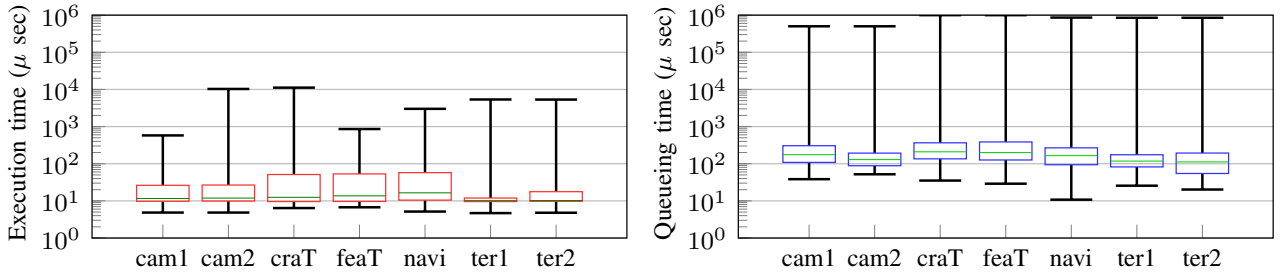


Fig. 13. The execution time and queuing time of the tasks in the use case under the FIFO scheduling using three executors on the Linux + Cortex-A53 platform.

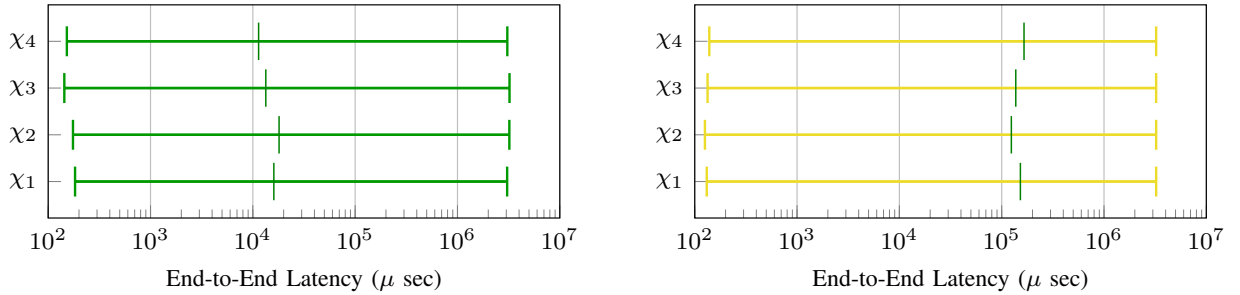


Fig. 14. The end-to-end latency of the chains considering the FIFO scheduling on the Linux + Cortex-A53 platform. In the left figure (in green), we use two executors, and we use three executors in the right figure (in yellow).

VIII. CONCLUSION AND OUTLOOK

With the increasing complexity of embedded software, especially on-board software, performance analysis using static methods faces the challenge of providing tight yet safe guarantees. Extracting timing properties using tracing is a promising technique to assist static methods to cope with the growing complexity of embedded software. In this work, we presented a tracing mechanism for performance analysis of data flow space applications that reuses open-source tools to offer a cross-platform solution. We showed how to use our solution to extract debugging and timing properties of a use case inspired by the optical navigation subsystem. Also, we studied the overhead of our solution.

Any tracing solution suffers from two main points: 1) the need for code instrumentation, 2) the overhead of the events.

As eliminating the two points is not realistic, reducing the overhead or the impact of the overhead on the measured parameters is a topic for future improvements. Additionally, code instrumentation can be automated using auto-code generators to guarantee less error-prone instrumentation. For this, we aim to employ the Timing Modeling Language (TML) [28] to automatically instrument the auto-generated code for our applications. In such a step, the developer can generate the traceable code and resulting traces with minimum effort and minimum human errors. Fig. 15 illustrates the TML modeling interface that would make such an automated approach possible.

REFERENCES

- [1] D. Casini, T. Blaß, I. Lütkebohle, and B. B. Brandenburg, "Response-time analysis of ROS 2 processing chains under reservation-based

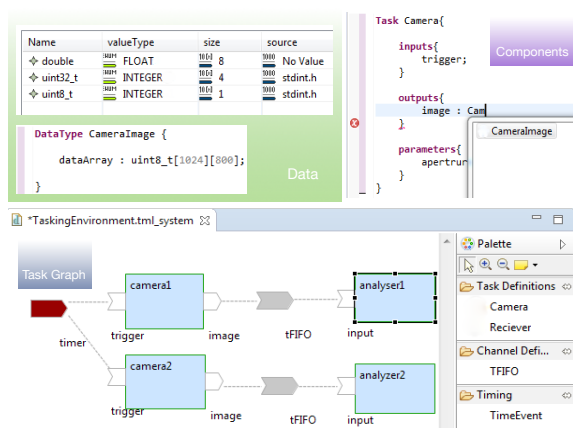


Fig. 15. The Timing Modeling Language (TML): an auto-code generator for Tasking Framework.

scheduling,” in *31st Euromicro Conference on Real-Time Systems (ECRTS 2019)*, vol. 133. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019, pp. 6:1–6:23.

- [2] Z. A. H. Hammadeh, T. Franz, O. Maibaum, A. Gerndt, and D. Lüdtkke, “Event-driven multithreading execution platform for real-time on-board software systems,” in *Proceedings of the 15th annual workshop on Operating Systems Platforms for Embedded Real-time Applications*, July 2019, pp. 29–34. [Online]. Available: <https://elib.dlr.de/128249/>
- [3] S. Chakraborty, S. Kunzli, and L. Thiele, “A general framework for analysing system properties in platform-based embedded system designs,” in *Proceedings of the Conference on Design, Automation and Test in Europe - Volume 1*, ser. DATE ’03. USA: IEEE Computer Society, 2003, p. 10190.
- [4] R. Henia, “System level performance analysis – the SymTA/S approach,” *IEE Proceedings - Computers and Digital Techniques*, vol. 152, pp. 148–166(18), March 2005. [Online]. Available: https://digital-library.theiet.org/content/journals/10.1049/ip-cdt_20045088
- [5] C. Bédard, I. Lütkebohle, and M. Dagenais, “ROS2_tracing: Multipurpose low-overhead framework for real-time tracing of ROS2,” *IEEE Robotics and Automation Letters*, vol. 7, no. 3, pp. 6511–6518, 2022.
- [6] S. Quinton, T. T. Bone, J. Hennig, M. Neukirchner, M. Negrean, and R. Ernst, “Typical worst case response-time analysis and its use in automotive network design,” in *Proceedings of the 51st Annual Design Automation Conference*, ser. DAC ’14. New York, NY, USA: Association for Computing Machinery, 2014, p. 1–6. [Online]. Available: <https://doi.org/10.1145/2593069.2602977>
- [7] M. Desnoyers. Common trace format (ctf) specification (v1.8.3). [Online]. Available: <https://diamon.org/ctf>
- [8] Tracecompass documentation. [Online]. Available: <https://www.eclipse.org/tracecompass/>
- [9] The babeltrace 2 documentation. [Online]. Available: <https://babeltrace.org/>
- [10] B. Cornelissen, A. Zaidman, A. Van Deursen, L. Moonen, and R. Koschke, “A systematic survey of program comprehension through dynamic analysis,” *IEEE Transactions on Software Engineering*, vol. 35, no. 5, pp. 684–702, 2009.
- [11] N. Ezzati-Jivan, G. Bastien, and M. R. Dagenais, “High latency cause detection using multilevel dynamic analysis,” in *2018 Annual IEEE International Systems Conference (SysCon)*. IEEE, 2018, pp. 1–8.
- [12] O. Iegorov, R. Torres, and S. Fischmeister, “Periodic task mining in embedded system traces,” in *2017 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2017, pp. 331–340.
- [13] H. Abaza, D. Roy, S. Fan, S. Saïdi, and A. Motakis, “Trace-enabled timing model synthesis for ROS 2-based autonomous applications,” in *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2024, pp. 1–6.
- [14] The Ittng documentation. [Online]. Available: <https://ittng.org/docs/v2.13/>
- [15] Percepio Tracealyzer. [Online]. Available: <https://percepio.com/tracealyzer/tracealyzer-for-linux/>
- [16] T.-Y. Wang, S.-H. Wang, C.-H. Tu, and W.-Y. Liang, “CAT: Context aware tracing for rust asynchronous programs,” in *Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing*, ser. SAC ’23. New York, NY, USA: Association for Computing Machinery, 2023, p. 483–492. [Online]. Available: <https://doi.org/10.1145/3555776.3577669>
- [17] S. Theil, N. A. Ammann, F. Andert, T. Franz, H. Krüger, H. Lehner, M. Lingenauber, D. Lüdtkke, B. Maass, C. Paproth, and J. Wohlfeil, “ATON (autonomous terrain-based optical navigation) for exploration missions: recent flight test results,” *CEAS Space Journal*, March 2018. [Online]. Available: <https://elib.dlr.de/119557/>
- [18] O. Maibaum and A. Heidecker, “Software evolution from TET-1 to Eu:CROPIS,” in *10th International Symposium on Small Satellites for Earth Observation*, R. Sandau, H.-P. Röser, and A. Valenzuela, Eds. Wissenschaft & Technik Verlag, April 2015, pp. 195–198. [Online]. Available: <https://elib.dlr.de/100859/>
- [19] A. Lund, Z. A. Haj Hammadeh, P. Kenny, V. Vishav, A. Kovalov, H. Watolla, A. Gerndt, and D. Lüdtkke, “ScOSA system software: the reliable and scalable middleware for a heterogeneous and distributed on-board computer architecture,” *CEAS Space Journal*, vol. 14, no. 1, pp. 161–171, 2022.
- [20] S. Künzli and L. Thiele, “Generating event traces based on arrival curves,” in *13th GIITG Conference-Measuring, Modelling and Evaluation of Computer and Communication Systems*. VDE, 2006.
- [21] Z. A. Haj Hammadeh, “Deadline miss models for temporarily overloaded systems,” Ph.D. dissertation, Technische Universität Braunschweig, 2019.
- [22] G. Lentaris, K. Maragos, I. Stratakos, L. Papadopoulos, O. Papanikolaou, D. Soudris, M. Lourakis, X. Zabulis, D. Gonzalez-Arjona, and G. Furano, “High-performance embedded computing in space: Evaluation of platforms for vision-based navigation,” *Journal of Aerospace Information Systems*, vol. 15, no. 4, pp. 178–192, 2018. [Online]. Available: <https://doi.org/10.2514/1.1010555>
- [23] D. Keymeulen, S. Shin, J. Riddley, M. Klimesh, A. Kiely, E. Liggett, P. Sullivan, M. Bernas, H. Ghossemi, G. Flesch, M. Cheng, S. Dolinar, D. Dolman, K. Roth, C. Holyoake, K. Crocker, and A. Smith, “High performance space computing with system-on-chip instrument avionics for space-based next generation imaging spectrometers (ngis),” in *2018 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2018, pp. 33–36.
- [24] D. Lüdtkke, T. Firchau, C. G. Cortes, A. Lund, A. M. Nepal, M. M. Elbarrawy, Z. H. Hammadeh, J.-G. Meß, P. Kenny, F. Brömer, M. Mirzaagha, G. Saleip, H. Kirstein, C. Kirchhefer, and A. Gerndt, “Scosa on the way to orbit: Reconfigurable high-performance computing for spacecraft,” in *2023 IEEE Space Computing Conference (SCC)*, 2023, pp. 34–44.
- [25] A. Bonenfant, M. de Michiel, and P. Sainrat, “oRange: A tool for static loop bound analysis,” in *Workshop on Resource Analysis, University of Hertfordshire, Hatfield, UK*, vol. 9, no. 09, 2008, p. 08.
- [26] H. Abaza, Z. A. Haj Hammadeh, and D. Lüdtkke, “DELOOP: Automatic flow facts computation using dynamic symbolic execution,” in *20th International Workshop on Worst-Case Execution Time Analysis (WCET 2022)*, ser. Open Access Series in Informatics (OASICs), C. Ballabriga, Ed., vol. 103. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022, pp. 3:1–3:12. [Online]. Available: <https://drops-dev.dagstuhl.de/entities/document/10.4230/OASICs.WCET.2022.3>
- [27] C. Ballabriga, H. Cassé, C. Rochange, and P. Sainrat, “OTAWA: an open toolbox for adaptive WCET analysis,” in *IFIP International Workshop on Software Technologies for Embedded and Ubiquitous Systems*. Springer, 2010, pp. 35–46.
- [28] T. Franz, A. M. Nepal, Z. A. Haj Hammadeh, O. Maibaum, A. Gerndt, and D. Lüdtkke, “Tasking Modeling Language: A toolset for model-based engineering of data-driven software systems,” in *OBDDP2021 - 2nd European Workshop on On-Board Data Processing*, no. 2, June 2021. [Online]. Available: <https://elib.dlr.de/145077/>

A Novel Heuristic Framework for Offline IMA Schedule Generation for Multicore Platforms

Alexandre Esper^{*†}, Jatin Arora^{§†}, Geoffrey Nelissen[‡], Eduardo Tovar[†]

^{*}Capgemini Engineering, Porto, Portugal [†]CISTER, ISEP, Porto, Portugal [§]VORTEX CoLab, Portugal

[‡]Eindhoven University of Technology, Eindhoven, the Netherlands

I. INTRODUCTION

In recent years, the industry has been confronted with the inevitable trend towards multicore processing platforms, which allows to greatly improve the performance/cost ratio of the system. Concurrently, the industry has shown an increasing interest in developing methods and tools to implement, deploy, validate, and certify independently developed applications of different “criticalities” on the same computing node. Such integrated systems are commonly referred to as mixed-criticality systems (MCS) [5], [30]. In our previous work [11], we provided an industrial view on the notion of mixed-criticality systems and showed that some of the existing works that are built upon the Vestal model [30] have some limitations, e.g. it considers that lower criticality tasks can be suspended in case higher criticality tasks overshoot their execution budget. This can be problematic, as safety-critical systems such as avionics generally require strict *space and time partitioning* among tasks of different criticalities executing on the same platform. In the aeronautical domain, this partitioning approach is referred to as the Integrated Modular Avionics (IMA) concept. ARINC-653 [2] is a standard widely adopted in the avionics industry for the development of IMA systems to enforce strong time and space partitioning. This allows applications of different criticalities (also known as Design Assurance Levels) to be developed and run independently on the same hardware platform. Although the ARINC-653 was originally defined for single-core architectures, it has been extended to multicore computing platforms [1] [16].

Ensuring temporal predictability is one of the most important factors while designing applications for the avionics domain. Consequently, *time-triggered scheduling* (TT) is prevalent in safety-critical systems such as [22], because TT scheduling is more predictable as the schedule is constructed at design time and is enforced at run-time. This allows system designers to determine the precise timing of each event, which is particularly important, for instance, in the design of control systems. Among others, some of the most important goals for solutions implementing TT scheduling of IMA applications are: 1) generation ARINC-653 compliant TT schedule; 2) efficient generation of a TT schedule, i.e., within a reasonable time; 3) generation of a TT schedule that efficiently utilizes the computing platform; 4) generation of a TT schedule that is scalable as per the requirements of modern avionic systems.

Several approaches exist in the literature [7], [8], [18], [22], [24], [31] that focus on TT scheduling of IMA applications. However, these existing approaches are either not compliant with ARINC-653 specifications [31] or are not scalable to large IMA applications and the number of processing cores [7], [8], [24]. This can be problematic as the aeronautics industry is witnessing an unprecedented increase in the complexity of aircraft-embedded computers [12]. Consequently, the traditional aviation development processes are having difficulties keeping up with the development requirements of large-scale complex avionics systems, mainly in terms of cost, time, and reusability [9]. This trend suggests that future avionics systems will require also more sophisticated methods

and tools that will enable handling larger systems with a higher number of cores [20]. For instance, the Boeing 787, which also uses an IMA architecture, hosts over 80 different applications in the core processor cabinet [13]. With such an increase in size and complexity, the importance of efficient, scalable, and effective real-time scheduling solutions becomes even more critical.

To achieve this goal, this paper proposes a *novel heuristic framework* for the next-generation avionics systems that can run a large number of ARINC-653 compliant IMA applications on top of multi/many core platform. The proposed framework can be used to efficiently generate a TT schedule for a large number of ARINC-653 IMA applications running on a large-scale multi/many-core platforms. Furthermore, the proposed framework allows partition instances of the same IMA application to be executed in any core to efficiently utilize the computing platform. The experimental results reveals that the proposed framework can outperform the state-of-the-art [24] by improving the schedulability ratio up to 46% even for the threshold timeout limit, i.e., the maximum time allowed to find a solution, of 4 hours.

The main **contributions** of the proposed heuristic framework are:

1. A novel algorithm for building a graph of the hierarchy of IMA applications partitions instances in an efficient manner based on a set of defined rules, which takes into account the impact of IMA partitions instances with smaller execution demands on IMA partitions instances with larger execution demands.
2. A novel algorithm to build a graph that abstracts the schedule in a hierarchy of smaller schedule intervals (sub-intervals), which is built based on a set of defined rules and that allows us to transform a large NP-Hard problem into a series of smaller problems that are relatively straightforward to solve;
3. A novel multi-core schedulability test that allows us to efficiently search for suitable sub-intervals in the schedule sub-intervals graph to allocate the partition instances of each IMA application; and
4. A novel scheduling strategy that allows to efficiently build the schedule, by scheduling the partition instances assigned to each sub-interval of the schedule sub-intervals graph.

II. SYSTEM MODEL

We consider a multicore platform comprising m identical cores, denoted by $\Pi_1, \Pi_2, \dots, \Pi_m$, with $m \in \mathbb{Z}^+$, where \mathbb{Z}^+ is the set of positive integers. We consider a set of n IMA hosted applications denoted by $\alpha = \{\alpha_1, \alpha_2, \dots, \alpha_n\}$, where each application α_i , $1 \leq i \leq n$, with $n \in \mathbb{Z}^+$. Associated with the n applications we have n processing partitions P_1, \dots, P_n . Each application α_i is associated to a single processing partition P_i . Each P_i is defined by an activation period T_i , an execution budget B_i , a relative deadline D_i and an offset O_i in relation to the start of the period T_i , which means that the partitions are *asynchronous* in relation to the start of the schedule. The value of B_i can be computed using existing methods [29]. We assume that each P_i has

constrained deadline, i.e., $D_i \leq T_i$. We assume that each P_i releases an infinite number of processing partitions instances, which we denote as $P_{i,j}$. Each instance $P_{i,j}$ of a processing partition of an application α_i is released periodically with period T_i until the end of the Major Frame (MAF). The length of the MAF is the *lcm* of all partition periods, i.e., $MAF := lcm_{\alpha_i \in \alpha} \{T_i\}$. The absolute release time (resp. start time) of the j^{th} instance of processing partition P_i is denoted as $r_{i,j}$ (resp. $s_{i,j}$). The absolute deadline of the j^{th} instance of P_i is denoted as $d_{i,j}$. We designate this set of input parameters of each $P_{i,j}$ as $\mathcal{P}_{i,j}$ and the set of all $\mathcal{P}_{i,j}$ as \mathcal{P} .

As we consider an offset O_i associated to each application α_i , we need to ensure that our schedule can be successfully repeated towards infinity. Hence, we extend the MAF definition to an observation window OBW. The OBW is defined as $OBW = O_{max} + MAF$, where $O_{max} = max_{\alpha_i \in \alpha} O_i$. To build our repetitive schedule, we only consider that $\forall P_{i,j}, r_{i,j} < OBW$. Within OBW , we also define the total utilization u_{tot} as $\forall_i : u_{tot} = \sum_1^n B_i^T / (m \times OBW)$, where B_i^T is the sum of the B_i of all α_i partitions instances $P_{i,j}$ within OBW , i.e., with $r_{i,j} < OBW$.

We assume that each $P_{i,j}$ of an α_i can execute on any of the m cores. This property brings an advantage over existing solutions [18], [24] that restrict the $P_{i,j}$ of an α_i to run on the same core, because it allows more efficient utilization of the computing platform. We assume in this work that the IMA processes of an α_i that run inside the $P_{i,j}$ will always resume its execution in the same state after a migration (either on the same or another core). Ensuring that the processor cache is always flushed whenever a migration occurs is one of the methods that can be used to ensure that these migrations can be safely performed. The discussion of other methods that can be used in conjunction with cache flushing to ensure safe migration are left outside the scope of this work.

To ensure deterministic scheduling of the partitions, we define the following **set of assumptions**:

- Partitions are scheduled on a fixed cyclic basis - a Major Frame (MAF) of fixed duration is maintained by the OS scheduler, which is usually defined as a multiple of the least common multiple of all partition periods;
- The partitions are then allocated to one or more execution windows within the MAF;
- Partitions are activated according to the defined offset from the start of the MAF and remain active for the duration of their execution windows;
- The sequence of activation of the partitions are defined during design time using configuration tables;
- The configuration table for the partition schedule contains the order of activation and the length of the execution windows within the MAF;
- A partition periodically releases a potential infinite number of "partition instances";
- The processing partitions can be mapped to any of the available processing cores;
- Mapping of a partition instance between cores is not allowed, but each processing partition instance released by the same application may run on different cores.

The problem of synchronizing access to I/O resources is out of the scope of this paper and is kept for future work.

Although our heuristic framework does not use a CP approach, we do formulate some **constraints** that must be respected by our implementation. First, we need to ensure that the start time of all $P_{i,j}$ of all α_i is not negative (Constraint (1)) and that the deadlines of each α_i are always respected, i.e., the completion time of each $P_{i,j}$ is no later than its corresponding application's absolute deadline (Constraint (2)). Since partitions are executed periodically, the corresponding

$P_{i,j}$ cannot be released before the beginning of each application period (Constraint (3)).

$$\forall P_{i,j} : s_{i,j} \geq 0 \quad (1)$$

$$\forall P_{i,j} : s_{i,j} + B_i \leq D_i \quad (2)$$

$$\forall P_{i,j} : r_{i,j} \geq (j-1) \times T_i \quad (3)$$

Knowing that only one $P_{i,j}$ can execute on a given core m at a time, we must ensure that the $P_{i,j}$ of different partitions allocated to that core do not overlap with each other. This is enforced by defining Constraint (4), where j and l denotes the j^{th} and l^{th} instances of processing partitions P_i and P_k , respectively, with $1 \leq i \leq n$ and $1 \leq k \leq n$.

$$\forall m, \forall P_{i,j}, \forall P_{k,l} | P_{i,j} \neq P_{k,l} : s_{i,j} \geq s_{k,l} + B_{k,l} \quad \vee \quad s_{k,l} \geq s_{i,j} + B_{i,j} \quad (4)$$

The number of constraints defined by (4) can rapidly grow with the number of applications, thus reducing the effectiveness of the CP approach, especially with higher number of cores under high load. Our proposed heuristic framework addresses this problem by efficiently breaking down this NP-hard problem [15] into a series of smaller and simpler problems that are relatively straightforward to solve, as discussed in the next section.

III. PROPOSED HEURISTIC FRAMEWORK

The goal of our heuristic framework is to efficiently generate an offline IMA-compliant schedule, where each $P_{i,j}$ is mapped to any of the m cores and is assigned a start time $s_{i,j}$, such that all the previously defined constraints are respected. Our heuristic consists of a set of deterministic algorithms that run sequentially, i.e., given a defined set of inputs, the heuristic will always produce exactly the same schedule. We use directed acyclic graphs (DAG), and more specifically *augmented trees*, to create hierarchical abstractions of partitions instances and schedule sub-intervals. This heuristic process consists of **four phases**. In **Phase 1**, we construct a graph reflecting the $P_{i,j}$ hierarchy. This graph is used to decide the order in which we schedule each $P_{i,j}$. In **Phase 2**, we build another graph, which abstracts the schedule in a hierarchy of smaller schedule intervals, thus simplifying the scheduling problem. Then in **Phase 3**, we traverse the graph built in Phase 2 and allocate each $P_{i,j}$ to a schedule interval. Finally, in **Phase 4**, we allocate each $P_{i,j}$ to one of the m cores within the allocated intervals and assign a start time $s_{i,j}$ to it. The referred phases are explained in detail in the following sections.

A. Phase 1 - IMA Partitions Hierarchy Graph Construction

This phase takes input a set of parameters of each IMA application α_i , namely \mathcal{P} . Based on these inputs, we create the IMA Partitions Hierarchy Graph, P_{graph} , which is a DAG object with its properties and methods (functions), which is built upon a set of defined rules. P_{graph} is used in our heuristic to determine the order in which we allocate the partitions to schedule sub-intervals. P_{graph} is formally defined as $P_{graph} = (\mathcal{V}_p, \mathcal{E}_p)$, where each vertex $v_p \in \mathcal{V}_p$ is an object representing an IMA partition instance $P_{i,j}$, with $p \in [1..T^P]$, where $T^P = |\mathcal{P}|$. Each v_p also has properties and methods associated to it. We store several important data in each v_p object, including the input parameters from \mathcal{P} . We use the "." operator to access the data stored in v_p . For example, to retrieve the deadline of the $P_{i,j}$ stored in a v_p object, we perform the following operation: $r_{i,j} \leftarrow v_p.deadline$. Note that given a $P_{i,j}$, it is possible to access the respective v_p object from P_{graph} through the following operation: $v_p \leftarrow P_{graph}[p]$. The edges e_p of P_{graph} connect the vertices

v_p in a hierarchical way, according to a set of defined criteria that will be explained in the following sections.

Figure 1a depicts an example schedule with the intuition behind P_{graph} construction process. In this scenario, we consider a set of random partitions $P1$ to $P6$ with their respective releases $r_{i,j}$ and deadlines $d_{i,j}$ in absolute time units. From this simple example, we can intuitively verify that $P1$ is more constrained by $P4$, $P5$, and $P6$, and not so much by $P2$ and $P3$. This principle leads to the idea of creating a graph of the partitions hierarchy, where each partition corresponds to a vertex of the graph, connected by edges in such a way that a hierarchy of those partitions is formed.

Equation 5 defines the condition to determine the hierarchical relationship between partitions. The equation establishes that for a partition $P_{k,l}$ to be considered a child of another partition $P_{i,j}$, two conditions must be met. Firstly, $P_{i,j}$ must encompass $P_{k,l}$, i.e., it must have an earlier release time and a later deadline than $P_{k,l}$. Secondly, there should not exist any other distinct $P_{s,t}$ that simultaneously encompasses $P_{k,l}$ and is encompassed by $P_{i,j}$.

$$\begin{aligned} & \forall P_{k,l}, (\exists P_{i,j}, i \neq k, | r_{i,j} \leq r_{k,l} \wedge d_{k,l} \leq d_{i,j}) \wedge \\ & (\nexists P_{s,t}, s \neq k, s \neq i, | ((r_{i,j} \leq r_{s,t} \wedge d_{s,t} \leq d_{i,j}) \wedge \\ & (r_{s,t} \leq r_{k,l} \wedge d_{k,l} \leq d_{s,t}))) \end{aligned} \quad (5)$$

Note that according to Equation 5, a parent partition $P_{i,j}$, can have only one child $P_{k,l}$ in the interval $[r_{k,l}, d_{k,l}]$, but one child partition $P_{k,l}$ is allowed to have several parents $P_{i,j}$. For example, in Figure 1a, if the deadline of $P2$ were equal to 26, then $P2$ would also be considered as a parent of $P6$, thus both $P2$ and $P3$ would be considered as parents of $P6$.

Next, we describe *Phase 1* sub-phases to create P_{graph} .

Sub-Phase 1.1 - Generation of a list of schedule events and Initialization of partitions hierarchy graph. The purpose of this sub-phase of our heuristic is twofold: the generation of the schedule events data set \mathcal{E} and the initialization of P_{graph} vertices v_p . \mathcal{E} is defined as $\mathcal{E} = \{\mathcal{T}_1, \dots, \mathcal{T}_q\}$, $q \in \mathbb{Z}^+$. Each $\mathcal{T}_x \in \mathcal{E}$, $x \in [1, q]$, is defined as $\mathcal{T}_x = \{t_x, \{E_1, \dots, E_l\}\}$, $l \in \mathbb{Z}^+$, where $0 \leq t_x \leq d_{last}$, and where d_{last} is the latest absolute deadline among all $P_{i,j}$ with $r_{i,j} < OBW$. Each $E_p \in \mathcal{T}$ associated with a $P_{i,j}$ with identifier p is represented as $E_p = \{"release" \vee "deadline", p\}$, with $p \in [1..TP]$. For the sake of simplicity, we designate each E_p as either E_p^r or E_p^d , to represent the release or deadline events of a $P_{i,j}$.

This sub-phase takes \mathcal{P} as input, and performs the following actions:

- initialize P_{graph} vertex with identifier p
- iterate over all $P_{i,j} \in \mathcal{P}$ to:
 - add the vertices objects v_p representing each $P_{i,j}$ to P_{graph} ;
 - add the E_p^r or E_p^d of $P_{i,j}$ to \mathcal{E} ;
- sort \mathcal{E} in ascending order of time stamps t to yield \mathcal{E}^O .

It is important to highlight that at this stage the vertices v_p are not connected by the respective edges.

Sub-Phase 1.2 - Construction of P_{graph} . In this sub-phase, we use function *ConstructPartitionHierarchyGraph()* (Algorithm 1) to build P_{graph} . This function iterates over all $\mathcal{T} \in \mathcal{E}^O$ to determine the hierarchical parents of each $P_{k,l}$ that satisfy the condition given by Equation 5. The analysis of each $\mathcal{T} \in \mathcal{E}^O$ is composed of three parts. In the first part (lines 2 to 6), we iterate over all $E_p \in \mathcal{T}_x$, and determine the set \mathcal{A}_x containing the identifiers b of the $P_{i,j}$ that are active at time $t_x \in \mathcal{T}_x$. A $P_{i,j}$ is considered to be active when $r_{i,j} \leq t_x \leq d_{i,j}$. Note that being active at t_x is a pre-condition for a $P_{i,j}$ to be a parent of $P_{k,l}$, with $i \neq k$. In the second part of the analysis we determine the parent(s) of each $P_{k,l}$

Algorithm 1: Sub-phase 1.2: P_{graph} construction

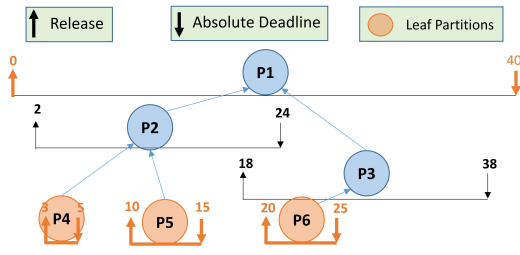
Output: P_{graph} construction completed

```

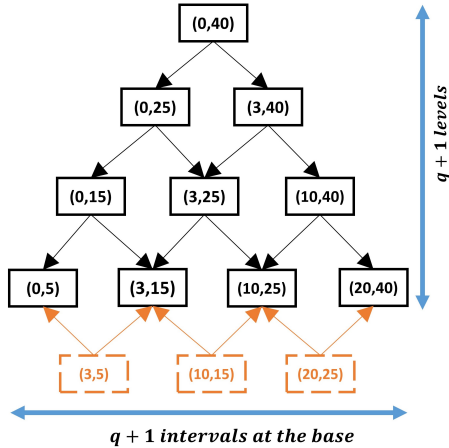
1 Function ConstructPartitionHierarchyGraph( $\mathcal{P}$ ,  $\mathcal{E}^O$ ,
    $P_{graph}$ ):
2    $\mathcal{A}_x \leftarrow \{\}$ 
3   for  $\mathcal{T} \in \mathcal{E}^O$  do
4     for  $E \in \mathcal{T}$  do
5       if Event  $e \in E$  is a "release" then
6         Get  $P_{i,j}$  identifier  $b \in E$  and append to  $\mathcal{A}_x$ 
7        $\mathcal{Q} \leftarrow \{\}$ 
8       for  $E \in \mathcal{T}$  do
9         Get  $P_{k,l}$  identifier  $a \in E$ 
10        if (Event  $e \in E$  is a "deadline")  $\wedge$  ( $|\mathcal{A}_x| > 0$ ) then
11           $\mathcal{A}_y \leftarrow \{\}$ 
12          for  $b \in \mathcal{A}_x$  do
13            if ( $a \neq b$ )  $\wedge$  ( $b \notin \mathcal{Q}$ )  $\wedge$ 
14              ( $v_b.r \leq v_a.r \wedge v_a.d \leq v_b.d$ ) then
15                if ( $v_b.r = v_a.r \wedge v_a.d = v_b.d$ ) then
16                   $P_{graph}.AddParent(v_b, v_a)$ 
17                   $P_{graph}.AddChild(v_a, v_b)$ 
18                   $\mathcal{A}_y \leftarrow \{\}$ 
19                  Append  $a$  to  $\mathcal{Q}$ 
20                  break
21                else
22                  Append  $b$  to  $\mathcal{A}_y$ 
23           $\mathcal{A}_z \leftarrow \mathcal{A}_y$ 
24           $p\_found = FALSE$ 
25          for  $b \in \mathcal{A}_y$  do
26            if  $p\_found = FALSE$  then
27              for  $c \in \mathcal{A}_y$  do
28                if ( $c \neq b$ )  $\wedge$ 
29                  ( $v_b.r \leq v_c.r \wedge v_c.d \leq v_b.d$ ) then
30                  if  $|\mathcal{A}_z| = 1$  then
31                     $p\_found = TRUE$ 
32                    break
33                  else
34                    Remove  $c$  from  $\mathcal{A}_z$ 
35          for  $b \in \mathcal{A}_z$  do
36             $P_{graph}.AddParent(v_b, v_a)$ 
37             $P_{graph}.AddChild(v_a, v_b)$ 
38          for  $E \in \mathcal{T}$  do
39            if Event  $e \in E$  is a "deadline" then
40              Get  $P_{i,j}$  identifier  $b \in E$ 
41              if  $b \in \mathcal{A}_x$  then
42                Remove  $b$  from  $\mathcal{A}_x$ 

```

(lines 7 to 34). We iterate again over all $E \in \mathcal{T}$ (line 8) to determine the $P_{i,j}$ with identifier b , with $b \in E_p$, that can be a parent of each $P_{k,l}$ with identifier a . The auxiliary set \mathcal{Q} initialized in line 7 is simply used to prevent a child vertex from being added as a parent of its parent. In line 9 we get the identifier a of the $P_{k,l}$. If $e \in E$ is a deadline event and \mathcal{A}_x is not empty (line 10), we initialize the set \mathcal{A}_y (line 11) that will store the candidate parents $P_{i,j}$ of $P_{k,l}$. Then in line 12, we iterate over all active $P_{i,j}$ with identifier $b \in \mathcal{A}_x$. The condition in lines 13 and 14 filters the $P_{i,j}$ that are candidate parents. In line 15 we test the special case where a $P_{k,l}$ with identifier a has exactly the same release and deadline as that of the $P_{i,j}$ with identifier k . The operations $v_b.r$, $v_a.r$ and $v_b.d$, $v_a.d$ allow us to retrieve the releases and deadlines stored in the objects v_b and v_a , respectively. If that is the case, we can add it straight away as parent of $P_{k,l}$ (lines 16 and 17), reset \mathcal{A}_y (line 18), and add a to \mathcal{Q} , to prevent $P_{k,l}$ from being added as parent of $P_{i,j}$. We then break the loop (line 20) and proceed to the analysis of the next event E_p . Otherwise, if the



(a) Intuition behind P_{graph} construction.



(b) Intuition behind I_{graph} construction.

Fig. 1: Examples describing the intuition behind the construction process of P_{graph} and I_{graph} .

condition in line 15 is not satisfied, we add b to \mathcal{A}_y (line 22) and proceed to the next stage (lines 23 to 34) to select the parent(s) among the candidates in \mathcal{A}_y . To achieve that goal, we must test all combinations among the $P_{i,j}$ stored in \mathcal{A}_y through the for loops in lines 25 and 27, to check if they are a parent of one another, thus violating the condition given by Equation 5 (lines 28 and 29). But before that, as preparation, we make a copy \mathcal{A}_z of \mathcal{A}_y (line 23) to avoid interfering with the iteration control variable and initialize the flag to detect when a parent has been found (line 24). If the condition in lines 28 and 29 is true, we remove the $P_{i,j}$ with identifier c from \mathcal{A}_z (line 34). We proceed with this iteration until all parents are found or until only one parent is left in \mathcal{A}_z (line 30). Once this analysis is finalized, we simply add the $P_{i,j}$ with identifier $b \in \mathcal{A}_z$ as parent(s) of the $P_{k,l}$ with identifier a (lines 35 to 37). The final part of the analysis (lines 38 to 42), consists simply of deactivating all $P_{i,j}$ with deadline events E_k^d at time t associated with event subset \mathcal{T} . The approach is the same as the one used in lines 3 to 6 but with a deadline event instead of a release.

B. Phase 2 - IMA Partitions Interval Hierarchy Graph Construction

Once P_{graph} has been built, the next phase is the construction of the IMA partitions intervals hierarchy graph, which is also a DAG object with its properties and methods (functions), similar to P_{graph} . We define this DAG as $I_{graph} = (\mathcal{V}_k, \mathcal{E}_k)$. Each vertex $v_k \in \mathcal{V}_k$ is an object representing an IMA schedule sub-interval I_k , defined as a tuple (t_b, t_e) , where $0 \leq t_b < t_e \wedge t_b < t_e \leq OBW$. Given an I_k , it is possible to retrieve the respective v_i object from I_{graph} through the following operation: $v_k \leftarrow I_{graph}[I_k]$. The edges e_k of I_{graph} connect the vertices v_k in a hierarchical way, according to a set of defined criteria. The construction process of I_{graph} is implemented by Algorithm 2.

Phase 2.1 Construction of \mathcal{LP} and \mathcal{LI} . The purpose of this sub-phase is twofold: (i) the creation of the set \mathcal{LP} , which

contains the list of identifiers p associated with the leaf vertex $v_p \in P_{graph}$; (ii) the initialization of I_{graph} leaf vertices. A $v_p \in P_{graph}$ is defined as a leaf iff $v_p.getChild() = \emptyset$, where $getChild()$ is a function of v_p that returns the set of identifiers of the children of v_p . Similarly, a $v_k \in I_{graph}$ is defined as a leaf iff $v_k.getChild() = \emptyset$. We designate each interval of \mathcal{LP} as L_k , with $k \in [0..|\mathcal{LI}| - 1]$. This sub-phase is implemented by function $constructLeafPartitionList()$ (line 4 of Algorithm 2), which performs the the following procedure steps:

- Iterate over each $v_p \in P_{graph}$:
 - If $v_p.getChild = \emptyset$:
 - * Add p to \mathcal{LP} ;
 - * Add the interval $I_k = (v_p.release, v_p.deadline)$ to \mathcal{LI} ;
 - * Create the vertex v_k associated with I_k in I_{graph} ;
- Sort \mathcal{LI} in ascending order of interval start time t .

Algorithm 2: Phase 2.1: I_{graph} construction

Output: I_{graph} construction completed

Function

```

constructIntervalHierarchyGraph( $\mathcal{P}, P_{graph}, \mathcal{E}^O, \mathcal{P}, \mathcal{IB}, \mathcal{LI}$ ):
1
  SCHEDULE_START  $\leftarrow$  0
2
  SCHEDULE_END  $\leftarrow$  max(getEventsTime( $\mathcal{E}^O$ ))
3
  constructLeafPartitionList( $\mathcal{P}, \mathcal{LP}, \mathcal{LI}$ )
4
  constructIntervalGraphBaseList( $\mathcal{LI}, \mathcal{IB}, SCHEDULE\_START, SCHEDULE\_END$ )
5
   $I_{graph} \leftarrow$  createIntervalGraph()
6
  graph_depth  $\leftarrow$  | $\mathcal{IB}$ |
7
  constructRightDiagonalVertices( $\mathcal{IB}, I_{graph}, graph\_depth$ )
8
  constructLeftDiagonalVertices( $\mathcal{IB}, I_{graph}, graph\_depth$ )
9
  addLeafIntervals( $\mathcal{LI}, \mathcal{IB}, I_{graph}$ )
10
  return
11
```

Phase 2.2 Construction of \mathcal{IB} . In this sub-phase we use the set \mathcal{LI} to build another set \mathcal{IB} , which will contain the intervals at the base of I_{graph} . This sub-phase is implemented by function $constructIntervalGraphBaseList()$ (line 5 of Algorithm 2), which performs the following procedure steps:

- Build the first interval as $I_{first} = (SCHEDULE_START, I_0[1])$, where $I_0[1]$ means the upper bound of the first interval $I_0 \in \mathcal{LI}$;
- append I_{first} to \mathcal{IB} and add it to I_{graph} ;
- Iterate over each $I_k \in \mathcal{LI}$, with $k \in [0..|\mathcal{LI}| - 1]$:
 - Build the subsequent intervals, except for the last interval, according to the following rule: $(I_k[0], I_{k+1}[1])$;
 - Append each I_k to \mathcal{IB} and add it to I_{graph} ;
- Build the last interval as $I_{last} = (I_e[0], SCHEDULE_END)$, where $e = |\mathcal{LI}| - 1$;
- Append each I_{last} to \mathcal{IB} and add it to I_{graph} ;

Phase 2.3 Construction of I_{graph} right diagonal vertices. In this sub-phase we create and connect the remaining vertices of I_{graph} , taking the set as the starting point \mathcal{IB} . This sub-phase is implemented by the function $constructRightDiagonalVertices()$ (line 8 of Algorithm 2). Our strategy consists in building the graph diagonally. Formally, we iterate over all intervals $J_k \in \mathcal{IB}$, with $k \in [0..|\mathcal{IB}| - 1]$, compute the new interval I_k as $(J_k[0], J_{k+1}[1])$, add I_k to I_{graph} and connect I_k with I_{k-1} .

Phase 2.4 Construction of I_{graph} left diagonal vertices. Since in the previous sub-phase we have created all vertices of I_{graph} , to finalize the construction process of I_{graph} we just need to connect the remaining vertices of I_{graph} . We follow the same process from the previous sub-phase, but iterating now from the last interval of \mathcal{IB} to the first. This sub-phase is implemented by the function $constructLeftDiagonalVertices()$ (line 9 of Algorithm 2).

We now use Figures 1a and 1b to illustrate the intuition behind the overall construction concept of I_{graph} . According to sub-phase 2.1, we take all the *leaves* of the partitions hierarchy graph P_{graph} (i.e., P_4 , P_5 and P_6) to build \mathcal{LP} . From \mathcal{LP} , we extract the releases and deadlines of P_4 , P_5 and P_6 , and build \mathcal{LI} , i.e., $(3, 5)$, $(10, 15)$, $(20, 25)$ in Figure 1b. Then according to phase 2.2, we take the leaf intervals \mathcal{LI} , augment by instants $t = 0$ (i.e. $SCHEDULE_START$) and $t = OBW$ (i.e., $SCHEDULE_END = 40$), to build \mathcal{IB} , which is given by intervals $(0, 5)$, $(3, 15)$, $(10, 25)$, $(20, 40)$. To build interval $(0, 5)$, we take the beginning of the schedule (i.e., $t = 0$) as the start time of the interval, and the upper bound of the first interval in \mathcal{LI} , i.e. the upper bound of $(3, 5)$. To build the next interval $(3, 15)$, we take the lower bound of the first interval in \mathcal{LI} , and the upper bound of the second interval $(10, 15)$. We continue with this process until the last interval of \mathcal{LI} , i.e., $(20, 25)$. To build last interval $(20, 40)$ of \mathcal{IB} , we take the lower bound of the last interval $(20, 25)$ of \mathcal{LI} and augment it with $SCHEDULE_END$. Then we build the I_{graph} right diagonal vertices based on \mathcal{IB} as per the process defined in sub-phase 2.2. To build the right diagonals of this example graph, we start with interval $(0, 5)$, then we create interval $(0, 15)$ and connect it with $(0, 5)$. Then we proceed with $(0, 25)$ up to $(0, 40)$. Then we follow the same process to build the remaining right diagonals, restarting at $(3, 15)$ up to $(3, 40)$, and so on. To finalize the graph according to phase 2.4, we construct the left diagonal using the same process, which means connecting intervals $(20, 40)$, $(10, 40)$, $(3, 40)$, $(0, 40)$, then $(10, 25)$, $(3, 25)$, $(0, 25)$ and so on, until all vertices are connected.

C. Phase 3 - Allocating Partitions to Schedule Sub-Intervals

Phase 3.1 P_{graph} traversal. The goal of this phase is to decide which $P_{i,j}$ will be assigned next to an interval I_k in I_{graph} or not. Our strategy to traverse P_{graph} consists of a bottom-up approach described in Algorithm 3. To perform the traversal, we use two FIFO queues, which we designate as p_queue and p_wait_queue , and that we initialized in lines 2 and 3. We start the process by iterating over all leaf $P_{i,j}$ whose identifiers p are stored in \mathcal{LP} (line 4), assigning them to their parent interval in the base intervals \mathcal{IB} with the largest lower bound t_b and enqueueing their parents (lines 5 to 8). The interval search process is implemented by function $intervalSearch()$ (line 15 and line 24), which is explained in **Phase 3.2**. In case during the first attempt to find an interval to assign each $P_{i,j}$ we detect that we have two options to choose from in the search path (lines 11 to 21), we opt not to assign $P_{i,j}$ to any of those two intervals yet, and function $intervalSearch()$ returns $TRUE$ (line 15). We then append p to p_wait_queue (line 17). This strategy allows us to reduce the number of decisions that we need to take, by giving all $P_{i,j}$ a first chance to try to find an interval as low as possible towards the base of I_{graph} . We proceed iterating over p_queue (lines 10 to 21) and p_wait_queue (lines 22 to 27) in this order, until both queues are empty, meaning that all $P_{i,j}$ have been assigned to an I_k . Note that the return value in line 24 will never be used, since we don't give a third chance for finding a suitable interval for q .

Phase 3.2 I_{graph} traversal. After selecting the $P_{i,j}$ that must be assigned to an interval in the previous phase, we now initiate or resume the interval downward search (top-down) process to allocate the $P_{i,j}$ to an I_k in I_{graph} . This process is implemented by Algorithm 4. This algorithm implements a set of four criteria that allow us to decide which downward path we will follow, i.e., whether a child interval I_k will be allowed in the i_queue or not. Next, we define each criterion.

Criteria 1. Enqueue child I_k in i_queue iff: $(msst = TRUE) \wedge ((I_k \in \mathcal{LI} \wedge I_k \notin \mathcal{IB}) = FALSE)$. **Rationale:** we

enqueue the child I_k if it passes the $msst$ and if it is not a leaf interval. For example, in Figure 1b, the leaf intervals are $(3, 5)$, $(10, 15)$ and $(20, 25)$.

Criteria 2. Enqueue in i_queue the child I_k with the highest value stored in the set A_{list}^x given by $\max(A_{list}^x)$. The set A_{list}^x is defined in section IV. **Rationale:** we enqueue the child I_k with the highest available CPU processing time to increase the likelihood of being able to run the $P_{i,j}$ in that I_k . Note that in the rest of this paper we use the terms CPU processing time and CPU time interchangeably.

Algorithm 3: Phase 3.1: P_{graph} traversal

Output: All $P_{i,j}$ assigned to an interval in I_{graph}

```

1 Function reversePartitionGraphTraversal( $P_{graph}$ ,
    $\mathcal{LP}$ ):
2    $p\_queue \leftarrow \{\}$ 
3    $p\_wait\_queue \leftarrow \{\}$ 
4   for  $p \in \mathcal{LP}$  do
5      $p\_int =$ 
6       ( $P_{graph}[p].getRelease(), P_{graph}[p].getDeadline()$ )
7     Assign  $p$  to the parent in  $\mathcal{IB}$  with the largest  $t_b$ 
8      $p\_parents = P_{graph}[p].getParent()$ 
9     Add  $p\_parents$  to  $p\_queue$ 
10  while  $|p\_queue| > 0 \wedge |p\_wait\_queue| > 0$  do
11     $temp\_p\_queue \leftarrow p\_queue$ 
12    for  $p \in temp\_p\_queue$  do
13      if  $P_{graph}[p].allChildScheduled() = FALSE$ 
14        then
15          Add  $p$  to the end of the  $p\_queue$ 
16          break
17       $p\_has\_options = intervalSearch(p)$ 
18      if  $p\_has\_options = TRUE$  then
19        Append  $p$  to  $p\_wait\_queue$ 
20      else
21         $p\_parents = P_{graph}[p].getParent()$ 
22        Append  $p\_parents$  to  $p\_queue$ 
23      Remove  $p$  from  $p\_queue$ 
24     $temp\_p\_wait\_queue \leftarrow p\_wait\_queue$ 
25    for  $q \in temp\_p\_wait\_queue$  do
26       $q\_has\_options = intervalSearch(q)$ 
27       $q\_parents = P_{graph}[q].getParent()$ 
28      Append  $q\_parents$  to  $p\_queue$ 
29      Remove  $q$  from  $p\_wait\_queue$ 
30  return
```

Criteria 3. Enqueue in i_queue the child I_k with the highest value of A_{max}^x . If we designate any pair of child vertices as v_l, v_r , then A_{max}^x is computed as follows: $A_{max}^x = \max(\max(A_{list}^x), \max(A_{list}^x))$. A_{max}^x is computed during the construction of I_{graph} for all I_k and stored in each v_k . **Rationale:** The value of A_{max}^x is a metric that allows us to choose a search path towards child intervals with potentially larger values of available CPU time.

Criteria 4. Given two child intervals I_l and I_r , such that $I_l[0] < I_r[0]$, then enqueue $I_r[0]$ in i_queue . **Rationale:** at this point we must force a decision, so we select the child interval with the latest largest lower bound.

Next, we described Algorithm 4 in detail. To perform the interval search, we also use a FIFO queue, which we designate as i_queue . This algorithm takes as inputs the $P_{i,j}$ identifier p and the flag p_from_queue . If this flag is $TRUE$, it means the $P_{i,j}$ comes from p_queue , otherwise, it comes from p_wait_queue .

If it comes from p_queue (line 4), we need first to compute the interval search starting point for $P_{i,j}$ in I_{graph} , which we designate as $I_{top} = (L_{bound}, U_{bound})$. To compute I_{top} lower and upper bounds, we use function $getTopInterval()$ (line 5). This function takes the $P_{i,j}$ release $r_{i,j}$ and deadline $d_{i,j}$, and iterates over \mathcal{IB} to search for the intervals $I_k, I_l \in \mathcal{IB}$,

with $k < l$, whose lower and upper bounds satisfy respectively the following rules:

$$L_{bound} = I_k[0] \mid I_k[0] \leq r_{i,j} < I_{k+1}[0] \quad (6)$$

$$U_{bound} = I_l[1] \mid I_{l-1}[1] < d_{i,j} \leq I_l[1] \quad (7)$$

For instance, in Figure 1, if we apply the above rule to compute the I_{top} of P_2 in Figure 1a, it would yield the interval (0, 25) in Figure 1b.

Algorithm 4: Phase 3.2: I_{graph} traversal

Output: All $P_{i,j}$ assigned to an interval in I_{graph}
Data: \mathcal{E}^O, P_{graph}

```

1 Function intervalSearch( $I_{graph}, p, p\_from\_queue$ ):
2    $p\_bi = I_{graph}[p].getBudget()$ 
3    $i\_queue \leftarrow \{\}$ 
4   if  $p\_from\_queue = TRUE$  then
5      $p\_top\_int = getTopInterval(p)$ 
6      $p\_top\_msst\_result, A^{tot}, A_{list}^x =$ 
7        $performMSST(p, p\_top\_int, p\_bi)$ 
8     if  $p\_top\_msst\_result = TRUE$  then
9        $update\_Ax\_params(p\_top\_int, p, A^{tot}, A_{list}^x)$ 
10      Append  $p\_top\_int$  to  $i\_queue$ 
11    else
12      EXIT - application set not schedulable
13  else
14     $p\_restart\_int = P_{graph}[p].getIntervals$ 
15    Append  $p\_restart\_int$  to  $i\_queue$ 
16  while  $|i\_queue| > 0$  do
17     $next\_i = pop(i\_queue)$ 
18     $i\_child = I_{graph}[next\_i].getSortedChild()$ 
19     $candidate\_ints \leftarrow \{\}$ 
20    if  $|i\_child| > 0$  then
21      for each  $child \in i\_child$  do
22         $p\_msst\_result, A^{tot}, A_{list}^x =$ 
23           $performMSST(next\_p, c, p\_bi)$ 
24        if  $Criteria\_1 = TRUE$  then
25          Append  $each\_child$  to  $candidate\_ints$ 
26    if
27       $(|candidate\_ints| = 2) \wedge (p\_from\_queue = TRUE)$ 
28      then
29         $assignPtoInt(p, next\_i)$ 
30        return TRUE
31    else if  $(|candidate\_ints| = 2) \wedge (p\_from\_queue =$ 
32       $FALSE)$  then
33      if (Evaluation of Criteria 2 was successful) then
34         $update\_Ax\_params(next\_i, p, A^{tot}, A_{list}^x)$ 
35        Append selected  $candidate\_ints$  to  $i\_queue$ 
36      else if (Evaluation of Criteria 3 was successful) then
37         $update\_Ax\_params(next\_i, p, A^{tot}, A_{list}^x)$ 
38        Append selected  $candidate\_ints$  to  $i\_queue$ 
39      else
40        Apply Criteria 4
41        Append selected  $candidate\_ints$  to  $i\_queue$ 
42    else if
43       $(|candidate\_ints| = 0) \wedge (p\_from\_queue = TRUE)$ 
44      then
45         $assignPtoInt(p, next\_i)$ 
46        return FALSE
47    else if  $(|candidate\_ints| = 0) \wedge (p\_from\_queue =$ 
48       $FALSE)$  then
49      if  $next\_i \neq p\_restart\_int$  then
50         $assignPtoInt(p, next\_i)$ 
51      return FALSE
52    else
53       $update\_Ax\_params(next\_i, p, A^{tot}, A_{list}^x)$ 
54      Append selected  $candidate\_ints$  to  $i\_queue$ 

```

Once we have determined the I_{top} of a $P_{i,j}$, before we can assign the $P_{i,j}$ to I_{top} , we must check if the interval I_{top} has sufficient CPU time available to run the candidate $P_{i,j}$. To

perform this check, we have developed a novel schedulability test for TT systems that we designate simply as *msst*. This is implemented by function *performMSST()* (line 6 of Algorithm 4). Due to *msst* complexity, and because it is one of the key contributions of this paper, we opt to describe it in the dedicated section IV. At this point, it suffices to say whether an attempt to assign a $P_{i,j}$ to an interval I_k was successful or not, i.e., whether it has passed the *msst* or not. The *msst* is a function that returns the result of the test (pass or fail) and a set of parameters that are used to determine whether an I_k has sufficient CPU time available in any of the cores to run a certain $P_{i,j}$ or not. Now, we designate these parameters as A^{tot} and A_{list}^x , which are stored in the I_{graph} vertices v_k associated with an I_k , and that help us track the available CPU time in each I_k . These parameters will be explained in detail in section IV. In line 7 we check the *msst* result. If it is a pass, we use function *update_Ax_params()* to update the A^{tot} and A_{list}^x parameters of I_k and to store them in v_k (line 8). We then append the p_top_int to i_queue to continue the search process (line 9). For the case where the *msst* fails when trying to assign a $P_{i,j}$ to its I_{top} (line 10), we halt the process and deem the system as unschedulable by our heuristic.

In case the condition in line 4 yields *FALSE*, it means that the $P_{i,j}$ that comes from the p_wait_queue , so we don't need to recompute its I_{top} , because we simply resume the downward search from the interval where the $P_{i,j}$ first search attempt was halted, when more than one option was detected. Hence, we just retrieve the interval where the search was previously halted (line 13) and append it to i_queue (line 14). We are now ready to initiate the downward search for an interval to assign $P_{i,j}$ by iterating over i_queue (lines 15 to 46). In this part of the algorithm, we take each child interval and apply the four defined Criteria, taking into consideration the number of selected children, and whether $P_{i,j}$ comes from the p_queue queue or not.

D. Phase 4 - Scheduling of Partitions

The last phase of our heuristic is the scheduling of the partitions assigned to each interval of I_{graph} , which means that we assign a start time $s_{i,j}$ to each partition $P_{i,j}$ and a core for it to execute. Algorithm 5 defines our scheduling strategy. It consists in building the schedule backward, by traversing the I_{graph} "diagonally" from right to left and from bottom to top. For example, in the graph in Figure 1b, we would schedule the $P_{i,j}$ assigned to each interval in the following order of intervals: (20,40), (10,40), (3,40), (0,40), then (10,25), (3,25), (0,25), and so on.

Since we build the schedule backward, we start Algorithm 5 by storing the reversed \mathcal{IB} in *reversed_IB* (line 2) and by storing the end of the schedule in *EOS*, which corresponds to the upper bound of the last interval of \mathcal{IB} , given by *reversed_IB*[0][1] (line 3). In line 4 we initialize all m elements of the schedule tracker set \mathcal{S} with the value *EOS*. We define \mathcal{S} as a set of size m , whose purpose is to keep track of the start time $s_{i,j}$ of the last $P_{i,j}$ scheduled in the m cores. Next we iterate over the *reversed_IB* to schedule the $P_{i,j}$ in each interval $I \in I_{graph}$ (lines 5). But before we initiate the scheduling of the $P_{i,j}$ assigned to I , we need to check if any of the schedule tracker values in \mathcal{S} is larger than the upper bound of I , given by $I[1]$. This is performed in the procedure in lines 7 to 10. First, we check if it is not the last interval in the schedule (line 6), and then we iterate over all elements of \mathcal{S} (line 7) and check if the schedule tracker values $\mathcal{S}[i]$ are larger than the interval upper bound $I[1]$ (line 8). If this condition is true, we update the respective $\mathcal{S}[i]$ value with the $I[1]$ (line 9). Next, we initiate the iteration over all intervals of the I_{graph} diagonal to schedule the $P_{i,j} \in I$, starting from

the base intervals $I \in \text{reversed_IB}$ (line 10) until I has no parent interval (i.e., we reach the top of I_{graph}). In each I , we perform the iteration over all the $P_{i,j} \in I$, and schedule each one of them (line 22) in the core that has the largest available CPU time, given by function $\text{max}(\mathcal{S})$ (line 27). If the computed P_{start} occurs before the release time P_r of the $P_{i,j}$ (check at line 31 fails), we deem the application set as unschedulable as per our heuristic (line 32).

Algorithm 5: Phase 4: generateSchedule

```

Output: All  $P_{i,j}$  assigned a start time  $s_{i,j}$  and a core  $m$ 
1 Function generateSchedule ( $P_{\text{graph}}, I_{\text{graph}}, \text{IB}$ ):
2    $\text{reversed\_IB} \leftarrow \text{reverseBaseList}(\text{IB})$ 
3    $\text{EOS} \leftarrow \text{reversed\_IB}[0][1]$ 
4   Let a set  $\mathcal{S}, |\mathcal{S}| = m, \forall w \in \mathcal{S}, w \leftarrow \text{EOS}$ 
5   for  $I \in \text{reversed\_IB}$  do
6     if  $I \neq \text{reversed\_IB}[0]$  then
7       for  $i \in [0..|\mathcal{S}| - 1]$  do
8         if  $\mathcal{S}[i] > I[1]$  then
9            $\mathcal{S}[i] \leftarrow I[1]$ 
10    while  $I$  has a parent interval do
11       $\mathcal{J} \leftarrow I_{\text{graph}}[I].\text{getCandidates}()$ 
12      if  $\mathcal{J} \neq \emptyset$  then
13         $\mathcal{B} \leftarrow \{\}$ 
14         $\mathcal{D} \leftarrow \{\}$ 
15        for  $J \in \mathcal{J}$  do
16           $\mathcal{B}_{\text{temp}} \leftarrow \{P_{\text{graph}}[j].\text{getBudget}(), J\}$ 
17           $\mathcal{D}_{\text{temp}} \leftarrow \{P_{\text{graph}}[j].\text{getDeadline}(), J\}$ 
18          Append  $\mathcal{B}_{\text{temp}}$  to  $\mathcal{B}$ 
19          Append  $\mathcal{D}_{\text{temp}}$  to  $\mathcal{D}$ 
20        Sort  $\mathcal{B}$  in descending order of  $B_i$ 
21        Sort  $\mathcal{D}$  in descending order of deadline
22        for  $d \in [0..|\mathcal{D}| - 1]$  do
23           $P \leftarrow \mathcal{D}[d][1]$ 
24           $P_d \leftarrow \mathcal{D}[d][0]$ 
25           $P_b \leftarrow P_{\text{graph}}[P].\text{getBudget}()$ 
26           $P_r \leftarrow P_{\text{graph}}[P].\text{getRelease}()$ 
27           $\text{latest\_core} \leftarrow (\text{index of the } \text{max}(\mathcal{S}))$ 
28          if  $P_d < \text{max}(\mathcal{S})$  then
29             $\mathcal{S}[\text{latest\_core}] \leftarrow P_d$ 
30           $P_{\text{start}} \leftarrow \mathcal{S}[\text{latest\_core}] - P_b$ 
31          if  $P_{\text{start}} < P_r$  then
32            EXIT - the system is not schedulable by
                 our heuristic
33           $P_{\text{graph}}[P].\text{scheduleP}(P_{\text{start}}, \text{latest\_core})$ 
34           $\mathcal{S}[\text{latest\_core}] \leftarrow P_{\text{start}}$ 
35         $I \leftarrow I_{\text{parent}} \mid I[1] = I_{\text{parent}}[1]$ 
36        break
37      else
38         $I \leftarrow I_{\text{parent}} \mid I[1] = I_{\text{parent}}[1]$ 
39        break
40  return

```

IV. PROPOSED MULTI-CORE SCHEDULABILITY TEST

A. MSST Formalization

Our heuristic consists of traversing I_{graph} in the search for a suitable I_k to assign a $P_{i,j}$ with B_i . For each $I_k \in I_{\text{graph}}$ traversed during the search process, we run the *msst* to check if I_k has sufficient CPU time available in the m cores to execute the $P_{i,j}$. Instead of deciding upfront the allocation of the tasks to $P_{i,j}$ to the m cores, and tracking the available processing time in each core, we propose an efficient approach by reducing this two-dimensional problem to a single-dimensional one. We explore the temporal properties of the system as a whole, instead of tracking the available CPU time in each core individually. Through this approach, we compress the information in such a way that it summarizes a lot of possible solutions, i.e., each time we assign a $P_{i,j}$ to an I_k , we may have one or more possible cores to execute it.

Whenever we assign a $P_{i,j}$ to an interval I_k , we consider that I_k changes its state from s to $s + 1$, with $s \in \mathbb{Z}^+$.

Each state s of interval I_k is characterized by a set of temporal parameters (attributes) stored in each v_k . Whenever a transition occurs from state s to $s + 1$, we use these attributes to decide whether an interval I_k has sufficient CPU time available to run a $P_{i,j}$ or not. Next, we define those attributes. Given an interval $I_k = (t_b, t_e)$ and m cores, the key parameters $A_s^x, \forall x, x \in [1, m], m \in \mathbb{Z}^+$, are defined as the maximum CPU time units that can be simultaneously available in x cores in the interval $[t_b, t_b + A_s^x]$ in a defined state s of I_k . The intuition behind A_s^x , for $x = 3$, is depicted in Figure 2 for an $I_k = (0, 100)$ and $m = 6$. The blue color in each core means available CPU time. $A_s^3 = 60$ means a maximum of 60 time units are available simultaneously in 3 cores. A_s^x is initialized as follows: $\forall I_k, \forall x, A_s^x = (t_e - t_b)$ time units. The next key parameter, A_s^{tot} , is defined as the total available CPU time available in all m cores within I_k at a state s . In Figure 2, A_s^{tot} corresponds to the sum of the total CPU time available in blue color in all cores. A_s^{tot} is initialized as $\forall I_k, A_s^{\text{tot}} = m \times (t_e - t_b)$ time units.

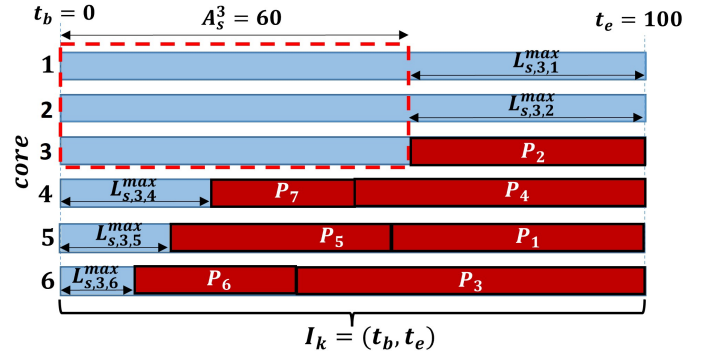


Fig. 2: Intuition behind key *msst* parameters

We now define the parameters $L_{s,x,y}^{\text{max}}, L_{s,x,z}^{\text{max}}$. Given an I_k at state s , $L_{s,x,y}^{\text{max}}$ is defined in Equation 8 as the maximum available CPU time available in core $y, \forall y < x$, with $x, y \in [1, m]$, in addition to A_s^x . $L_{s,x,y}^{\text{max}}$ for $x = 3$ can be visualized in Figure 2, for cores $y = 1$ and $y = 2$. We also define $L_{s,x,y}$, which is an approximation towards $L_{s,x,y}^{\text{max}}$, where $L_{s,x,y} \leq L_{s,x,y}^{\text{max}}$, and that is computed by means of a heuristic described later in this section.

$$\forall x \in [1, m], \forall y < x, L_{s,x,y}^{\text{max}} = A_s^y - A_s^x \quad (8)$$

Similarly, given an I_k at state s , $L_{s,x,z}^{\text{max}}$ is defined as a constraint given by Equation 9.

$$\forall x \in [1, m], \forall z > x, L_{s,x,z}^{\text{max}} < A_s^x \quad (9)$$

We also define $L_{s,x,z}$, which is an approximation towards $L_{s,x,z}^{\text{max}}$, where $L_{s,x,z} \leq L_{s,x,z}^{\text{max}}$, and that is also computed by means of a heuristic described later. Hereinafter, we use the term "L parameters" to generically refer to $L_{s,x,y}$ and $L_{s,x,z}$.

Next, we define $A_{s,x}^{\text{other}}, \forall x \in [1, m]$, which is the total available CPU time in x cores in addition to A_s^x , at state s of interval I_k , and that is computed according to Equation 10. The intuition behind $A_{s,x}^{\text{other}}$, for $x = 3$, can be visualized in Figure 2. In Equation 10, the term $x \times A_s^x$ yields the total maximum CPU time available in all x cores within I_k , which corresponds to the sum of the CPU time available in blue color inside the red dashed lines. If we subtract this value from the total available CPU time in all cores, given by A_s^{tot} , we obtain the value of $A_{s,x}^{\text{other}}$ for a given x .

$$\forall x \in [1, n], A_{s,x}^{\text{other}} = A_s^{\text{tot}} - x \times A_s^x \quad (10)$$

We also define Equation 11 to aid in our computations. By looking at Figure 2, we can intuitively see that this equation holds true.

$$\forall x \in [1, m], \sum_{\forall y < x} L_{s,x,y}^{\text{max}} + \sum_{\forall z > x} L_{s,x,z}^{\text{max}} = A_{s,x}^{\text{other}} \quad (11)$$

As previously referred, since $L_{s,x,y} \leq L_{s,x,y}^{max}$ and $L_{s,x,z} \leq L_{s,x,z}^{max}$, Equations 12 or 13 also hold true.

$$\sum_{\forall y < x} L_{s,x,y} + \sum_{\forall z > x} L_{s,x,z} = A_{s,x}^{other} \quad (12)$$

$$\sum_{\forall y < x} L_{s,x,y} + \sum_{\forall z > x} L_{s,x,z} < A_{s,x}^{other} \quad (13)$$

For the previous definitions to hold, the constraint in Equation 14 must be respected. In Figure 2 it can be easily visualized that if this constraint is violated, the definition of A_s^x becomes invalid.

$$\forall x \in [1, m], 0 \leq A_{s+1}^x \leq A_s^x \quad (14)$$

Algorithm 6: Multicore Schedulability Test Algorithm

```

Output: TRUE  $\vee$  FALSE,  $A_{s+1}^{tot}$ ,  $A_{list}^x$ 
Data:  $A_s^{tot}$ ,  $m$ ,  $A_s^x, x \in [1, m]$ 
1 Function performMSST ( $B_i, I_k$ ):
2    $partition\_schedulable \leftarrow FALSE$ 
3   if  $A_s^{tot} > B_i$  then
4      $x \leftarrow 1$ 
5     if  $A_s^x \geq B_i$  then
6        $partition\_schedulable \leftarrow TRUE$ 
7       while  $x \leq m$  do
8         Compute  $A_{s,x}^{other}$  (Eq.10)
9         if  $A_{s,x}^{other} > 0$  then
10           $y \leftarrow 1$ 
11          while  $y < x$  do
12            Compute  $L_{x,y}^{max}$  (Eq.8)
13             $y \leftarrow y + 1$ 
14           $z \leftarrow x + 1$ 
15          while  $z \leq n$  do
16            Compute  $L_{x,z}$  (Eq.16)
17             $z \leftarrow z + 1$ 
18           $y \leftarrow x - 1$ 
19           $A_{s,tmp}^{other} \leftarrow A_{s,x}^{other}$ 
20          while  $y \geq 1$  do
21            Compute  $L_{x,y}$  (Eq.17)
22             $A_{s,tmp}^{other} \leftarrow A_{s,tmp}^{other} - L_{x,y}$ 
23             $y \leftarrow y - 1$ 
24          if ((Eq. 12 = FALSE)  $\wedge$  (Eq. 13 =
                FALSE))  $\vee$  ((Eq. 12 = TRUE)  $\wedge$ 
                ( $x = 1$ )) then
25            Compute  $A_{s+1}^x$  (Eq.15)
26            break
27          else if (Eq. 12
                = TRUE)  $\wedge$  ( $x \neq 1$ )  $\wedge$  ( $L_{s,x,1} < B_i$ )
                then
28            Compute  $A_{s+1}^x$  (Eq.19)
29          else
30            Compute  $A_{s+1}^x$  (Eq.18)
31          else
32            Compute  $A_{s+1}^x$  (Eq.15)
33            break // exit the while loop
34           $x \leftarrow x + 1$ 
35    $A_{list}^x \leftarrow SortDescending(A_{s+1}^x, \forall x)$ 
36    $A_{s+1}^{tot} \leftarrow A_s^{tot} - B_i$ 
37   return  $partition\_schedulable, A_{s+1}^{tot}, A_{list}^x$ 

```

B. MSST Heuristic Intuition

Whenever we try to allocate a $P_{i,j}$ to an I_k , our strategy consists in preserving as much as possible the CPU time reserved in the A_s^x parameters, giving priority to the cores with lower values of x , by consuming first the CPU time available in the L parameters, according to a defined strategy. By maximizing A_s^x towards state $s+1$, we increase the likelihood of being able to allocate the next candidate $P_{i,j}$ to I_k . We can visualize the effect of this strategy in Figure 2, where core 1 will always have the largest available amount of CPU time

and core 6 will always have the lowest. We show in this figure an example of a possible allocation of the $P_{i,j}$ among the 6 cores in red color. But it is important to emphasize that when executing the *msst*, we are not concerned yet about where the $P_{i,j}$ will be executed. Through this approach, we compress the information in such a way that it summarizes a lot of possible solutions, i.e., each time we assign a $P_{i,j}$ to an I_k , we may have one or more possible cores to execute it. Therefore, by keeping track of the values of the A_s^x and A_s^{tot} parameters, and by updating them accordingly every time we allocate a $P_{i,j}$ to an I_k , it is possible to establish an effective *msst*.

C. MSST Heuristic Implementation

Our solution is implemented by the function *performMSST* in Algorithm 6, which receives as input the execution time B_i of a candidate $P_{i,j}$ and the target interval I_k to be tested. The data parameters used by the algorithm are A_s^{tot} , the set of the A_s^x parameters, and the number of cores m . This function returns a boolean variable indicating whether it was possible to assign the candidate $P_{i,j}$ to I_k or not, as well as the updated values of A_s^{tot} and $A_s^x, \forall x$, for the next state s of I_k . The values of $A_s^x, \forall x$, are stored in the set A_{list}^x . Next we explain the computation steps performed by function *performMSST*.

Step 1. Check that $A_s^{tot} \geq B_i$ (line 3) and then check that $A_{sI,i}^1 \geq B_{sI,i}^{job}$ (lines 4 and 5). If both conditions are satisfied, that means we have at least one core with sufficient available CPU time to execute the $P_{i,j}$, so we can set the boolean variable $partition_schedulable \leftarrow TRUE$ in line 6.

Step 2. Compute $A_{s,x}^{other}$ and the L parameters by iterating over all values of x (line 7). We further divide Step 2 into four sub-steps.

Step 2.1. Compute $A_{s,x}^{other}, \forall x$, and check if $A_{s,x}^{other} > 0$ (line 8 and 9). If $A_{s,x}^{other} = 0$, it means no CPU time is available in addition to A_s^x for core x . In this case, it is not worth continuing with the computation of the L parameters for core x , because they are equal to zero. So here we have no choice but to consume the CPU time available in the A_s^x parameter, which is updated according to Equation 15 (line 32).

$$A_{s+1}^x \leftarrow A_s^x - B_i \quad (15)$$

By updating the value of A_s^x , we know that at least one core with sufficient processing time to execute the $P_{i,j}$ (remember Step 1) exists. Hence, we stop looping over the values of x (line 33) and proceed to Step 6 described later. For the cases when $A_{s,x}^{other} > 0$ (line 9), it implies that we can be able to use the available CPU time in the L parameters, thus maximizing A_s^x for the next state $s+1$. To achieve this, we proceed to **Step 2.2**, where we compute the parameter $L_{s,x,y}^{max}, \forall y < x$, according to Equation 8 (lines 10 to 13). In **Step 2.3**, we compute the value of $L_{s,x,z}, \forall z > x$ (lines 14 to 17) according to Equation 16. Since we do not know how the value of $A_{s,x}^{other}$ is distributed among the L parameters, we make an assumption that $L_{s,x,z}, \forall z > x$, is just below B_i , i.e., $B_i - 1$. Through this assumption, we prioritize the consumption of the CPU time available in the cores with smaller values of x , because their $L_{s,x,y}$ will always contain the largest CPU time reserves. This increases the possibility of scheduling the $P_{i,j}$ with larger B_i values. Later in the heuristic, we evaluate if this assumption was accurate or not. We divide Equation 16 in two parts, to improve the accuracy of the computation, by subtracting in the second part the sum of the previously computed values of $L_{s,x,z}$ from $A_{s,x}^{other}$.

$$\forall x, \forall z > x, L_{s,x,z} = \begin{cases} \min(B_i - 1, A_{s,x}^{other}), & \text{if } z = x + 1 \\ \min(B_i - 1, A_{s,x}^{other} - \sum_{k=x+1}^{z-1} L_{s,x,k}), & \text{if } z > x + 1 \end{cases} \quad (16)$$

By knowing the values of $A_{s,x}^{other}$ and $L_{s,x,z}, \forall z > x$, we proceed to **Step 2.4**, where we compute $L_{s,x,y}, \forall y < x$ (lines 18 to 23). The computation method to try to maximize $L_{s,x,y}$ is defined by Equation 17 (line 21). Here our assumption is that in a worst case, the values of $L_{s,x,y}, \forall y < x$, are equally distributed among the y cores. After the computation of $L_{s,x,y}$, for each y , we update the value of $A_{s,x}^{other}$ (line 22) by subtracting the computed value of $L_{s,x,y}$ in each iteration to improve the accuracy.

$$\forall x, \forall y, L_{s,x,y} = \max \left(\min \left(\frac{A_{s,x}^{other} - \sum_{\forall z > x} L_{s,x,z}}{y}, L_{s,x,y}^{\max} \right), 0 \right) \quad (17)$$

Step 3. Having computed the value of $A_{s,x}^{other}$ and of the L parameters, next we decide how the A_s^x parameters should be updated, so that their value is maximized for $s + 1$. The three methods to update the value of the A_s^x parameters are defined by Equations 15, 18 and 19, and are implemented in lines 24 to 30.

$$A_{s+1}^x \leftarrow A_s^x \quad (18)$$

$$A_{s+1}^x \leftarrow \max(A_s^x - (B_i - L_{s,x,1}); 0) \quad (19)$$

The first part of the condition in line 24 checks if computation of the L parameters was optimistic, i.e., if Equations 12 and 13 are violated. In that case, we don't have a safe bound, so we take a conservative approach and update A_{s+1}^x according to Equation 15. The second part of the condition in line 24 checks the special case for $x = 1$, because $L_{s,x,1} = 0$, which implies that $\sum_{\forall z > x} L_{s,x,z} = A_{s,x}^{other}$, thus violating our assumption in Eq. 16 that $L_{s,x,z}$ is always smaller than $B_i, \forall z > x$. Hence we have no choice but to update A_{s+1}^1 according to Equation 15. If the conditions in line 24 do not hold, we test in line 27 Equation 12 for $x > 1$ and $L_{s,x,1} < B_i$. If this happens, it means that we have CPU time available in the $L_{s,x,1}$ parameter, which will always have the largest CPU time reserve among all $L_{s,x,y}$, but it is not sufficient to completely execute the $P_{i,j}$. So we update A_{s+1}^x according to Equation 19, where we first consume all the CPU time available in the $L_{s,x,1}$ parameter, and the remaining we take it from A_s^x , but always ensuring that the term $A_s^x - (B_i - L_{s,x,1})$ does not lead to a negative value of A_s^x . Finally, if no in lines 24 and 27 hold true, it means that we have sufficient CPU time available in at least one of the $L_{s,x,y}$ parameters, hence we can safely keep the values of A_s^x for the next state $s + 1$, according to Equation 18.

Step 5. After computing the A_{s+1}^x parameters, $\forall x$ at state s , depending on the value of B_i , nothing prevents the case where $A_{s+1}^p - B_i < A_{s+1}^q, \forall p, q \in [1, m]$, with $p < q$. If this case would happen, this would imply that $A_{s+1}^p < A_{s+1}^q$, which would be a violation of the constraint defined by Equation 14. Therefore, to prevent this situation from happening, we define a reordering function that reorders the computed values of $A_{s+1}^x, \forall x$, in descending order, named *SortDescending()* (line 35). This function takes as input all computed values of A_{s+1}^x and outputs the reordered values to be used for state $s + 1$, which are stored in the set $A_{s+1}^{x_{list}}$.

Step 6. In this final step we update the value of A_s^{tot} for the next state, according to Equation 20 (line 36). Once this final update is performed, we return variable *partition_schedulable* equal to *TRUE* or *FALSE* (line 37), and the computed $A_{s+1}^{x_{list}}$.

$$A_{s+1}^{tot} \leftarrow A_s^{tot} - B_i \quad (20)$$

V. EXPERIMENTAL RESULTS

In this section, we discuss the experimental results to evaluate the effectiveness of the proposed work. As explained in [22], due to the NP-hardness of the general periodic scheduling problems, it is a common approach to compare the

performance of heuristic solutions against formal approaches that obtain optimal solutions (e.g., ILP, SMT, or CP). This approach allows us to estimate the quality of the heuristic solution compared to the optimal solution. To the best of our knowledge, we are the first ones to compare with [24].

Experimental Setup: Our framework is implemented in a simulation environment that runs on Ubuntu 16.04 running on an Intel® Core™ i7-6700K CPU 4.2 GHz with 64GB RAM. For the default configuration, we generated synthetic data sets with the following parameters: $m = 16$, 60 IMA applications per application set α , with non-harmonic periods T_i randomly chosen from [10; 20; 30; 50; 60; 90; 100] * 1000. The applications' utilizations (u_i) are randomly chosen between 10% and 50% according to a uniform distribution using the randfixedsum [27] algorithm. The B_i of each α_i was computed as $B_i = T_i \times u_i$. A random offset O_i was assigned to each application in relation to the start of T_i such that $0 \leq O_i \leq T_i$. For each run, the same input data sets were provided to the proposed heuristic framework and the CP approach in [24], using CPLEX optimization studio. We also defined a threshold timeout t_{out} of 4 hours, i.e., the maximum time allowed for both approaches to find a solution. This is a common practice [24] as the solving time for the CP approach can drastically increase with the search space, so, the threshold limits the maximum time to find the solution.

We compare the proposed approach against the existing CP approach [24] by evaluating the schedulability ratio, i.e., the percentage of application sets deemed schedulable, average solving time, i.e., the average time required to find a valid schedule, and varying the number of cores. In all the experiments, our approach is marked as "OUR" and the existing approach of [24] is marked as "CP". In all the experiments, the x-axis represents the total application set utilization u_{tot} .

1. Schedulability Ratio: In this experiment, we vary the total application set utilization u_{tot} in the range [50%, 100%] with a step size of 5% and evaluated the schedulability ratio using the proposed framework and existing CP-based approach [24] as plotted in Figure 3a. We can see in Figure 3a that the schedulability ratio using both approaches reduces with the increase in u_{tot} . This happens because the increase in u_{tot} results in an increase of the u_i of each α_i , which increases B_i as $B_i = T_i \times u_i$. Consequently, there is an increase in the system workload, which degrades the schedulability ratio. However, we can see in Figure 3a that the proposed approach was able to schedule up to 46% more applications compared to the CP approach [24]. This gain is mainly observed because, for most of the runs, the CP solution could not find a solution within $t_{out} = 4$ hours for the default configuration, i.e., $m = 16$ with 60 IMA applications, whereas the solving time for the proposed approach was mostly within the t_{out} limit. We observed that the gain of the proposed approach over the CP approach increases significantly for a higher number of cores, e.g, $m = 32$, but we have not reported it in the paper due to space constraints.

2. Average Solving Time: In this experiment, we evaluate the average solving time in relation to the u_{tot} considering default configuration. For this, we varied u_{tot} in the range [50%, 100%] with a step size of 5% and plotted the resulting average solving time using the proposed approach and CP approach in Figure 3b. For this experiment, to plot each point, we only consider the cases in which both approaches were able to find a valid schedule, e.g., if out of 100 runs, an approach can find a valid for only 30 runs, we take the average solving time for those 30 runs. This is the reason that the average solving time for both approaches does not significantly increase with the increase in the u_{tot} value as shown in Figure 3b, since the number of successful runs diminishes as the workload

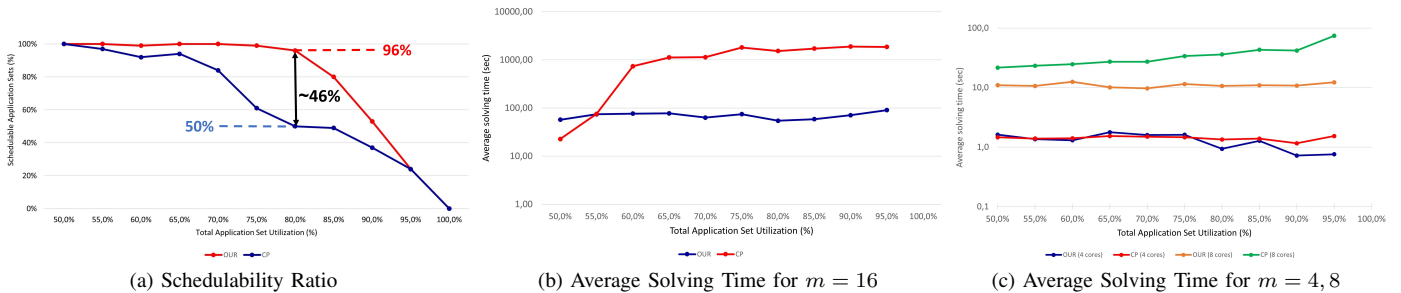


Fig. 3: Experimental Results

increases. Nonetheless, it presents a big picture of the average time consumed by both approaches to find a valid schedule. We can see in Figure 3b that for the successful cases, the proposed approach was generally able to find a solution 10x faster than the existing CP solution.

3. Number of Cores and IMA applications: In this experiment, we redo experiment 2 by varying the value of m and IMA partitions as $m = 4$ with 20 IMAs, $t_{out} = 10$ minutes; and $m = 8$ with 40 IMAs, $t_{out} = 10$ minutes. The average solving time for successful runs using both approaches is plotted in Figure 3c. We can observe in Figure 3c that the difference between the proposed approach and the CP approach is marginal because the existing CP approach can efficiently find a solution for a smaller search space.

VI. RELATED WORK

The multiprocessor scheduling can be broadly categorized into event-driven scheduling, i.e., scheduling decisions are made at run-time based on different parameters, and TT scheduling, i.e., a system-level schedule is constructed at design time which is then enforced at run time, (see surveys [21], [22]). The TT scheduling is proven to be more predictable, as the schedule is constructed at design time so the precise information of each event that will take place at run time is known at the design time. As a consequence, TT scheduling is preferred for designing safety-critical systems (e.g. avionics control systems) due to higher predictability. It also simplifies the process of design, verification, and (re-)certification. Furthermore, designing such systems requires strict *space and time partitioning* among applications of different criticalities executing on the same platform, as mandated by the ARINC-653 standard. Such partitioning ensures sufficient isolation between applications of different criticalities (possibly running on different cores), in such a way that they can be modified/upgraded independently, thus minimizing the system re-certification costs. Considering this, a plethora of works in the literature [6]–[8], [14], [17]–[19], [22], [24], [28], [31] focus on building solutions to generate TT schedule for tasks/IMA applications on multicore platform.

Xu et al. [31] presented a scheduling algorithm based on a branch and bound heuristic to find a feasible non-preemptive schedule on M identical processors. However, in contrast to the proposed work, the work in [31] does not comply with the specifications of ARINC-653, which requires a static allocation of IMA partitions to cores. Deroche et al. [7] proposed an exhaustive branch-and-bound heuristic based approach to build a TT schedule for IMA applications running on multicore platform. Even though their solution is important, it suffers from the problem of scalability as their approach does not scale well for systems with a large number of avionics functions distributed in a limited number of processors. To overcome this challenge, in their subsequent work, Deroche et al. [8] propose an improvement by eliminating the backtracking during the decision tree search process, using a greedy heuristic. To achieve this goal they choose the most

promising valid MAF set in each node of the search tree, based on a metric that takes into consideration the margin of a communication chain, which is the difference between the chain end-to-end delay constraint and the current delay of each chain. The authors performed a comparison with the exhaustive (optimal) approach from [7] and showed improvement in terms of solving time. Although their approach is efficient, it is limited to IMA partitions with synchronous harmonic periods, which is a much less complex problem than the one we are trying to solve (i.e., non-harmonic asynchronous case). Furthermore, contrary to our approach, the solution from [8] does not consider the migration of tasks among cores.

Other existing approaches use Constraint Programming (CP) or Integer Linear Programming (ILP) to build a TT schedule of IMA partitions [4], [10], [23]–[26]. Among all these approaches, the solution in [24] is the closest to the proposed work in terms of contribution and assumptions. Puffitsch et al. [24] presents a CP approach for the generation of TT schedule of real-time dependent periodic non-preemptive asynchronous task sets on multi/many-core platforms for IMA systems. Their solution considers a) precedence constraints between partitions of different IMA applications; b) spatial mapping of IMA application to cores; and c), mapping of communication buffers in the message passing area. Even though the existing CP-based approach [24] can efficiently find a solution for a relatively smaller problem, it does not scale well with the increase in the search space. If a valid TT schedule is not found by an approach within a reasonable time, it will directly impact the schedulability ratio as the taskset will be deemed unschedulable if the schedule cannot be found within a reasonable time. As reported in Section V (see Figure 3a), even with 4 hours of threshold limit, the proposed approach outperformed approach [24] by improving the schedulability ratio up to 46%. Furthermore, the work in [24] uses commercial constraint solver [3] which is limited to internal undisclosed search algorithms. Our tool on the other hand offers many possibilities for future improvements, such as the adoption of different strategies to traverse P_{graph} , I_{graph} , and scheduling strategies.

VII. CONCLUSION AND FUTURE WORK

In this paper, we proposed a novel heuristic framework for configuring and generating IMA-compliant schedules which is efficient in terms of finding a valid schedule, scalable to a large number of IMA applications, and efficiently utilizes the computing platform. The experimental results reveal that our solution can outperform the state-of-the-art CP solution by [24] in terms of solving time, memory usage and does not perform significantly poorly compared to the CP-based optimal solution. In the future, we plan to extend our framework to a) include precedence relations between partitions; and b) consider synchronization of access to system I/O resources by mapping the I/O partitions to a dedicated I/O core.

REFERENCES

- [1] *Avionics Application Software Standard Interface (Part 1): Required Services, ARINC Specification ARINC653P1-4*, 2015. SAE International, 2015.
- [2] *Avionics Application Software Standard Interface (Part 0): Overview of ARINC 653, ARINC Specification ARINC653P0-3*. SAE International, 2021.
- [3] Ibm ilog cplex optimization studio. https://www.ibm.com/products/ilog-cplex-optimization-studio?mhsrc=ibmsearch_a&mhq=ilog, 2023. Accessed: 2023-11-24.
- [4] Matthias Becker, Dakshina Dasari, Borislav Nolic, Benny Akesson, Vincent Nélis, and Thomas Nolte. Contention-free execution of automotive applications on a clustered many-core platform. In *2016 28th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 14–24, 2016.
- [5] Alan Burns and Robert Davis. Mixed criticality systems-a review. *Department of Computer Science, University of York, Tech. Rep*, pages 1–69, 2013.
- [6] Jinchao Chen, Chenglie Du, Fei Xie, and Zhenkun Yang. Schedulability analysis of non-preemptive strictly periodic tasks in multi-core real-time systems. *Real-Time Syst.*, 52(3):239–271, may 2016.
- [7] Emilie Deroche, Jean-Luc Scharbarg, and Christian Fraboul. Mapping real-time communicating tasks on a distributed ima architecture. In *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–8. IEEE, 2016.
- [8] Emilie Deroche, Jean-Luc Scharbarg, and Christian Fraboul. A greedy heuristic for distributing hard real-time applications on an ima architecture. In *2017 12th IEEE International Symposium on Industrial Embedded Systems (SIES)*, pages 1–8. IEEE, 2017.
- [9] Xiaoyan Du, Chenglie Du, Jinchao Chen, Mei Yang, and Wenquan Yu. A simulation and verification platform for avionics systems based on future airborne capability environment architecture. *Applied Sciences*, 12(22):11533, 2022.
- [10] Friedrich Eisenbrand, Karthikeyan Kesavan, Raju S. Mattikalli, Martin Niemeier, Arnold W. Nordsieck, Martin Skutella, José Verschae, and Andreas Wiese. Solving an avionics real-time scheduling problem by advanced ip-methods. In Mark de Berg and Ulrich Meyer, editors, *Algorithms – ESA 2010*, pages 11–22, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.
- [11] Alexandre Esper et al. An industrial view on the common academic understanding of mixed-criticality systems. *Real-Time Systems*, 54, 2018.
- [12] D. Aubrey et al. The future of avionics: High performance, machine-learned and certified. White paper, October 2023. Available at: <https://www.intel.com/content/www/us/en/content-details/791291/the-future-of-avionics-high-performance-machine-learned-and-certified.html>.
- [13] Thomas Gaska, Chris Watkins, and Yu Chen. Integrated modular avionics past, present, and future (vol 30, pg 12, 2015). *IEEE Aerospace and Electronic Systems Magazine*, 30(11):11–11, 2015.
- [14] Jia Huang, Jan Olaf Blech, Andreas Raabe, Christian Buckl, and Alois Knoll. Static scheduling of a time-triggered network-on-chip based on smt solving. In *2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 509–514, 2012.
- [15] Kevin Jeffay et al. On non-preemptive scheduling of periodic and sporadic tasks. In *IEEE real-time systems symposium*, pages 129–139, 1991.
- [16] Hyun-Chul Jo, Joo-Kwang Park, Hyun-Wook Jin, Hyung-Sik Yoon, and Sang Hun Lee. Portable and configurable implementation of arinc-653 temporal partitioning for small civilian uavs. *IEEE Access*, 7:142478–142487, 2019.
- [17] Omar Kermia. An efficient approach for the multiprocessor non-preemptive strictly periodic task scheduling problem. *Journal of Systems Architecture*, 79:31–44, 2017.
- [18] Jung-Eun Kim et al. Integrated modular avionics (ima) partition scheduling with conflict-free i/o for multicore avionics systems. In *38th Annual Computer Software and Applications Conference*, pages 321–331, 2014.
- [19] H. Kopetz. Time-triggered real-time computing. *Annual Reviews in Control*, 27(1):3–13, 2003.
- [20] Georgios Kornaros. *Multi-Core Embedded Systems*. CRC Press, 2018.
- [21] C. Maiza et al. A Survey of Timing Verification Techniques for Multi-Core Real-Time Systems. *ACM Computing Surveys*, 52(3):1–38, June 2019.
- [22] Anna Minaeva and Zdeněk Hanzálek. Survey on periodic scheduling for time-triggered hard real-time systems. *ACM CSUR*, 54(1):1–32, 2021.
- [23] Clément Pira and Christian Artigues. Line search method for solving a non-preemptive strictly periodic scheduling problem. *Journal of Scheduling*, 19, 07 2014.
- [24] Wolfgang Puffitsch et al. Off-line mapping of multi-rate dependent task sets to many-core platforms. *Real-Time Systems*, 51:526–565, 2015.
- [25] Eike Schweissguth, Peter Danielis, Dirk Timmermann, Helge Parzyjegl, and Gero Mühl. Ilp-based joint routing and scheduling for time-triggered networks. In *Proceedings of the 25th International Conference on Real-Time Networks and Systems, RTNS '17*, page 8–17, New York, NY, USA, 2017. Association for Computing Machinery.
- [26] Ahmad Sheikh, O. Brun, Pierre-Emmanuel Hladik, and Balakrishna Prabhu. Strictly periodic scheduling in ima-based architectures. *Real-Time Systems*, 48, 07 2012.
- [27] Roger Stafford. Random vectors with fixed sum. <https://www.mathworks.com/matlabcentral/fileexchange/9700-random-vectors-with-fixed-sum>, 2023. Accessed: 2023-11-19.
- [28] Yasin Unlu and Scott J Mason. Evaluation of mixed integer programming formulations for non-preemptive parallel machine scheduling problems. *Computers & Industrial Engineering*, 58(4):785–800, 2010.
- [29] Steven H VanderLeest and Samuel R Thompson. Measuring the impact of interference channels on multicore avionics. In *2020 AIAA/IEEE 39th Digital Avionics Systems Conference (DASC)*, pages 1–8. IEEE, 2020.
- [30] Steve Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *28th IEEE international real-time systems symposium (RTSS 2007)*, pages 239–243. IEEE, 2007.
- [31] Jia Xu. Multiprocessor scheduling of processes with release times, deadlines, precedence, and exclusion relations. *IEEE Transactions on software engineering*, 19(2):139–154, 1993.

Session Tu.1.C

Network

Tuesday 11th June

11:30

–

Pastel Room

Specifying network switches using the P4 language: lessons learned

Marc Boyer
DTIS / Université de Toulouse
ONERA
Toulouse, France
ORCID: 0000-0003-0344-6991

Julien Brunel
DTIS / Université de Toulouse
ONERA
Toulouse, France
ORCID: 0009-0004-3639-6681

Piere-Julien Chaine
Airbus Operations
Toulouse, France
pierre-julien.chaine@airbus.com

Valentin Kretschmar
Airbus Operations
Toulouse, France
valentin.kretschmar@airbus.com

Pierre Roux
DTIS / Université de Toulouse
ONERA
Toulouse, France
ORCID: 0000-0003-2910-4738

Abstract—The P4 language has been defined to simplify the definition of the behavior of network devices. The idea is to define their behavior with a P4 program instead of a specific hardware, which is promising in terms of flexibility, simplicity, cost reduction. In this article, we investigate on the use of P4 for the specification of embedded network devices and study possible advantages compared to natural language specifications, with a focus on correctness assurance. In particular, we consider the formal verification of properties for a P4 program. Due to the absence of mature verification frameworks for P4 programs, we experiment by generating C code and applying the Frama-C verification framework over the generated code.

Index Terms—network device, formal verification, P4

I. MOTIVATION

Developing embedded network switches requires some exchanges between the equipment provider and its client, such as an aircraft manufacturer. Currently, this is done through requirements that are written in natural language (e.g., in English). The provider then prototypes and manufactures the hardware based on those requirements. In this paper, we will investigate a way to formalize, at least part of, those requirements. This would enable the client to perform some kind of consistency checks on the requirements as well as refinements from high level expectations to more precise requirements. The formalization could also help the provider to better satisfy its client needs. Finally, this could enable both parties to better test the resulting product: indeed, the P4 code can be used to generate test cases satisfying some coverage criteria. Then, the client can use execute the tests on a P4 simulation framework. The results can be considered as the expected outputs for the tests to be executed on the real switch by the provider.

In the remainder of the article, we clarify the objectives of using P4 in this context. We then present the P4 language through a simple running example. We review existing works on P4 formal validation and show a first feedback on practical formal verification of P4. We also stretch the benefit of P4 in terms of test generation.

II. OBJECTIVES

Our goal is to investigate the usability of the P4 language to specify embedded network switches. More precisely, we hope to be able to enjoy the following benefits from formalizing a specification in the P4 language:

- Using the P4 compiler¹ and network simulators, the P4 program makes it easy to experiment and play with the specification, which could help making the requirements more precise.
- Ideally some high level requirements could be statically verified on the P4 program, further strengthening our confidence in the specification.
- The program could be used to automatically generate test cases [19] that ensure a good coverage of the P4 program. These tests could then be used to assert the compliance of the final product with the specification.
- The P4 program could constitute a reference implementation acting as interface with the equipment provider which manufactures the switch. It can help this provider when implementing the hardware.

III. THE P4 LANGUAGE

A. General presentation

The Programming Protocol-independent Packet Processors (P4) language [18] is an open source, domain-specific programming language for specifying how network devices process packets. The language originated in the software-defined network (SDN) world. The fundamental idea underlying the language development was to replace specific hardware by P4 programs describing the switch behavior that could be compiled and executed on different possible hardware devices (FPGA, programmable ASICs). The hardware device is called *target* is P4 terminology.

Figure 1 (from the P4 language specification [18]) shows a typical workflow when programming a target using P4. The

¹<https://github.com/p4lang/p4c>

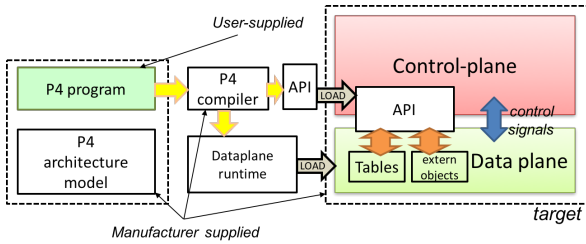


Fig. 1. P4 workflow

P4 program is written for a specific architecture model which relies on hardware capabilities of the target. The architecture model and the P4 compiler are provided by the target manufacturer (but it is expected that some architecture model will be shared by different manufacturers [18, § 4.1]). The result of the compilation implements the forwarding logic described in the P4 program. The right-hand side of the figure represents the P4-programmed device, called *target*. Two crucial functionalities are represented: the control-plane, which describes how the packets should be forwarded (the creation of routing tables is part of control plane), and the data plane, which is in charge of forwarding the packets (complying with the routing tables). A P4 program describes the data plane and its interface with the control plane. The latter can be specified by hand or generated but is out of the scope of a P4 program.

In practice, a P4 program is composed of two kinds of blocks, which are executed in sequence for each incoming packet:

- `parser` blocks describe a finite state machine reading (headers of) the packet, filling some metadata fields attached to the packet and eventually accepting or rejecting the packet;
- `control` blocks written in an imperative language akin to C can perform various treatments based on the above metadata and read some kind of routing table, to eventually decide how to route the packet.

The fact that the language is based on finite state machines and imperative programming, both well studied in the formal method field [16], [20], seems to make it an ideal target for formal verification. The finite nature of the manipulated data makes it even more appealing.

B. Illustration: basic forwarding

Let us illustrate the language with a very simple switch consisting of two blocks. It is extracted from the specification of a switch implementing basic forwarding (from the P4 official tutorial²).

Each block has parameters, which can basically represent metadata associated with the packet. Their types can be either user-defined or predefined in the architecture model or in a P4 library. Besides, they can be tagged `in` (read-only), `out` (uninitialized) or `inout`.

²<https://github.com/p4lang/tutorials>

The P4 file `basic.p4` for our example starts with the inclusion of a P4 library (`core.p4`) and the architecture model file (`ebpf_model.p4`) and the definition of data types. Here is an extract.

```
#include <core.p4>
#include <ebpf_model.p4>
header ethernet_t {
    macAddr_t dstAddr;
    macAddr_t srcAddr;
    bit<16> etherType;
}
header ipv4_t {
    ...
}
struct headers {
    ethernet_t ethernet;
    ipv4_t ipv4;
}
```

The first block is the parser and is described below. The overall behavior is the following: `ethernet` headers are first extracted, then if the just extracted `etherType` field of the packet header is equal to `TYPE_IPV4`, the `ipv4` headers are extracted. In both cases, the packet is eventually accepted. The P4 code is as follows

```
parser MyParser(packet_in packet, out headers hdr) {
    state start {
        transition parse_ethernet;
    }
    state parse_ethernet {
        packet.extract(hdr.ethernet);
        transition select(hdr.ethernet.etherType) {
            TYPE_IPV4: parse_ipv4;
            default: accept;
        }
    }
    state parse_ipv4 {
        packet.extract(hdr.ipv4);
        transition accept;
    }
}
```

The parser is defined as a finite state automaton with five states: the implicit `accept` and `reject` and the user given `start` (the initial state), `parse_ethernet`, and `parse_ipv4`. In each state a sequence of basic statements to compute can be specified. A basic statement is either the manipulation of a variable or the call to a predefined method (e.g., `packet.extract(hdr.ethernet)` in the state `parse_ethernet`). Notice that the function `extract`, which reads the header of the packet, is defined in a library (`core.p4`). Then, with the keyword `transition`, the transition to the next state is specified. A switch/case-like construct (keyword `select`) allows the programmer to choose the next state depending on a boolean condition.

Let us now consider the second block: the ingress block.

```
control MyIngress(inout headers hdr, out bool pass) {
    action drop() {
        pass = false;
    }
    apply {
        if (hdr.ipv4.isValid()) {
            pass = true;
        }
        else drop();
    }
}
```

The block starts with the definition of the unique action in this example (`drop`). Actions are a kind of function local to the

block. The behavior of the control blocks is defined by the `apply` field. Here, if the `ipv4` field of the packet header is valid, then the `pass` output parameter is set to true.

The architecture model we are using (called `ebpfFilter`) requires two blocks. We specify below that we are using the above defined parser and ingress blocks as the two required blocks from the architecture model.

```
ebpfFilter (
    MyParser (),
    MyIngress ()
) main;
```

In order to illustrate the proposed approach, let us consider a simple property that we want to ensure. This property states that if the input packet is long enough, to include both an ethernet (112 bits) and an IPv4 header (160 bits), then the parsing succeeds.

Property. *For any input packet p , if $p.length > 112+160$ then p is forwarded.*

The proof of this property will be demonstrated in section V.

IV. RELATED WORK

The authors were pleasantly surprised to discover that verification of P4 programs is a pretty active and diverse research subject.

It is worth noting that P4 is, as seen above, a pretty specific language with no loop construct, be it `while` loops or recursive functions. That is, P4 does not have the full expressiveness of general purpose programming languages: in other words, the language is not Turing complete. In fact, with mild assumptions, the language has equivalent computing power to finite automatas. This limited computing power makes the language a very attractive candidate for formal verification.

The main works to be found in the literature can be organized in the following way:

Formal semantics and proof assistants

targets the formalization of the semantics of the P4 language in proof assistants, such as Coq or Isabelle/HOL, or logical frameworks, such as the dreaded K. This enables reasoning on the language and manually proving correctness of small programs.

Deductive verification

back propagates a property to verify through the P4 program under study (computation of weakest precondition) leading to a first-order logic formula that is then automatically checked using an SMT solver.

Typechecking

techniques can be used to statically guarantee the absence of some classes of runtime errors (for instance dereferencing invalid pointers).

Runtime verification

consists in instrumenting the program with tests that are run along it during its execution [10]. This is less interesting than the other, static, verification methods in a critical context, except maybe for offline testing.

Test generation

contrary to previous methods doesn't target verification of P4 programs seen as white box implementations, but takes P4 programs as specification to check hardwares seen as black box implementations. Test generation methods can be used to attempt generating sets of tests that would guarantee coverage of each line of the considered P4 program.

A. Formal Semantics and use of Proof Assistants

The goal of Petr4 [3] is to define a formal semantics based on the P4 official Language Specification [18] and independent from the reference implementation. The first objective is to uncover ambiguities and inconsistencies in the Language Specification and bugs in the reference implementation. A formal operational semantics is defined for a fragment of P4. This fragment excludes parser blocks. However, the authors argue that it is possible to encode parsers with unrolled loops (to eliminate recursion), using a function for each state. An effort is made to be parameterized by the architecture model and target. Type soundness and termination are proved for the operational semantics. Moreover, an interpreter has been implemented in OCaml. It is not formally related to the operational semantics and handles the full P4 language. The interpreter is a functor that can be instantiated with different architecture models. It has been validated against a set of tests coming from the reference implementation. No formal verification framework based on Petr4 has been proposed yet.

Nano-p4 [1] is a formalization of P4 using the proof assistant Isabelle/HOL. It seems to be the first formalization of P4 using a proof assistant (P4K has some anteriority but the K framework it's based on isn't an actual generic proof assistant). The authors provide a small step semantics of P4 actions as well as a semantics of P4 parsers. They use this semantics to prove properties like absence of access to uninitialized variables or out of bound accesses in header stack, as well as reachability in the parser automata. They also prove the correctness (i.e., semantics preservation) of an optimization routine for P4 actions, namely constant folding. The Isabelle/HOL code is made of 4000 lines of code³, 200 lemmas and theorems and is available at: <https://github.com/Johanmyst/Nano-P4>. Note however that the author states "freely available" but there seems to be no precise license. The code is apparently no longer developed since the initial master thesis (that lasted six months in 2020).

Leapfrog [4] is a tool to check equivalence of P4 parsers. This allows for instance to check that a simple implementation of a parser (considered as specification) behaves exactly (i.e., accepting and rejecting the exact same packets) like a more complex, optimized, one (considered as the implementation). The tool is developed within the proof assistant Coq offering some strong guarantees with respect to its correctness. However, it calls external SMT solvers without rechecking their results with Coq (i.e., it adds a Coq axiom for each solver

³Proof assistants being pretty verbose, this is relatively small.

call). Tools to check SMT solver results in Coq do exist [9] but the authors were apparently not able to use them.

In P4K [13], an executable formal semantics of P4 is given in the K framework. K is a programming language semantics engineering framework based on term rewriting. The idea is to take benefit from the existing language-independent tools offered by K such as a parser, an interpreter and a symbolic model checker. In the article, the authors provide an informal explanation of the definition of the P4 semantics. Then they present some examples of properties it is possible to check with P4K. The framework offers a good expressiveness but requires a good level of expertise in rewriting and in particular in K. The following property, which is expressible in P4K, illustrates the high expressiveness of the framework: for any input stream of packets, after processing all the packets, no packet is dropped and no new packet is added; all the packets are either sent to port 0 or to port 1, and the difference between the nb of packets sent to 0 and 1 is 0 or 1.

B. Deductive methods

The tool p4v [15] performs formal verification of P4 programs. More precisely, it verifies that a given P4 control block satisfies a given property. If assumptions about the control plane are needed to prove the desired property, then the user must explicitly add these assumptions as annotations in the P4 program. The property to check is added as an annotation in the P4 program. Technically, the P4 program is translated into a first-order logic formula. The fact a violation of the property to be checked is accessible from an initial state is also translated into a formula. Verification conditions based on weakest pre-conditions are then generated and verified with the Z3 SMT solver. A specific effort is made to illustrate and classify some properties that can be verified. The authors distinguish:

- basic safety properties, such as: headers are valid, header stacks are accessed within statically declared bounds, arithmetic operations do not overflow;
- architectural properties, such as: any packet not blocked by a table is rewritten by another table;
- program-specific properties, such as: an internal server is isolated from the rest of the network.

C. Typechecking

Typechecking can be seen as a simple form of abstract interpretation. The use of types in programming languages started to indicate to the compiler the memory size of the program variables. For instance, in C a `char` should use 8 bits of memory whereas a `double` usually spans 64 bits⁴. But since the end of the 20th century, stronger type checking is used by many programming languages to make their compiler statically (i.e., at compile time) enforce much stronger guarantees on

⁴Although, strictly speaking, this second size is not defined in the language specification, we are not aware of a C compiler making an alternative implementation choice.

the programs they accept. For instance, an OCaml program is guaranteed not to segfault by the compiler.⁵

Safe-P4 [7] is an extension of P4 with some additional typechecking to ensure that only packet fields that are valid are accessed. The property is basically guaranteed by the fact that each field access is guarded by some check of its validity (using some `if ... then ... else` construct). It was tested on 15 codes in P4₁₄ (that were more common on Github than current P4₁₆ programs, at the time the paper was written) and found 58 bugs. The bugs were relatively easy to fix and SafeP4 enabled to guarantee the correctness of the fixes. The original prototype used in the paper doesn't seem to be available but an OCaml prototype for P4₁₆ is available at <https://github.com/cornell-netlab/p4check>

Following the same line of work, $\Pi 4$ is a language inspired by P4 with dependent types to enable checking more complex properties [6], [8]. Interestingly, thanks to the fact the language is not Turing complete, typechecking of $\Pi 4$ is decidable, a rare property for dependently typed languages. This language seems more prototypical than SafeP4 with limitations with respect to P4 such as the absence of registers to store information throughout the processing of multiple packets. The implementation doesn't seem to be available.

D. Runtime Verification

The DBVal toolchain [14] aims at verifying at runtime that some properties are verified, by the use of assertions (like the `assert` builtin of most programming languages). This assertion-based infrastructure uses network-specific constructions, to ease the expression of assertions: a filtering capacity (to check assertions only on some packets), capacity to reason on the path of the packet through the architecture model.

The prototype source code is freely available but no license is provided.

bf4 [5] is a tool performing some static analysis on P4 programs to infer rules on the tables guaranteeing the absence of some classes of bugs (mostly runtime errors). The program is then instrumented so as to detect at runtime the insertion of table entries violating those predicates.

The tool was implemented as a backend for the p4c compiler but its availability is not mentioned in the paper.

This approach allows to catch errors in the semantics of a P4 program, but also to detect errors in the compilation chain or in the execution platform that can not be detected only by looking at P4 code and architecture and target specification.

The main goal of these tools (during verification at runtime) seems not suitable for our aeronautical context, where the objective is not to detect errors at runtime but to rather prevent occurrences of errors at design time. Nevertheless, it may be used for test purposes, during the development phase. Moreover, the tool syntaxes have been designed to catch P4 oriented properties and may be an inspiration for other works.

⁵Except if the programmer voluntarily uses some trapdoor like 'Obj.magic' to bypass the typechecker, but we all know that "Obj.magic is not OCaml".

E. Test Generation

To trust P4 program verification results, one need to trust their execution target. Ideally, one would get formal proofs, down to transistor level, that the target satisfy its expected semantics. However, in practice, the hardware is often a black box. Then, one can only rely on testing to check that it matches its specification. Except in specific cases, exhaustive testing is unachievable. However, given a P4 program, one can at least try to ensure that the exercised test cases provide a good coverage of the program. That is, that as many part of the program as possible are exercised in the test set. The number of test cases that can be hand made being limited, there is an interest in automatically generating test cases. There are a few works in the area, most being specific to a given target. P4TestGen [19] is a test generation tool that attempts to be more generic, enabling the user to extend it with new target specifications.

Indeed, generating tests requires a "whole-program semantics" of P4 programs, that is the combination of the P4 language semantics and some target specific semantics mandating the scheduling of P4 program blocks with "interstitial target-specific elements". Given a target specific semantics and a P4 program, P4Testgen is then able to generate a set of test cases, that is a sequence of pairs (input, expected output). These tests can then be executed on the target platform to check that provided with each input, its actual output matches the expected one. It is worth noting that the input here consists in both packets received and table configurations.⁶ The tool attempts to generate tests that offer a good coverage of the program. There are multiple variations of the notion of coverage, among which path coverage and statement coverage. Path coverage ensures that every possible path in the program is exercised, it can require an exponential number of tests in the program size⁷. Statement coverage is a weaker definition, only ensuring that each program line is exercised, thus only requiring a number of tests linear in the program size. P4Testgen chose to ensure statement coverage, as real-world P4 programs tend to have a lot of paths.

Non determinism is a serious obstacle to test generation. This nondeterminism can be handled by leaving don't cares in the generated test outputs. P4Testgen uses taint analysis (a variation of typing) to keep track of such nondeterminism. This nondeterminism can come from undefined behaviors in P4 leading to target specific behavior (for instance rejecting packets in the parser can lead to target dropping the packet, or considering the headers uninitialized, or silently adding padding to initialize the headers). Non determinism can also come from reading uninitialized variables, or random number generators.

⁶However, in our case we are probably more interested in verifying a P4 program along with some given tables, so we would rather consider the tables as part of the program, which could only ease the testing process, beyond the expectable small adaptation to the P4TestGen tool to handle this particular point.

⁷Just think of a sequence of n if-then-elses, this program has 2^n paths.

P4Testgen implementation relies on SMT solvers: it selects a path in the program, encodes it as an SMT problem⁸ and when the solver finds a model, it generates a test case. Heuristics are used to try to maximize statement coverage with few paths.

The author tested their tool on 4 targets: "v1model architecture of BMv2", "ebpf_model for the Linux kernel", "tna and tna2 architecture for the Tofino 1 and 2 chip respectively"⁹ with their example programs. They found 16 bugs in the toolchain for the Tofino compiler and 9 bugs in the toolchain of BMv2.

The C++ implementation is available under an open source Apache2 license in the reference P4 compiler repository.¹⁰

V. DEDUCTIVE PROOFS

We'd like to apply usual deductive verification techniques on imperative languages, like provided by the Why3 [11] tool, to the P4 language. To experiment deductive proofs on P4 code, we chose to go through an intermediary form in C. This workflow presents many drawbacks. Indeed, C being a much lower level language than P4, going through it means we have to manage many details we are not really interested in, like pointer arithmetic for instance. However, C has the advantage of being already equipped with readily usable deductive proof frameworks like Frama-C [2]. If we were to do some actual deductive-proof work on P4 code, our best bet would probably be to develop a P4 frontend to a tool like Why3 [11].

So we first need to translate the P4 program of our running example, from Section III-B, into some piece of C code. We first attempted to do that using tools from the literature [12] but they were working on some old intermediary form of the P4 official compiler parser and were no longer working on the current compiler.¹¹ We eventually resorted to the official-compiler backend for eBPF [17], which happens to output C code. Thus, from our running example presented in section III-B we obtain the following C code, after some slight modifications to fit in the paper and be readable by Frama-C.

Two files are generated `basic.h` and `basic.c`. The header file `basic.h` mostly contains the declaration of structures for the packet headers, which are the direct translation of the ones in the P4 source code.

```
/* Automatically generated by p4c-ebpf from basic.p4 */
#ifndef _P4_GEN_HEADER_
#define _P4_GEN_HEADER_
#include "ebpf_kernel.h"

#define MAP_PATH "/sys/fs/bpf/tc/globals"
struct ethernet_t {
    u64 dstAddr; /* macAddr_t */
    u64 srcAddr; /* macAddr_t */
    u16 etherType; /* bit<16> */
    u8 ebpf_valid;
};

struct ipv4_t {
    u8 version; /* bit<4> */
    u8 ihl; /* bit<4> */
    u8 diffserv; /* bit<8> */
    u16 totalLen; /* bit<16> */
    u16 identification; /* bit<16> */
    u8 flags; /* bit<3> */
    u16 fragOffset; /* bit<13> */
};
```

⁸Loops in the parser are unrolled, up to some bound.

⁹Which means they implemented each target semantics in the tool.

¹⁰<https://github.com/p4lang/p4c/tree/main/backends/p4tools/modules/testgen>

¹¹<https://github.com/p4lang/p4c>

```

u8 ttl; /* bit<8> */
u8 protocol; /* bit<8> */
u16 hdrChecksum; /* bit<16> */
u32 srcAddr; /* ip4Addr_t */
u32 dstAddr; /* ip4Addr_t */
u8 ebpF_valid;
};

struct metadata {
};

struct headers {
    struct ethernet_t ethernet; /* ethernet_t */
    struct ipv4_t ipv4; /* ipv4_t */
};

#ifdef CONTROL_PLANE
static void init_tables()
{
    u32 ebpF_zero = 0;
}
#endif
#endif

```

The `basic.c` file then starts with the usual header include

```

/* Automatically generated by p4c-ebpf from basic.p4 */
#include "basic.h"

#include "ebpf_kernel.h"

```

then comes a sum type of result codes

```

enum ebpf_errorCodes {
    NoError,
    PacketTooShort,
    NoMatch,
    StackOutOfBounds,
    HeaderTooShort,
    ParserTimeout,
    ParserInvalidArgument,
};

```

and a few macros to read or write a selected number of bits

```

#define EBPF_MASK(t, w) (((t) < 1) << (w) - (t) & 1)
#define BYTES(w) ((w) / 8)
#define write_partial(a, s, v) do {
    u8 mask = EBPF_MASK(u8, s);
    *((u8*)a) = ((*((u8*)a) & ~mask) | (((v) >> (8 - (s))) & mask));
} while (0)
#define write_byte(base, offset, v) do { *(u8*)(base + (offset)) = (v); } while (0)

void* memcpy(void* dest, const void* src, size_t num);

#define bpf_trace_message(fmt, ...)
REGISTER_START()
REGISTER_END()

```

The main function then starts, it takes as input a buffer `skb` with the content of the frame to parse

```
int ebpf_filter(SK_BUFF *skb){
```

The function first declares a variable `hdr` of the above structure type and initializes its validity fields to 0 in order to record that nothing has been read yet

```

struct headers hdr = {
    .ethernet = {
        .ebpF_valid = 0
    },
    .ipv4 = {
        .ebpF_valid = 0
    },
};

```

A few other local variables are declared, among which two pointers `ebpf_packetStart` and `ebpf_packetEnd` pointing to the start and end of the buffer to read and an offset `ebpf_packetOffsetInBits` recording the number of bits already read

```

unsigned ebpf_packetOffsetInBits = 0; unsigned ebpf_packetOffsetInBits_save = 0;
enum ebpf_errorCodes ebpf_errorCode = NoError;
void* ebpf_packetStart = ((void*)(long)skb->data);
void* ebpf_packetEnd = ((void*)(long)skb->data_end);
u8 pass = 0;
u32 ebpF_zero = 0;
unsigned char ebpf_byte;
u32 ebpf_pkt_len = ebpf_packetEnd - ebpf_packetStart;

```

The control flow first moves to `start`, a label to be defined later. Each state of the P4 parser then gets translated to some sequence of statements, following such a label. The control flow in the P4 automaton will thus be implemented by `gotos`.

```

goto start;
goto start;

```

The first parser state to appear in the C code is `parse_ipv4`. Note that this is not the same order as in the original P4 code but order doesn't matter, since there is no ordering between states of a finite state automaton. This code first implements the P4 instruction `packet.extract(hdr.ipv4)` by checking that there is enough data to read in the buffer. When this test succeeds, each field of the `hdr.ipv4` structure is then filled with data from the buffer.

```

parse_ipv4: {
/* extract(hdr.ipv4) */
    if (ebpf_packetEnd < ebpf_packetStart
        + BYTES(ebpf_packetOffsetInBits + 160 + 0)) {
        ebpf_errorCode = PacketTooShort;
        goto reject;
    }

    hdr.ipv4.version = (u8)((load_byte(ebpf_packetStart,
        BYTES(ebpf_packetOffsetInBits)) >> 4) & EBPF_MASK(u8, 4));
    ebpf_packetOffsetInBits += 4;

    hdr.ipv4.ihl = (u8)((load_byte(ebpf_packetStart,
        BYTES(ebpf_packetOffsetInBits)) & EBPF_MASK(u8, 4));
    ebpf_packetOffsetInBits += 4;

    hdr.ipv4.diffserv = (u8)((load_byte(ebpf_packetStart,
        BYTES(ebpf_packetOffsetInBits)));
    ebpf_packetOffsetInBits += 8;

    hdr.ipv4.totalLen = (u16)((load_half(ebpf_packetStart,
        BYTES(ebpf_packetOffsetInBits)));
    ebpf_packetOffsetInBits += 16;

    hdr.ipv4.identification = (u16)((load_half(ebpf_packetStart,
        BYTES(ebpf_packetOffsetInBits)));
    ebpf_packetOffsetInBits += 16;

    hdr.ipv4.flags = (u8)((load_byte(ebpf_packetStart,
        BYTES(ebpf_packetOffsetInBits)) >> 5) & EBPF_MASK(u8, 3));
    ebpf_packetOffsetInBits += 3;

    hdr.ipv4.fragOffset = (u16)((load_half(ebpf_packetStart,
        BYTES(ebpf_packetOffsetInBits)) & EBPF_MASK(u16, 13));
    ebpf_packetOffsetInBits += 13;

    hdr.ipv4.ttl = (u8)((load_byte(ebpf_packetStart,
        BYTES(ebpf_packetOffsetInBits)));
    ebpf_packetOffsetInBits += 8;

    hdr.ipv4.protocol = (u8)((load_byte(ebpf_packetStart,
        BYTES(ebpf_packetOffsetInBits)));
    ebpf_packetOffsetInBits += 8;

    hdr.ipv4.hdrChecksum = (u16)((load_half(ebpf_packetStart,
        BYTES(ebpf_packetOffsetInBits)));
    ebpf_packetOffsetInBits += 16;

    hdr.ipv4.srcAddr = (u32)((load_word(ebpf_packetStart,
        BYTES(ebpf_packetOffsetInBits)));
    ebpf_packetOffsetInBits += 32;

    hdr.ipv4.dstAddr = (u32)((load_word(ebpf_packetStart,
        BYTES(ebpf_packetOffsetInBits)));
    ebpf_packetOffsetInBits += 32;
}

```

The `hdr.ipv4` part is then marked as valid.

```
hdr.ipv4.ebpF_valid = 1;
```

And the P4 instruction transition `accept` is implemented by a `goto`.

```

goto accept;
}

```

Then, the `parse_ethernet` state of the P4 parser is implemented in a similar way.

```

start: {
/* extract(hdr.ethernet) */
    if (ebpf_packetEnd < ebpf_packetStart
        + BYTES(ebpf_packetOffsetInBits + 112 + 0)) {
        ebpf_errorCode = PacketTooShort;
        goto reject;
    }

    hdr.ethernet.dstAddr = (u64)((load_dword(ebpf_packetStart,
        BYTES(ebpf_packetOffsetInBits)) >> 16) & EBPF_MASK(u64, 48));
    ebpf_packetOffsetInBits += 48;

    hdr.ethernet.srcAddr = (u64)((load_dword(ebpf_packetStart,
        BYTES(ebpf_packetOffsetInBits)) >> 16) & EBPF_MASK(u64, 48));
    ebpf_packetOffsetInBits += 48;
}

```



```

hdr.ethernet.etherType = (u16)((load_half(ebpf_packetStart,
    BYTES(ebpf_packetOffsetInBits))));
ebpf_packetOffsetInBits += 16;

hdr.ethernet.ebpf_valid = 1;

```

The transition `select` at the end of that state is a bit more involved and gets compiled as if-then-elses.

```

u16 select_0;
select_0 = hdr.ethernet.etherType;
if (select_0 == 0x800) goto parse_ipv4;
if ((select_0 & 0x0) == (0x0 & 0x0)) goto accept;
else goto reject;

```

Finally we get the `accept` and `reject` states, that are always implicit in P4 parsers.

```

reject: {
    return TC_ACT_SHOT;
}

accept:
{
    u8 hit;
    {
        pass = true;
    }
}
ebpf_end:
if (pass)
    return TC_ACT_OK;
else
    return TC_ACT_SHOT;
}

```

To this code, we add, just before the first line of the function, a special comment, starting with `@`, that contains annotations for Frama-C. The `requires` are pre-conditions and the `ensures` are post-conditions. Frama-C will use SMT (Satisfaction Modulo Theory) solvers to prove that whenever the preconditions are satisfied and the function terminates, its output satisfies the post-condition.

```

/*@ requires ((void*)(long)skb->data_end) >= ((void*)(long)skb->data) + BYTES(112+160);
@ // The following requires states that the two fields are comparable
@ requires \exists unsigned offset; ((void*)(long)skb->data_end
@ == ((void*)(long)skb->data) + offset;
@ ensures \result == TC_ACT_OK ; */

```

Here, the contract means that whenever the function has at least 112+160 bits in the `skb` input buffer, then the function successfully returns `TC_ACT_OK`. The second `requires` is more technical, it is here to tell Frama-C that the two pointers `skb->data` and `skb->data_end` point to the same memory region, meaning they are comparable, which is required by the inequalities appearing in the guards in the program. Without this precondition, the memory model of Frama-C assumes different pointers point to different memory regions and are incomparable. The property is then proved in a matter of seconds by SMT solvers. However, this kind of technical details makes the method very labor intensive.

To dramatically minimize the human effort, we think analysis should not happen at the level of C code, but directly on the P4 programs. This would require designing a tool enabling to directly translate P4 code to, for instance, the Why3 deductive verification tool [11], rather than going through a C intermediate representation. Finally, it is also unclear how maintainable the proofs would be, across successive versions of the analysis tool or SMT solvers.

VI. TEST GENERATION

The P4Testgen tool [19] takes as input a P4 program and uses solvers to generate test cases while attempting to achieve

a good line coverage, that is having as many lines of the input P4 program covered by at least one test case. The generated test cases are pairs of (input, expected output), where the inputs are carefully selected to maximize coverage of the P4 program, and the outputs are the one expected for each input, according to the semantics of P4. Those tests case can then be replayed on the hardware to test, checking that the output on the hardware matches the expected output. Thus, one can test that a given hardware¹² behaves the same than a given P4 program, at least on the test cases.

For instance, on the running example, from Section III-B, we would expect two test cases, each exercising one branch of the `if` (`hdr.ipv4.isValid()`).

The technique is complementary to the previous verification of functional properties of P4 programs. It doesn't aim at proving anything on programs but rather at checking that an implementation of a program keeps the same semantics. Contrary to the previous verification of functional properties, the technique is very "cheap" as it doesn't require writing any specification. Indeed, the P4 program is the specification here.

VII. CONCLUSION

The P4 language is designed to program network switches. We studied its use as a way to specify embedded network switches, independently of the way they are designed and manufactured. Implementing the expected behavior of a future switch in a P4 program could then enable to both

- conduct verification activities on this P4 program, to ensure some expected properties of the design ;
- check that the resulting hardware meets the P4 specification.

Our goal was a preliminary assessment of the feasibility of the approach.

State of the art already features a wide range of works on verification and validation of P4 programs. Unfortunately, no standard specification language for P4 seem to have emerged yet, meaning we don't have anything comparable to the Java Modelling Language (JML) for Java or the ANSI C Specification Language (ACSL) for C. More precisely, the P4 language itself is equipped with a well defined semantics, but we lack another consensual language to express the functional properties we'd like to verify on P4 programs.

One of the most stringent limitation is the strong dependency of the P4 semantics on the architecture target. For instance, things like

- the signature of the blocks ;
- the scheduling of the blocks (can blocks treating different packets be interleaved for instance) ;
- primitives like `packet.emit`

are all architecture dependent, hindering the verification of many properties. Indeed, one can notice that the property studied in section V did not involve any of those aspects.

¹²This hardware can either be an actual hardware or the simulation of some VHDL implementation for instance.

Less crucially, configuration tables are usually considered as a dynamic input of the P4 programs, whereas for critical embedded applications, they should rather be considered as almost part of the P4 program and be included in the verification activity.

Finally, there is no notion of time in P4. This makes it impossible to express properties about rate limiting that are critical in aircraft embedded networks like AFDX or TSN. This would require either designing some extension of P4 to handle time or writing directly the specification in some target language of P4 like DPDK. For instance, to limit rate on input ports in order to mitigate babbling idiot errors, it is common to use token bucket policing elements. Those include a local variable that is steadily incremented with time. Modeling such a variable in P4 would then require some access to some clock variable. Hardware switches being usually already equipped with a clock, it seems to be mostly a matter of extending the language, its semantics and the P4 compiler, at least for the target of interest.

Our work let us hope that using P4 programs as an intermediary specification language while designing an embedded switch could be a good way to formally specify the behavior of the future hardware, by :

- 1) verifying some expected properties on the P4 program, before or while the hardware itself is designed ;
- 2) once the hardware is available, check that its behavior matches the behavior prescribed by the P4 program.

The second activity seem to be a low cost, no regret, one. Indeed, for critical systems, handwritten tests are already performed anyway, so adding more, automatically generated, tests should only incur a marginal cost overhead. The cost-benefit ratio of the first activity is less clear. Formal verification on the P4 program would enable to gain more confidence in the design before any hardware is even built, which could catch errors early in the design process, hence saving massive amounts of wasted effort. Nevertheless, the activity also has a non negligible cost as it requires to develop precise enough specifications to formalize them into mathematical formulas. Whether the benefits outweighs the cost remains unknown and could only be evaluated by actually attempting such a verification activity on some use case. Unfortunately, we currently have no precise plans to conduct that evaluation on our industrial use case.

REFERENCES

- [1] Johannes Blaser. Nano p4: Towards formal verification of p4 and p4 applications using isabelle/hol. Master's thesis, Vrije Universiteit Amsterdam and University of Amsterdam, 2021. supervised by Manuel Wiesinger.
- [2] Loïc Correnson, Pascal Cuoq, Florent Kirchner, André Maroneze, Virgile Prevosto, Armand Puccetti, Julien Signoles, and Boris Yakobowski. *Frama-C User Manual*.
- [3] Ryan Doenges, Mina Tahmasbi Arashloo, Santiago Bautista, Alexander Chang, Newton Ni, Samwise Parkinson, Rudy Peterson, Alaia Solko-Breslin, Amanda Xu, and Nate Foster. Petr4: Formal foundations for p4 data planes. *Proc. ACM Program. Lang.*, 5(POPL), jan 2021.
- [4] Ryan Doenges, Tobias Kappé, John Sarracino, Nate Foster, and Greg Morrisett. Leapfrog: certified equivalence for protocol parsers. In Ranjit Jhala and Isil Dillig, editors, *PLDI '22: 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation, San Diego, CA, USA, June 13 - 17, 2022*, pages 950–965. ACM, 2022.
- [5] Dragos Dumitrescu, Radu Stoenescu, Lorina Negreanu, and Costin Raiciu. bf4: towards bug-free P4 programs. In Henning Schulzrinne and Vishal Misra, editors, *SIGCOMM '20: Proceedings of the 2020 Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication, Virtual Event, USA, August 10-14, 2020*, pages 571–585. ACM, 2020.
- [6] Matthias Eichholz. *Type-Safe Data Plane Programming*. PhD thesis, Technical University of Darmstadt, Germany, 2022.
- [7] Matthias Eichholz, Eric Hayden Campbell, Nate Foster, Guido Salvaneschi, and Mira Mezini. How to avoid making a billion-dollar mistake: Type-safe data plane programming with safep4. In Alastair F. Donaldson, editor, *33rd European Conference on Object-Oriented Programming, ECOOP 2019, July 15-19, 2019, London, United Kingdom*, volume 134 of *LIPICs*, pages 12:1–12:28. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.
- [8] Matthias Eichholz, Eric Hayden Campbell, Matthias Krebs, Nate Foster, and Mira Mezini. Dependently-typed data plane programming. *Proc. ACM Program. Lang.*, 6(POPL):1–28, 2022.
- [9] Burak Ekici, Alain Mebsout, Cesare Tinelli, Chantal Keller, Guy Katz, Andrew Reynolds, and Clark W. Barrett. Smtcoq: A plug-in for integrating SMT solvers into coq. In Rupak Majumdar and Viktor Kuncak, editors, *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part II*, volume 10427 of *Lecture Notes in Computer Science*, pages 126–133. Springer, 2017.
- [10] Yliès Falcone, Klaus Havelund, and Giles Reger. A tutorial on runtime verification. In *Engineering Dependable Software Systems*, 2013.
- [11] Jean-Christophe Filiâtre and Andrei Paskevich. Why3 — where programs meet provers. In Matthias Felleisen and Philippa Gardner, editors, *Proceedings of the 22nd European Symposium on Programming*, volume 7792 of *Lecture Notes in Computer Science*, pages 125–128. Springer, March 2013.
- [12] Lucas Freire, Miguel Neves, Lucas Leal, Kirill Levchenko, Alberto Schaeffer-Filho, and Marinho Barcellos. Uncovering bugs in p4 programs with assertion-based verification. In *Proceedings of the Symposium on SDN Research, SOSR '18*, New York, NY, USA, 2018. Association for Computing Machinery.
- [13] Ali Kheradmand and Grigore Rosu. P4K: A formal semantics of P4 and applications. *CoRR*, abs/1804.01468, 2018.
- [14] K Shiv Kumar, Ranjitha K, P S Prashanth, Mina Tahmasbi Arashloo, Venkanna U., and Praveen Tammana. Dbval: Validating p4 data plane runtime behavior. In *Proceedings of the ACM SIGCOMM Symposium on SDN Research (SOSR)*, SOSR '21, page 122–134, New York, NY, USA, 2021. Association for Computing Machinery.
- [15] Jed Liu, William Hallahan, Cole Schlesinger, Milad Sharif, Jeongkeun Lee, Robert Soulé, Han Wang, Călin Cașcaval, Nick McKeown, and Nate Foster. P4v: Practical verification for programmable data planes. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication, SIGCOMM '18*, page 490–503, New York, NY, USA, 2018. Association for Computing Machinery.
- [16] Zohar Manna and Amir Pnueli. *Temporal verification of reactive systems: safety*. Springer, 1995.
- [17] Sebastiano Miano, Matteo Bertrone, Fulvio Rizzo, Massimo Tumolo, and Mauricio Vásquez Bernal. Creating complex network services with eBPF: Experience and lessons learned. In *2018 IEEE 19th International Conference on High Performance Switching and Routing (HPSR)*, pages 1–8. IEEE, 2018.
- [18] The P4 Language Consortium. *P4₁₆ Language Specification*, version 1.2.3 edition, 2022.
- [19] Fabian Ruffy, Jed Liu, Prathima Kotikalapudi, Vojtech Havel, Rob Sherwood, Vlad Dubina, Volodymyr Peschanenko, Nate Foster, and Anirudh Sivaraman. P4testgen: An extensible test oracle for P4. *CoRR*, abs/2211.15300, 2022.
- [20] Glynn Winskel. *The Formal Semantics of Programming Languages*. The MIT Press, 1993.

Yet another experience on TSN tools interoperability for critical embedded networks

Philippe Cuenot
Continental Automotive
Toulouse, France

philippe.cuenot@continental-corporation.com

Thierry Leydier
Virtualité Réelle
Saint-Orens, France

thierry.leydier@virtualitereelle.com

Damien Fruchard
Airbus Defence and Space
Toulouse, France

damien.fruchard@airbus.com

Massimo Barbero
Thales Alenia Space
Cannes, France

massimo.barbero@thalesaleniaspace.com

Quentin Bailleul
IRT Saint Exupéry
Toulouse, France

quentin.bailleul@irt-saintexupery.com

Abstract—The introduction of Ethernet into critical embedded applications opens new needs to master and secure network development and deployment. While Ethernet is a well known Information Technology (IT) brick and deployed in the industry on a case-by-case basis, Time-Sensitive Networking (TSN) complements have only recently emerged in the aerospace and automotive industries as a promising solution to provide real-time, reliability, and availability guaranties for safety-critical systems. The complexity and diversity of TSN mechanisms enforce the use of specialized tools to assist the network engineer for the design, configuration and deployment of the network parameters. On the other hand, IETF has proposed Yet Another Next Generation (YANG) modeling language for interoperability in configuration and monitoring of various network devices. In this paper, we propose to revisit and complement the YANG standardised model in order to enable tool interoperability, with the aim of providing these complements as open source. The benefit of the proposed YANG model will be demonstrated on a TSN industrial use case with a set of tools ranging from network design and configuration to deployment on a Proof of Concept (PoC) platform.

Index Terms—Embedded Network, Tool Interoperability, Ethernet TSN, YANG model

I. INTRODUCTION

Aerospace and automotive industries are moving to Software Defined Systems (SDS) based on virtual resource allocation: a control layer is deployed in the system in order to configure virtual resources for the services offered by the specific domain application. In this sense SDS breaks the tight coupling between the software (SW) and the hardware (HW) of classical information systems, where dedicated device performs the domain application service. Such systems offer an easier management of HW and SW resources for the application but require higher design constraints for a flexible and generic execution platform. At the same time, classical information systems lack of scalability, flexibility and configurability. The new SDS-related architecture paradigms can be applied to critical embedded systems and the associated communication network where the increase of data exchange in terms of bandwidth, the rising number of devices deployed

in the network and the need of standardization of the network interfaces suggest to naturally move to switched Ethernet network, with the advantage of reuse and cost reduction on the overall infrastructure. The critical expected properties of embedded applications with mixed criticality traffic imply preconditions on the network, such as tight and predictable communication traversal time, reliable and safe communication, strong availability of data/frame exchanged coupled to network configuration complexity. In this context, the Ethernet Time-Sensitive Networking (TSN) can satisfy the requirements of safety-critical embedded applications. The IEEE802.1Q [1] standard complemented with TSN standard extensions emerging in industrial/automotive domain proposes dedicated means to guarantee real-time and safety on these mixed criticality embedded systems. Nevertheless, the design, the provisioning and the verification of the application requirements of a full TSN Ethernet network in a mixed criticality embedded system requires the usage of specialized tools to let the network engineer be more effective, due to the complexity of the TSN standard. Moreover, these tools are necessary to support certification of the final system.

To simplify exchanges between these different tools, we propose in this article to study the use of the YANG data model for tool interoperability purposes. As YANG is designed for network device configuration and monitoring, we proposed augmentations to the standard models to enable their use in the upstream design, study, test and validation phases of critical embedded networks. These modifications are then tested to ensure automated, flexible and vertical interoperability between a network design and analysis tool and a generic hardware test bench configuration tool. This vertical interoperability has enabled us to explore multiple TSN safety-critical embedded industrial use cases (from design to deployment on a hardware generic test bench) in short iterative loop.

This study has been elaborated in the context of an IRT Saint-Exupéry project called EDEN project. EDEN is a multi-domain research project (automotive, aeronautics and space) with the aim to demonstrate confidence in TSN deployment

in critical embedded system domain.

This article digs into the details of the study proposing the following subjects: Section II introduces TSN and YANG. Then, section III analyses the state of art of Ethernet network tools for embedded system. Section IV defines the approach and organization of YANG model applied to Ethernet TSN in the context of EDEN project. Section V documents the use of our YANG models on the industrial use case. Next, section VI discusses the limits of our approach. Finally section VII resumes the conclusion and proposes future work perspective.

II. TSN AND YANG OVERVIEW

TSN provides deterministic and reliable Ethernet communication for real-time traffic, which can coexist with non-real time data traffic. The IEEE802.1Q [1] standard complemented with TSN standard extension is emerging in industrial domain: several working groups for TSN standardisation propose TSN profiles for specific industrial domains. For example, the Aerospace Onboard Ethernet Communications P801.2DP standard is a profile defined for the aerospace industry.

Several TSN's intrinsic mechanisms permit to guarantee QoS for the traffic. The Time-Aware Shaper (TAS), proposed in IEEE802.1Qbv [2], combined with network-wide synchronization provided by gPTP protocol, described in [3], enable time-triggered communication. Credit-Based Shaper (CBS) enables traffic flow regulation controlled by credit as specified in IEEE802.1Qav [4]. Frame Replication and Elimination for Redundancy (FRER), proposed in IEEE802.1CB [5], allows the frame redundancy. Per Stream Filtering and Policing (PSFP), defined in IEEE802.1Qci [6], enable flow filtering and policing.

Yet Another Next Generation (YANG) is a data modeling language intended for network configuration and monitoring. It was proposed by IETF in RFC6020 [7]. YANG language is used to describe a data structure. Instances of this data structure can be exported in XML or JSON format. These instances travel through the network to configure or monitor a device using YANG-based protocol like NETCONF [8] or RESTCONF [9]. YANG can be seen as a successor of Management Information Base (MIB) and YANG-based protocols as a successor of Simple Network Management Protocol (SNMP).

YANG model files are called `module`. To illustrate this paragraph, a very simple `module` modeling a scientific article is given as an example in Listing 1. The `container` `article` is a high level object that groups `leaf`, `list` and even `container`. In this `container`, a `list` of `section` is described. A `list` object is a collection of key/value pair that can contain multiple objects (e.g. `leaves`). In this `section` `list`, a `section-id` `leaf` is described. A `leaf` is an object that can contain only one value. An instantiation of this `module` for a two-section article is given in Listing 2. YANG also proposes a mechanism called `augmentation` which allows a `module`, without modifying in it, to be extended by another `module`. In our example, it's possible to propose a `module` that `augment` the `section` object of the article

`module`, to add a `list` that would describe tables present in the `section` in the same way as the `list` of images.

```

container article {
  description "Article";
  list section {
    key "section-id";
    leaf "section-id" {
      type uint32;
    }
    leaf content {
      type string;
    }
  }
  list image {
    key "title";
    leaf "title" {
      type string;
    }
    leaf "path" {
      type string;
    }
  }
}

```

Listing 1: Simplified example of a scientific article YANG module

```

<article>
  <section>
    <section-id>0</section-id>
    <content>This is the first section
  </content>
  </section>
  <section>
    <section-id>1</section-id>
    <content>This is the second section
  </content>
  <image>
    <title>My chart</title>
    <path>/chart.png</path>
  </image>
</section>
</article>

```

Listing 2: Simplified xml instantiation of the scientific article YANG module

In practice YANG is mainly used in the IT world to manage large infrastructures. In this context, IETF has proposed a number of models such as model for network and interface, or for protocols such as IP. These models are augmented by vendors such as CISCO or HUAWEI to model proprietary mechanisms. In the world of TSN, standardization working group proposes a set of standard YANG `module` to describe TSN mechanisms and enable their configuration and monitoring.

III. RELATED WORK

As mentioned previously, Ethernet IT device providers have developed complete tool suites adopting YANG programmable interfaces using NETCONF or RESTCONF protocol for Ethernet network configuration, supervision, and maintenance [10]. A typical example is the Cisco YANG suite [11] that provides a set of tools and plugins to learn, test, and adopt YANG model for the supervision of an Ethernet network.

For non-IT network, the standardized Centralized Network Configuration (CNC) architecture, part of IEEE Std 802.1Qcc [12], is a key element of TSN standard for configuration of embedded critical network application.

A first implementation of such CNC architecture capable to configure a TSN network of an industrial construction equipment, using the NETCONF protocol and YANG models, was demonstrated in [13]. This demonstration is limited to dynamic TAS configuration for a SMART MPSoC bridge representative of an embedded network application distributed by SoCe company. Despite the use of Linux service for YANG parsing and configuration, the device set-up is operated with specific SoCe drivers, so not applicable to different devices.

In a recent review on TSN network configuration management [14], the authors state that despite the central role of YANG in promoting unified network management, the current standard still needs to be improved to cover configuration of a network composed with different device manufacturers. Similarly in the automotive industry, [15] promotes the use of model-based development, validation and configuration of TSN embedded application. But the authors state that the TSN network configuration and scheduling algorithms are not integrated into the existing software development tools and require further research to enable efficient configuration of TSN network.

Indeed, before configuring the network hardware, it is necessary to design and validate the network configuration using multiple specialized tools (e.g. tools for design, configuration, formal analysis, simulation, ...). In order to use such a variety of tools safely and efficiently, tool interoperability is of major interest. Here also the YANG model can play a central role. To answer the design need, several model-based tools for TSN network architecture design have been proposed on the commercial market. We can mention Pegase from RealTime-at-Work (RTaW) [16] mostly used in automotive industry, Chronos from General Electric [17] targeting aerospace market, TSN designer from RealTime IT [18] or IxNetwork from Keysight [19] addressing the IoT and industrial market. Those products enable to explore and analyze network architecture in order to generate the configuration of TSN mechanisms. But to our knowledge none of them implement a feature enabling non-proprietary tool interoperability all the way down to the hardware network device configuration tool.

RTaW has introduced a first YANG export feature in Pegase tool, completed by a complete tool chain in TSN Studio [20]. TSN Studio enables to configure industrial devices running Linux with standard NETCONF protocol. Despite the use of YANG models, there are still some dependencies to the Pegase data model, for example with regards to the traffic definition or the fixed labeling of physical device. The NETCONF protocol is the only way to configure the devices, which could mismatch the embedded systems requirements. Indeed, NETCONF is quite heavy in term of computing resources and software stack, and it relies on TCP which may not be determinism-friendly. In addition, manufacturers of devices dedicated to embedded systems offer configuration through proprietary solution that do not support YANG interface.

As critical embedded network, industrial IoT domain imposes stringent requirements on the dependability and performance of communication networks. In this context, Chahed

et al. [21] explore TSN state of the art after identifying that the large and continuously evolving set of standards poses challenges for adopters seeking to understand it. They exhibit that clear understanding of use-case, available device resources and constraints are key points and raise that the performance of the control plane design and management operation especially for device configuration is an important aspect aimed to be tackled in the future.

To overcome these limitations around configuration integration in development and validation tools as well as TSN network exploration in the context of critical embedded systems, our contribution proposes to complement standard YANG model to enable full interoperability between TSN network design tool chain and device configuration targeting network exploration experiments.

IV. YANG DATA MODEL FOR TSN NETWORK

A. Preamble and initial objectives

To fully understand our approach, it is important to begin by historically describing our issue:

- Initially, interoperability aimed to facilitate the exchange of network models as well as TSN configurations among various design tools (Pegase RTaW and Timaeus-Net in the first step of our project). This initial step was pivotal in shaping our approach to standard Yang models and their limitations, and in subsequent decision-making terms of design. We can refer to this as "**horizontal interoperability**", as interoperability actors have similar or closely related roles in the network development process.
- Subsequently, our objective shifted towards seeking interoperability between network design tools and deployment tools on hardware platforms. Here, we can speak of "**vertical interoperability**", as interoperability actors have different roles. This second step introduced new challenges.

The pursuit of vertical interoperability first faced tools limitations, as tools did not often offer a means of importing from a Yang structure and only supported a single meta-model. Effective interoperability at that time in our project context was thus unidirectional: RTaW-Pegase to Timaeus. The underlying model at this time was already an augmented model proposed by RTaW. However, this scenario was interesting as it paved the way for a new potential need: the ability of a tool to support a Yang model as an interoperability "parameter". When we aimed to further enhance this vertical interoperability to ensure the sustainability of our solutions, it became evident that adopting a cleaner approach was essential. This involved creating a customized Yang model that we could subsequently disseminate to the embedded network community.

B. Requirements for a model

To create and setup the YANG model, we started with the definition of requirements, driven by 3 principles: universality, diversity, and reversibility.

a) *Universality*: Because our final aim is to ensure the global interoperability of our network design and deployment environment, the solution was not only to cover the network configuration, but to have a full description of the network shared by a large set of tools: design and configuration tools, network simulators, deployment solutions. This first principle of universality is crucial regarding both the structure and content of the model. Particularly, to encompass the scope of vertical interoperability, we need to incorporate into the model, information that will be used in a very localized manner. This principle also diverges from the aim of IETF and IEEE models, which solely targeted network devices. For instance, high level network development tools have to deal with pure graphical information related the rendering of networks (e.g. size and position of nodes inside a display). This kind of information has not interest for the network deployment but can be very important for the network designer. So a YANG model shall support any information required along the full network development process.

b) *Diversity*: At this stage, let's introduce the concept of perspective: for two tools playing the same role in a network design chain, the high-level view of the network may differ. For instance, a critical embedded network would be highly interested in security information, whereas a standard network may not necessarily be concerned with this issue. Another example is openness to applications. For instance, you can limit the traffic definition to the network or extend this definition to the related applications. Indeed, applications can greatly influence traffic behavior and configuration. For example, if the application contains time-triggered temporal constraints and if these constraints are implemented using the time-triggered solution TAS, then the configuration of the TAS shall match these application constraints. Therefore, we need a solution to express these constraints inside the YANG model. This principle of diversity will significantly increase the size of the model; it will also lead to diversity in usage: all elements of the model will not be used in the same way in every network. Thus, there will be diversities in usage or instantiations resulting from the diversity supported by the model as well as the functional diversity of users. So a YANG model shall support any perspective and point of view used along the full network development process.

c) *Reversibility*: The YANG model shall be sufficient so that when a tool exports a network description, it should be able to recreate the same network description by importing the previously exported model. Due to this principle, new information shall be taken into account: implementation information introduced by each tool. A highly interesting example concerns the modeling of TAS schedulers. When this shaper is used, the network design tool assists by calculating the configuration tables of the TAS schedulers, which contain the opening and closing times of gates on the output ports. This configuration constitutes the implementation of TAS and is specific to each TAS configuration algorithm; for example, in Timaeus-Net, there are options that allow for adjusting the porosity of TAS windows relative to the rest of the traffic.

The porosity of a TAS expresses the ability of time triggered traffic to be interlaced with non time triggered traffic. If the porosity is low, then the use of TAS will generate a tunneling effect that will reduce the efficiency of the non time triggered traffic. Changing this option thus leads to a different TAS configuration and different performance for the entire traffic. The modeling proposed by the IEEE standard regarding TAS is very relevant regarding the configuration of gate opening tables. Therefore, this initial information can be reused; however, the IEEE model does not implement any association between the gates configuration and the specific traffic. Thus, if two flows of the same priority and profile (same message size and periodicity) exist, it will not be possible to infer the flows from gate management information. Respecting reversibility can lead to an increase in information: we call this kind of information tool-dedicated information. But other solutions are also possible: in the case of TAS schedulers, for example, we have chosen to address it at the level of the import function itself. Indeed, it would have been too complex and too specific to overload the model in order to trace unequivocally the options of the scheduler creation algorithm. In other words, interoperability stops at the implementation specifics of the exchanging tools. So a YANG model shall support any information which are mandatory to recover a network after an export-import sequence.

d) *Maintainability*: To conclude this list of requirements, adding one final principle that concerns not the creation of the model but its utilization: the principle of maintainability. Indeed, as soon as we recognize that standard models are incomplete, not yet stabilized, or customizable, we must be prepared in an interoperability scenario to encounter a YANG model that is different from the one we are going to create. Alternatively, for a given model, we must be prepared for different interpretations of the same model. We will delve into the implications of this scalability principle later in the article. This last criteria does not directly concern the YANG model, but rather the way the tools could be adapted to work with such model. Our conclusion of this first activity of requirements, was a set of mandatory information that should be managed inside our model: topology information, traffic information, configuration information, dependability information, tool-dedicated information etc.

C. Elaboration of the YANG model

Implementing the model involves defining the classes and relationships necessary to cover our network modeling needs. At this stage, we are torn between two implementation approaches that need to be reconciled: a primary categorical approach, which is to reuse existing classes and relationships from IEEE and IETF modules whenever possible – this is imperative, just because during the deployment phase, we will need to leverage this information. A secondary contingent approach is to address our principles of universality, diversity, and reversibility. To cover the first approach, our starting point was the set of concepts already implemented inside IETF and IEEE standards; IETF standards contain concepts like net-

work, node, link, interface, etc.; IEEE standards contain TSN concepts like bridge, bridge-port, TT scheduler, gPTP/PTP instances, talkers and listeners, etc. We split these standard concepts into 3 categories:

- concepts that can be reused as they are,
- concepts partially matching our requirements and which have to be improved,
- concepts not mandatory for HW configuration, which are not matching our requirements or which could not be improved.

In the first category (concepts that can be reused as they are), we only have concepts which are required for the hardware configuration itself: for instance, to configure the TAS mechanism, we need to define “gate parameter tables” which are containing the TAS schedulers entries. The YANG concepts defined in the `ieee802-dot1q-sched` module play this role perfectly. The second category (concepts partially matching our requirements) contains most other standard concepts: improvement can here be done using the YANG Augment feature. For this study, we organized the YANG folders in order to separate what is standard, and what is customized. Let’s note that even the most obvious concepts must be completed to fit our needs; for instance, the IETF node concept has been augmented to support the “manufacturer-reference”, or the “bench-id” (host name of the node in the test bench network, such as “PC_2”). These fields are useful when deploying the network on a generic platform. The last category contains concepts that are not mandatory for configuration, and which are not generic nor detailed enough to implement some of our needs: for instance, the talker-listener paradigm is defined inside the `ieee802-dot1q-tsn-types` module and is not relevant to implement all kind of traffics. In order to avoid modifications or patches of the standard model, when a part of the model was too far from our need and when it was not possible to augment it, we preferred to create our own class breakdown.

To address the second approach, we need to complement our model with new classes or relationships. Here, we have more freedom, but our approach aims to achieve a complete and high-quality model: at this stage, we aim to ensure that additions do not degrade the overall quality. Once again, compromises will need to be made between the two initial objectives (horizontal and vertical interoperability). To measure the quality of the whole, we use conventional design criteria borrowed from the state of the art in model design: class coherence (classes should implement only one clear and unique concept); coupling (classes should be loosely coupled, avoiding logical and implementation couplings); primitiveness (each class attribute should implement information that cannot be decomposed into elementary information or duplicate already modeled information).

D. Model Overview

We will not expose the entirety of the model (see summary in Table I) but will briefly present some specificities. Let’s

start with the “additions,” which are a few modules we have created to complement existing elements in the standards:

- `Irt-tsn`: This module contains information for TSN configuration. The simplest example here is CBS. Although very old and common, there is no standard for defining the configuration of CBS parameters.
- `Irt-topology`, `irt-interface`, `irt-ptp`: In these three cases associated with standard modules, we have added informative supplements such as buffer sizes, transmission capacities, references to network nodes (for interfaces as with “bench-id” in the interface module presented earlier), etc..

Now let’s talk about classes that we have created from scratch. The simplest example is traffic modeling (Fig 1). The `irt-traffic` module is intended to cover the generic needs for defining traffic that we identified in our research project. Main container is the “Traffic” class: this singleton is supposed to contain all the items used to describe a full traffic. It mainly contains:

- A collection of flows: each flow itself contains a set of cast ; a flow has properties (like its payload or its period) and constraints (like a maximal latency)
- A collection of classes: a class is a high level concept to describe QoS or standard shapers configuration.
- A collection of protocols: a protocol can be used by a flow
- A collection of additional constraints: these constraints are used to express applicative constraints: for instance, for a cyclic traffic, the exchanged windows or for a chained cyclic, the items of the chain.

E. YANG model instance life cycle in the tool chain

According to the interoperability context, several activities are concerned: network model design, network model validation, devices configuration generation, deployment on the HW platform, validation of the HW platform. These activities can be organized around a workflow, creating a life cycle to enrich the YANG instance of the network.

Parts of the YANG model have to be initialized in the design activity, other will be created later. For instance, the actual IP address is only assigned during the platform deployment. This concept of life cycle can be compared to the “config” standard YANG field used to distinguish parameters that actually can be configured: in our case, model concepts have to map to activities of the life cycle. Sometimes, the same initial requirement projected onto two activities will be implemented using two distinct YANG concepts: it is the case of the Traffic Safety requirement. At network design level this requirement consists in being able to define for critical flows some redundant paths. To implement this first level of requirement, due to the reversibility principle, we decided to create the concept of flow cast: a cast is a set of segments; each segment can be single or multiple; a single segment is a path; a multiple segment is a set of paths, having same extremities but distinct intermediate nodes to ensure safety in

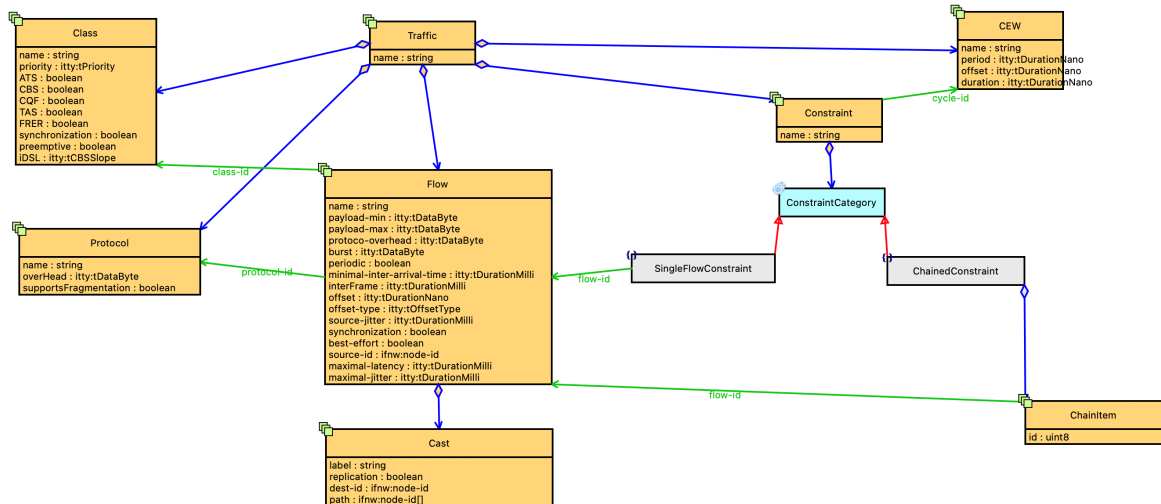


Fig. 1: Class Diagram for Traffic

case of line break. Then at deployment level, the principle is to use the FRER mechanism to implement the redundant paths. A model complement is thus required, focusing on each node ieee802-dot1cb-frer configuration.

F. YANG model implementation in the tools

Both interoperability objectives (horizontal and vertical interoperability) suppose that each tool of the framework is “understanding” the common YANG data model. However, in each tool there is an element of interpretation in this understanding of the model. For example, our YANG model leaves some freedom for port naming (numeric, alphanumeric, etc.); the nature of the tools will also play a role: a tool close to hardware will rely on the hardware identification of the ports while a high level network analysis tool can go so far as to ignore this naming. The consequence is that each tool must be able to adapt its own data model to the generic YANG model. The tools must allow this adaptation to be configured, in order to achieve maximum interoperability.

To ensure the scalability principle of interoperability, the technique of meta-model mapping can be a solution. This technique is based on the following principles:

- Each tool has its own meta-model, which generally corresponds to the tool’s specific design, but also to the underlying purpose of the tool.
- Each tool will use an external YANG meta-model, which is different from its own meta-model. This external meta-model will not be hard-coded in the tool but exchangeable: the tool will therefore offer a principle to select the external YANG meta-model.
- An additional mapping interface will be supported by the tool, an interface that should allow aligning the elements of the native meta-model with the elements of the external meta-model

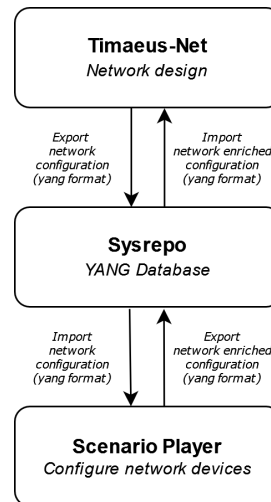


Fig. 2: Block diagram of the tool chain

V. THE EXPERIMENTAL SETUP

In this section, the experimental setup used to validate the interoperability enabled by the model is presented. Starting with a presentation of the tool chain, followed by a presentation of the industrial case study used for this setup, and finally a feasibility study is described and discussed.

A. Tool chain

The tool chain developed as part of this work aims to support the user from the design and configuration of a TSN network right through to deployment on the hardware. It is summarized in Fig. 2. It can be broken down into three distinct tools, which are detailed below.

a) *Design and validation of a TSN network:* Timaeus-Net [22] is a tool allowing the user to describe a network, its message flows and their constraints. Using these informations,

Standard	Standard changed	Contribution
iana-type-if.yang ieee1588-ptp.yang ieeee802-dot1as-ptp.yang ieeee802-dot1cb-frer-types.yang ieeee802-dot1cb-frer.yang ieeee802-dot1cb-stream-identification-types.yang ieeee802-dot1cb-stream-identification.yang ieeee802-dot1q-ats.yang ieeee802-dot1q-bridge.yang ieeee802-dot1q-preemption.yang ieeee802-dot1q-stream-filters-gates.yang ieeee802-dot1q-tsn-types.yang ieeee802-dot1q-types.yang ieeee802-types.yang ietf-inet-types.yang ietf-interfaces.yang ietf-ip.yang ietf-network-topology.yang ietf-network.yang ietf-yang-types.yang	ieeee802-dot1q-psfp.yang ieeee802-dot1q-sched.yang	irt-eden-usecases.yang irt-frer.yang irt-interface.yang irt-ptp.yang irt-topology.yang irt-traffic.yang irt-tsn.yang irt-types.yang

TABLE I: Lists of YANG models used

the tool proposes a configuration of the different TSN mechanisms. Compliance with flows constraints such as latency/jitter is then validated using Network Calculus Approach [23] [24] to compute them in the worst case. When the user is satisfied with their configuration, the tool can then export these different parameters according to the YANG data model described above.

b) Central YANG models storage: These YANG datas are then imported and stored in a centralized database that ensures compliance with the format described in the model. The database used in our case is an open source project called Sysrepo [25]. This database can be queried using NETCONF and it enables interoperability between tools. In the future, other tools (e.g a simulator or analysis tool) could also query this database to perform computations on the network described using Timaeus-Net or other design tools that can export the data in the format described above.

c) Deployment on the targeted hardware network: To deploy the network designed with Timaeus-Net on the hardware, a tool called Scenario Player has been developed. This tool begins by retrieving the configuration described in the central database. It then allows users to describe test scenarios, with fields such as "duration" of the use case `module`, and to completes the Timaeus-Net data by adding information describing the test bench via a graphical user interface. To be more precise, the user maps the objects (end stations, switches, interfaces, ...) described in Timaeus-Net to the test bench hardware using above-mentioned fields such as "bench-id" and "manufacturer-reference". The tool then uses these fields to complete the data with the MAC and IP addresses of each device and generate forwarding and ARP tables. These completed datas are then sent back to the central YANG database for storage and potential use by other tools. Next, Scenario Player configures the various test bed devices (switches, end-stations, traffic generators, measurement instruments) using the completed datas describing the previously designed configuration. Finally, it executes the scenario described, enabling

experimental measurements to be carried out on complex use cases.

Note that none of the devices used in the test bed supports a standardized configuration/monitoring protocol such as NETCONF, RESTCONF or other configuration protocol more suited for the critical embedded world. Therefore, Scenario Player translates the configuration of each device according to the proprietary configuration protocol. However, NETCONF is used by our tools to exchange data with the SYSREPO database (the four arrows in Fig. 2).

B. Spatial use case

The following industrial case study was used to investigate the interoperability capabilities of the model described above. This case study is based on the unification and replacement of a satellite's current networks (i.e. MIL-STD-1553 and SpaceWire) using a single 1Gb/s TSN network with 4 switches and 14 end stations. It is described in detail by Chainé et al. in [26].

However, due to hardware limitations (e.g. number of ports or FRER support on Network Interface Cards (NICs)), the case study was reduced to the topology described in Fig. 3. It consists of 5 switches and 10 end stations. The switches SWOBC_A and SWRIU_A have been introduced to overcome the limitation of FRER use on NICs for two end stations, i.e. OBC_A_FhI and RIU_A. From a traffic point of view, there are 101 flows, which have been transposed from the flows transiting on the MIL-STD-1553 and SpaceWire networks currently in use. These heterogeneous flows carry a payload between 2 and 1472 bytes with burst from 1 to 4788 packets. They are grouped by similarity into 13 traffic classes. The lowest latency and jitter constraints for each traffic class are summarized in Table II. To meet these constraints, Timaeus-Net has proposed and configured different TSN latency control mechanisms (i.e. CBS and TAS) for the different traffic classes. The use of the TAS imposes a network-wide common clock. This is provided by the gPTP synchronization protocol.

Traffic class Id	1	2	3	4	5	6	7	8	9	10	11	12	13
Lowest latency constraint (ms)	33	100	125	N/A	N/A	31.25	125	1	125	125	N/A	N/A	N/A
Lowest jitter constraint (ms)	N/A	1	10	N/A	0.001	N/A	0.5	0.1	N/A	N/A	N/A	N/A	N/A

TABLE II: Lowest duration constraint for each traffic class

To meet availability requirements, three synchronization domains are used. And finally, the flows between OBC_A_FhI and RIU_A are replicated using the FRER mechanism on the first switch in the path and eliminated on the last.

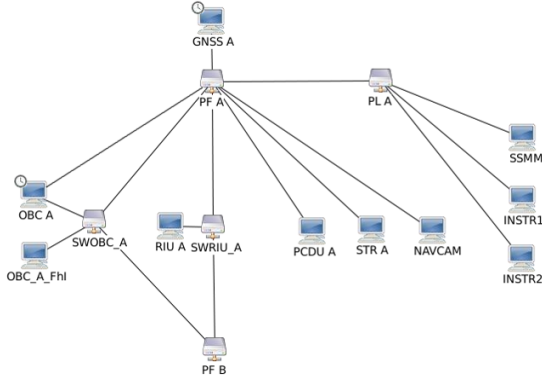


Fig. 3: Unified TSN network satellite topology (view from Timaeus-Net)

C. Feasibility study

To illustrate the interoperability of our augmented YANG model, the case study described above is designed, configured, validated and deployed using the tool chain.

The first step in this tool chain is to design and configure the network. The topology and message flows (periodicity, size, latency constraints, etc.) of the satellite case study are first described in Timaeus-Net. Then, after exploring several configurations using formal analysis and optimization algorithms, a configuration is selected and then exported as a set of xml files following the data format described in the YANG model. These files are then loaded into the database using NETCONF.

Scenario Player then queries the database using NETCONF to retrieve the network described and configured above. It then enriches this data with the information required for deployment on the hardware. For example, each switch, end station and network interface described in Timaeus-Net is assigned to a switch, end station or interface available on the test bench. These assignments also trigger the addition of data such as the forwarding tables needed to configure flow paths, or the information needed to configure traffic generators such as source and destination mac addresses and VLAN numbers for each flow. These new datas are then sent to the database, once again using NETCONF, for future use by Scenario Player or other tools.

All the data are then parsed, separated by device and used to configure the corresponding devices. In this case study, two types of switch from two different manufacturers, 3 desktop computers running Ubuntu 20.04 totalling 8 NICs, and 2

FreeRTOS targets are automatically configured using the data stored in the database.

Although imperfect, the following metrics illustrate well the complexity of the case study deployment, due to the number of parameters to be configured, and the importance of the effort put into tool interoperability.

The first of these metrics is the number of `leaves` stored in the database. When first imported, this number is 4138 `leaves`. After adding test bed specific datas, it increases to 6462 `leaves`. The distributions are detailed in Table III. We can note that 87.4% of the `leaves` in the first import are additions proposed by our contribution. 93% of these are linked to the description of traffic crossing the network. After adding test bed specific datas, our contribution represents only 58.7% of the `leaves`. This is mainly due to the creation of forwarding tables stored in the `ieee-bridge` model and the addition of ARP tables in the `ietf-interface` model. Other industrial case studies designed and deployed with this toolchain showed different distributions (e.g. 10301 `leaves`, 45% of which came from our contribution), explained by differences such as fewer flows with more hops, different TSN mechanisms and larger numbers of interfaces. The differences caused by the choice of TSN mechanisms are illustrated in Table IV, which shows three different configurations on the same topology of an industrial automotive network use case. Table V describes a metric that is more stable to changes of use case. It details the distribution of `leaf` types used to meet our interoperability needs. We can see that few data types are needed (114 after enrichment) to describe and deploy a network, but it's the instantiation of these that greatly increases the number of `leaves`. Take the example of the 41 `leaves` needed to describe a flow, but the instantiation of this model for 101 flows leads to the use of 3483 `leaves` on the satellite use case. We also note that 66.7% of the `leaf` types used in this use case originate from our contribution, which highlights the shortcomings of standard models in terms of tool interoperability.

The second metric is the number of commands made by Scenario Player to configure the 15 devices that reach 2919. This total is broken down into 2199 commands for switches and 720 commands for end stations.

The next metric is the time required to deploy a case study on the hardware. Using the data exported by Timaeus-Net, an experienced user can perform model enrichment in less than ten minutes, and automatic hardware configuration takes less than two minutes. Without this tool interoperability, an experienced user would need at least 8 hours and 45min to reproduce the configuration described in Timaeus-Net. This duration is deducted from the time needed to configure the PL A switch by hand (35min multiplied by 15 devices). This

	Original import		Enriched import	
	Standard	IRT	Standard	IRT
ietf-network	143	106	143	133
ietf-interface	352	112	646	156
irt-usecase	0	0	0	3
ieee-bridge	0	0	1841	0
irt-traffic	0	3382	0	3483
ieee1588-ptp	25	18	39	18
Total	520 (12.6%)	3618 (87.4%)	2669 (41.3%)	3793 (58.7%)

TABLE III: Distribution of the standard and IRT `leaf` instantiations for the two importations of the use case in the central database

Configuration	CBS	FRER	Std	Irt	Total
1	No	No	990 (27%)	2729 (73%)	3719
2	Yes	No	990 (23%)	3241 (77%)	4321
3	No	Yes	990 (26%)	2799 (74%)	3789
4	Yes	Yes	990 (23%)	3311 (77%)	4391

TABLE IV: Distribution of the standard and IRT `leaf` instantiations after first importation of three network configurations of an automotive use case

duration is probably very optimistic, as the PL A is the switch with the simplest configuration (82 commands). Moreover, this metric does not take into account the possibility of human error.

And finally, on the other side of the tool chain, this interoperability allows us to change design tools without impacting the rest of the chain.

VI. DISCUSSION AND LIMITS

We demonstrated the feasibility and benefits of tool interoperability using YANG models in the previous section, however there are limitations which are discussed in this section.

First limitations are about the design of the model itself, according to the criteria that we have defined. The universality and diversity principles have conduct us to cover all points of view and all needs for modeling a network and associated traffic. This has led to add or amend missing elements being to the standard models. However, these standard models will evolve and will certainly be partially completed. As a consequence, our add-on shall be updated.

The reversibility principle has highlighted that the cost of implementing an import feature is much higher than that of cost for the export features. Therefore, for a commercial tool the benefits of importing is limited because it does not provide any direct extra features in its own solution.

The last principle, namely maintainability, leads to establish a specific implementation which consists in considering the YANG meta-model as a parameter of the interoperability features. This can be complex especially if the tool does not implement a meta-model to manage its data and/or if it contains limitations. Indeed, some tools of the tool chain are more or less impacted than others by change in the associated models. For example, Timaeus-Net has its own internal meta-model. The data mapping with the YANG model is direct, so a change in model can be handled easily by a few change in the transformation mapping. Unlike Scenario Player that

implements a simple parser from YANG to the proprietary configuration tools of the various devices, any change in the YANG model may require more consequent code evolution. This enforces the question of the stability of YANG models and the interconnection between the model and the tools.

Other limitations are inherent to the standard YANG models. One can cite the lack of completeness between the various module creators, for example such as the IETF or IEEE, or the restrictions imposed by certain models. For example, the fact that the standard interface model proposed by the IETF only enables the modeling of single device interfaces is a significant limitation when using the YANG model to describe a network that will inevitably contain multiple devices. This limitation arises from the fact that it's not possible to reference YANG elements which are not explicitly defined as `prototypes` from another element. Therefore, it's not possible to create for each node of the network, the collection of its interfaces. In our case, we used workaround in such cases but standard evolution may be needed. Note that, this limitation is not encountered with the normal use of YANG, but only when trying to describe a complete network, as in our work.

Finally, the last limitation is economic. Indeed, the market of TSN network design and configuration tool for critical embedded system is very fragmented and today established as a niche market whereas tool interoperability is a competitive criteria. The deployment of YANG models as core technology for vertical and horizontal tool interoperability standard for classical IT domain, even supported by vendor specific extension (e.g. with proprietary YANG `augment`s), needs to be organized around an alive and open industrial ecosystem. This would enable to increase the YANG models maturity and enlarge the community for additional TSN tool market such as network simulator, network verification and validation, etc. to foster a non-competitive technology facilitating tool supplier collaboration with positive return on invest. Through the work presented in this article with the support of industrial partners and IRT Saint Exupéry hosting TSN research activities, this contribution aims to push forward a YANG ecosystem related to embedded critical system.

VII. CONCLUSION AND FUTURE WORK

In this paper, we propose to `augment` the standard YANG models to ensure vertical interoperability of the tools needed to develop and deploy complex TSN networks. We illustrate the usefulness of this new model by using it to connect two tools

	Original import		Enriched import	
	Standard	IRT	Standard	IRT
ietf-network	8	14	8	15
ietf-interface	16	4	19	5
irt-usecase	0	0	0	3
ieee-bridge	0	0	8	0
irt-traffic	0	41	0	42
ieee1588-ptp	7	3	11	3
Total	31 (33.3%)	62 (66.7%)	46 (40.4%)	68 (59.6%)

TABLE V: Distribution of the standard and IRT leaf type for the two importations of the use case in the central database

used to design and configure a satellite’s critical embedded TSN network and deploy it on hardware targets. The proposed models and augmentations are delivered in open source [27] to initiate an ecosystem dedicated to TSN tool interoperability for embedded critical application. However, this first version needs to be challenged by commercial products that aim to offer interoperable solutions for TSN networks in order to evolve.

Several future works are envisioned around this model in a follow up project. First, we planned to extend the model to support more feature like instrumentation devices. Other tools, such as a network simulator, are also planned to be connected to the central database to take advantage of this unique interface and will most probably require new additions to the model. Finally, this work will also be continued to propose a YANG-based configuration and monitoring protocol adapted to the world of critical embedded systems.

ACKNOWLEDGMENT

The authors thank all people and industrial partners involved in the EDEN project. The French Research Agency (ANR) and the partners of IRT Saint-Exupéry Scientific Cooperation Foundation support this work: Airbus Operations, Airbus Defence and Space, CNES, Continental Automotive, INPT/IRIT, ISAE-SUPAERO, ONERA, Safran Electronics and Defence, Thales Alenia Space and Thales Avionics.

REFERENCES

- [1] “Ieee standard for local and metropolitan area networks—bridges and bridged networks,” *IEEE Std 802.1Q-2022 (Revision of IEEE Std 802.1Q-2018)*, pp. 1–2163, 2022.
- [2] “Ieee standard for local and metropolitan area networks – bridges and bridged networks - amendment 25: Enhancements for scheduled traffic,” *IEEE Std 802.1Qbv-2015 (Amendment to IEEE Std 802.1Q-2014 as amended by IEEE Std 802.1Qca-2015, IEEE Std 802.1Qcd-2015, and IEEE Std 802.1Q-2014/Cor 1-2015)*, pp. 1–57, 2016.
- [3] “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.
- [4] “Ieee standard for local and metropolitan area networks— virtual bridged local area networks amendment 12: Forwarding and queuing enhancements for time-sensitive streams,” *IEEE Std 802.1Qav-2009 (Amendment to IEEE Std 802.1Q-2005)*, pp. 1–72, 2010.
- [5] “Ieee standard for local and metropolitan area networks—frame replication and elimination for reliability,” *IEEE Std 802.1CB-2017*, pp. 1–102, 2017.
- [6] “Ieee standard for local and metropolitan area networks—bridges and bridged networks—amendment 28: Per-stream filtering and policing,” *IEEE Std 802.1Qci-2017 (Amendment to IEEE Std 802.1Q-2014 as amended by IEEE Std 802.1Qca-2015, IEEE Std 802.1Qcd-2015, IEEE Std 802.1Q-2014/Cor 1-2015, IEEE Std 802.1Qbv-2015, IEEE Std 802.1Qbu-2016, and IEEE Std 802.1Qbz-2016)*, pp. 1–65, 2017.
- [7] M. Björklund, “YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF),” RFC 6020, Oct. 2010. [Online]. Available: <https://www.rfc-editor.org/info/rfc6020>
- [8] R. Enns, M. Björklund, A. Bierman, and J. Schönwälder, “Network Configuration Protocol (NETCONF),” RFC 6241, Jun. 2011. [Online]. Available: <https://www.rfc-editor.org/info/rfc6241>
- [9] A. Bierman, M. Björklund, and K. Watsen, “RESTCONF Protocol,” RFC 8040, Jan. 2017. [Online]. Available: <https://www.rfc-editor.org/info/rfc8040>
- [10] J. Schönwälder, M. Björklund, and P. Shafer, “Network configuration management using netconf and yang,” *IEEE communications magazine*, vol. 48, no. 9, pp. 166–173, 2010.
- [11] “Cisco yang suite,” <https://developer.cisco.com/yangsuite/>.
- [12] “Ieee standard for local and metropolitan area networks—bridges and bridged networks – amendment 31: Stream reservation protocol (srp) enhancements and performance improvements,” *IEEE Std 802.1Qcc-2018 (Amendment to IEEE Std 802.1Q-2018 as amended by IEEE Std 802.1Qcp-2018)*, pp. 1–208, 2018.
- [13] I. Álvarez, A. Servera, J. Proenza, M. Ashjaei, and S. Mubeen, “Implementing a first cnc for scheduling and configuring tsn networks,” in *2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2022, pp. 1–4.
- [14] B. Shi, X. Tu, B. Wu, and Y. Peng, “Recent advances in time-sensitive network configuration management: A literature review,” *Journal of Sensor and Actuator Networks*, vol. 12, no. 4, 2023. [Online]. Available: <https://www.mdpi.com/2224-2708/12/4/52>
- [15] M. Ashjaei, L. L. Bello, M. Daneshalab, G. Patti, S. Saponara, and S. Mubeen, “Time-sensitive networking in automotive embedded systems: State of the art and research opportunities,” *Journal of systems architecture*, vol. 117, p. 102137, 2021.
- [16] “Pegase, realtime-at-work,” <https://www.realtimeatwork.com/rtaw-pegase/>.
- [17] “Chronos, general electric,” <https://www.ge.com/research/project/time-sensitive-networking-tsn>.
- [18] “Tsn designer, realtime it,” <https://www.realtime-it.de/en/index.php/tsn-designer/>.
- [19] “Ixnnetwork, keysight,” <https://www.keysight.com/sg/en/products/network-test/protocol-load-test/ixnetwork.html>.
- [20] “Tsn studio, realtime-at-work,” <https://tsn.studio/>.
- [21] H. Chahed and A. Kassler, “Tsn network scheduling—challenges and approaches,” *Network Journal*, vol. 3, pp. 585–624, 2023.
- [22] M. Barbero, T. Leydier, P. Cuenot, D. Fruchard, and B. Attanasio, “How to design a safe Ethernet TSN network on spacecraft application,” in *Data Systems in Aerospace (DASIA 2023)*, 2023.
- [23] J.-Y. Le Boudec and P. Thiran, Eds., *Network Calculus*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 3–81. [Online]. Available: https://doi.org/10.1007/3-540-45318-0_1
- [24] L. Zhao, P. Pop, and S. Steinhorst, “Quantitative performance comparison of various traffic shapers in time-sensitive networking,” 03 2021.
- [25] “Sysrepo storing and managing yang-based configurations for unix/linux applications,” <https://www.sysrepo.org>.
- [26] P.-J. Chaîne, M. Boyer, C. Pagetti, and F. Wartel, “TSN Support for Quality of Service in Space,” in *10th European Congress on Embedded Real Time Software and Systems (ERTS 2020)*, Toulouse, France, Jan. 2020. [Online]. Available: <https://hal.science/hal-02441327>
- [27] “Eden yang, irt saint exupéry,” <https://sahara.irt-saintexupery.com/embedded-systems/eden-yang>.

Session Tu.2.Po

Poster overview

Tuesday 11th June

14:00

–

Auditorium

Short paper: Acceleration of Embedded Reasoning in Symbolic AI

Youssef Amari^{1,2} Florence De Grancey² Matthieu Roy¹ H el ene Waeselynck¹

¹LAAS-CNRS
Toulouse, France

²Thales
Toulouse, France

Abstract—Current advances in Artificial Intelligence (AI) technologies pave the way to consider new services to assist aircrew, possibly in embedded systems. Symbolic AI reasoning provides both opportunities and challenges for these services. On the one hand, symbolic AI provides proven and explainable results. On the other hand, recent studies showcase that those reasoning methodologies suffer from long and unpredictable execution times, and high memory consumption. Such limitations currently refrain the use of this approach in embedded systems. The objective of this thesis work is to explore ways to deploy such reasoning in embedded architectures focusing on optimisations and benchmarking.

Keywords—Symbolic AI, Reasoning, Avionics, Embedded System, Ontology, Description Logic, OWL, Optimisation.

I. INTRODUCTION

Whereas tremendous strides have been done in hardware and software technologies in aeronautics, new systems must provide the same level in safety as the usual “Federated Architecture”, where there is an absence of interferences [1]. According to [2], we are in the 4th revolution in History: “smart” and “connected” systems. One of the emergent technologies in this 4th revolution is Artificial Intelligence (AI). AI is mainly known for its wide public applications such as image recognition or text generation. These new services now penetrate the aeronautical field to assist operators.

Therefore, one of the envisioned applications is the assistance to the aircrew in order to circumvent human workload limitations. For instance, when a warning light appears in the cockpit, the pilot has to assess, as quickly as possible, the seriousness of the alarm in order to express the criticality of the situation. Detaining a high level of real-time situation monitoring is one of the most complex and critical features in the aviation domain. If perception and prediction steps can be performed by data-based approaches, the comprehension step of this situation must use aeronautic domain knowledge to give a structured meaning of the alarm.

For such case, *symbolic AI* seems a pertinent candidate to implement and manage a real-time situation monitoring.

Symbolic AI is based on different concepts such as first-order logic rules, ontologies, decisions trees, reasoning... This technology detains notable features such as its explainability, inter-operability between applications/humans and reasoning on conclusions [3].

A. Use Case: a Virtual Assistant

Thales is developing a virtual assistant exploiting symbolic AI in order to assist the aircrew. This assistant is the targeted application and would be deployed in a tablet-type platform with time response/memory constraints (see Fig. 1).

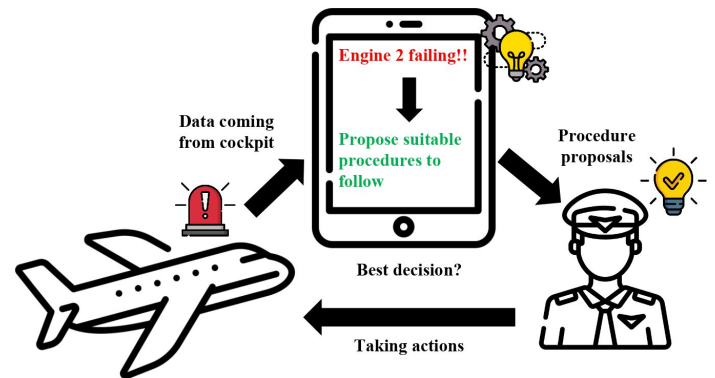


Figure 1: Schema of a use case example (icons from Flaticon.com)

B. Ontologies and Reasoners

We are dealing with knowledge bases in ontology format and these ontologies are formalised using the Web Ontology Language (OWL) based on the Description Logics (DL). A case in point, an airplane ontology consists of two regular sets:

- A TBox (terminology box), which is the skeleton of the domain of aircraft, describing it using concepts and relations, e.g. an airplane has two wings and wheels,
- An ABox (assertion box), which is the base of individuals related to our domain, e.g. an Airbus A380 or Boeing 738 aircraft.

A DL reasoner is used to deal with OWL ontologies via deductive reasoning to operate tasks such as satisfiability, consistency, and classification [4]. Its role is similar to inference systems which deduce logical conclusions from axioms [5].

Symbolic AI using ontologies and reasoners was not initially thought for embedded devices. First, [6] and [7] indicate that the world of embedded systems such as mobiles or constrained devices imply a sheer number of constraints regarding the deployment of a reasoner, knowing that many state of the art reasoners are too resource-intensive to be introduced on such systems. Usually, embedded systems exhibit low memory capacities, power, or strict time constraints. Second, in this respect, several studies show up that those reasoning technologies suffer from long and unpredictable execution times, usually saturating memory capacities of usual computers. A meaningful case exists in [8] where non-deterministic execution times are observed, and some reasoners are not adapted to scan large ontologies (> 1, 000 axioms).

Accordingly, the objective of this work is to focus on:

- Optimisations for hardware architectures regarding time/space envelope;
- Bounding reasoning runtime and memory footprint, and providing guarantees on its Worst-Case Execution Time.

C. Contributions

In this paper, we present some initial contributions of this work. More precisely, thanks to a wide exploration of the Description Logics (DL), Web Ontology Language (OWL), reasoners and optimisations thereof, we provide a categorisation of optimisation families. Then, we describe and analyse preliminary results from experimentations.

II. SCIENTIFIC POSITIONNING AND MAJOR MILESTONES

The main objective is to explore optimisation methods for execution of reasoners on ontologies to improve reasoning time and memory space management. The chosen approach consists of 5 stages:

1. Establish a state of the art mainly concerning the optimised reasoning algorithms and their implementation on hardware, and classify the various optimisations;
2. Find solutions and implement on toy cases such as LUBM benchmark [9];
3. Improve existing reasoner algorithms;

4. Develop efficient mapping on new architectures such as Graphics Processing Unit (GPU);
5. Implement the solution on the real case of virtual assistant for validation.

We currently consider various ontologies from multiple domains to benchmark implementations, tackling the following questions:

1. Which reasoning tasks (e.g. classification) are exploited?
2. Which level of expressiveness is needed?
3. What are the characteristics of the ontology?
4. What are the trade-off to be taken into account between expressiveness and complexity?
5. Which methodologies the reasoner is using?
6. Which approaches can be applied/tailored for use in embedded systems?
7. How to accelerate reasoning? By constraining the expressiveness, optimising reasoning methodologies, or both?

III. PRELIMINARY RESULTS

The real start step is to get a deep understanding of the language to deal with ontologies, OWL. OWL is based on DL, a logic language that is decidable. More precisely, there are two versions of OWL: OWL (2004) and OWL 2 (2009). Our research is focused on the second version, as it offers different sub-profiles (see Fig.2) encompassing more different levels of expressiveness (the level of details for an ontology to be modelled using symbols) and complexity to work with an ontology. [4] explicits the different profiles of OWL 2: OWL 2 EL is used over large ontologies with a lower complexity comparing to OWL 2 DL. OWL 2 QL targets SQL-based ontologies for performing query answering tasks. It detains a lower complexity too. And OWL 2 RL is the part that can be used to express knowledge using rules. It is used with rule-based systems such as expert systems.

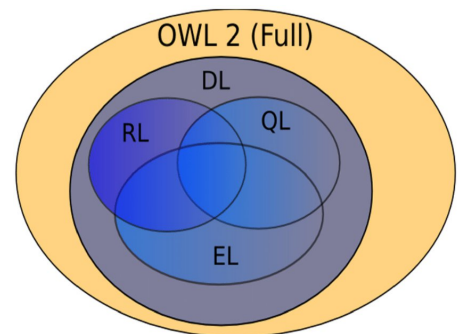


Figure 2: Organisation of the different flavours of OWL 2 [10]

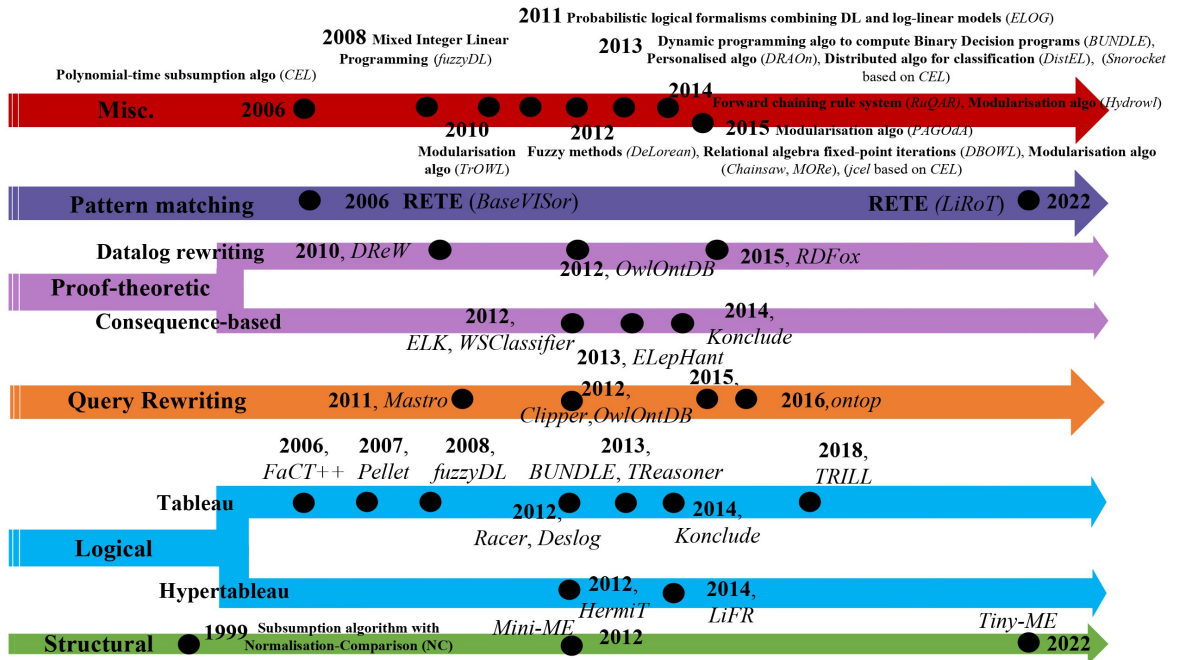


Figure 3: Timeline regarding reasoning approaches appearance with their associated reasoners from 1999 to 2022

A. Existing reasoners and optimisations

A review of existing reasoners and speed up techniques has been made with 2 distinct parts. A preliminary work of gathering well-known reasoners into a list has been established. A table has been created which paper reports 85 reasoners, mainly coming from [11], with their characteristics retrieved from a notable study of their respective paper(s) too. On top of that, 39 out of the 85 reasoners have been considered because of their popularity but also of their date of last update (at least 2012). Moreover, only official reasoners have been considered because of their recognition and community in literature, thus no prototypes were selected. These reasoners cover a large spectrum of methodologies as we show in Fig. 3. As a result, we have categorised in Fig. 4 the different optimisation families from studied existing reasoners in a logical order from those concerning low-level to software ones. This optimisation tilling enlightens which techniques could be useful for our use case.

B. Test bench

A preliminary work was made to compare several reasoner performances on a toy case ontology. It consists of 9, 228 axioms, 1, 459 individuals (instances), and has an expressiveness corresponding to the the DL family $ALCROIF(D)$ ([12] shows the nomenclature of DL families). *Fact++*, *HermiT*, and *Pellet*, very well-known reasoners (informations on them in [11]), have been tested with this ontology and the main reasoning task was classification which role is to build the hierarchy order among classes (or concepts).

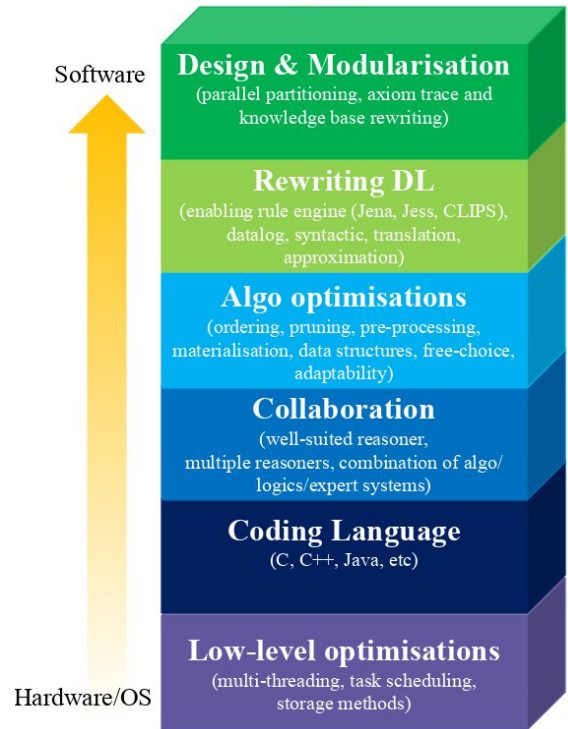


Figure 4: Categorisation of different optimisation families of existing reasoners

Fig. 5 shows variable runtime for each reasoner. The well-known user interface tool Protégé, mainly used for ontology implementation, was used to perform the tests. From [11], we can deduce that these reasoners use different kind of reasoning approaches namely algorithms. This could naturally explain why we have different notable reasoning time from a reasoner to another one for the same ontology. So then, this is important to study this by investigating ontology features impacting reasoner performances.

The idea is to find the best fit between ontologies and a given reasoner. This is the next step of this test bench work.

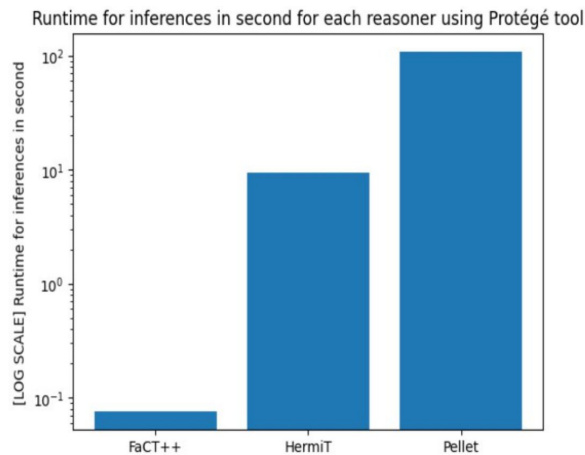


Figure 5: Classification inference runtime in second for each reasoner using Protégé tool

In a second step, we are focusing on implementing test benches to eschew the links between expressiveness and complexity of different kind of reasoners against ontologies. Moreover, found optimisations will help to make improvements mainly regarding time and space metrics. To assess the performance of a reasoner and understand its functioning, we are generating a set of ontologies (barely 3,000), ensuring that at least all of OWL 2 profiles are represented with different size. The goal is to identify which ontology metrics impact the more the reasoning process and importantly at which level. An additional goal is to find bottlenecks by performing a profiling process in order to identify code obstructions and find ways of improvement towards our problematic.

C. Current work

We are currently tackling the following tasks:

- Build test benches using a sheer number of reasoners from different families of algorithm and optimisations;
- Understand and explain why measurement of the time/space envelope is different between reasoners against ontologies.

IV. CONCLUSION AND FUTURE DIRECTIONS

Our literature review has shown that all reasoners have their own features and restrictions. We presented the first steps towards providing a clear evaluation phase in an embedded system context.

Our future works include investigating how embedded architectures and hardware acceleration techniques can go in pair with optimisation of algorithms. Our ultimate goal is to implement a solution in the real case of a virtual assistant.

REFERENCES

- [1] F. K. Bapp and J. Becker (2018), *Advances in Avionic Platforms: Multi-core Systems*, Book: *Advances in Aeronautical Informatics, Technologies Towards Flight 4.0*, Springer
- [2] U. Durak (2018), *Flight 4.0: The Changing Technology Landscape of Aeronautics*, Book: *Advances in Aeronautical Informatics, Technologies Towards Flight 4.0*, Springer
- [3] E. Ilkou and M. Koutraki (2020), *Symbolic Vs Sub-symbolic AI Methods: Friends or Enemies?*, Conference: *CSSA'20: Workshop on Combining Symbolic and Sub-Symbolic Methods and their Applications co-located with CIKM2020*
- [4] C. Maria Keet (2020), Book: *An Introduction to Ontology Engineering*
- [5] K. Abicht (2023), *OWL reasoners still useable in 2023*
- [6] T. Kleemann (2006), *Towards Mobile Reasoning*
- [7] A. Bento, L. Médini, K. Singh, F. Laforest (2022), *Do Arduinos Dream of Efficient Reasoners?* In: *The Semantic Web*. Cham
- [8] F. De-Grancey and A. Audouy (2023), *Towards the deployment of Knowledge Based Systems in Safety Critical Systems*, *The Semantic Web: ESWC 2023 Satellite Events*
- [9] J. Heflin et al. (2005), *LUBM: a benchmark for OWL knowledge base systems*, Journal: *Journal of Web Semantics*
- [10] P. Bonte et al. (2022), *Bridging the gap between expressivity and efficiency in stream reasoning: a structural caching approach for IoT streams*, Journal: *Knowledge and Information Systems*
- [11] The University of Manchester (last update: 2018), website: <http://owl.cs.manchester.ac.uk/tools/list-of-reasoners/>
- [12] Wikipedia (last update date: 2023), *Description logic*, https://en.wikipedia.org/wiki/Description_logic#Nomenclature

How to efficiently handle real world ECU traffic in MICROSAR Adaptive

1st Johannes Hötzer

Vector Informatik GmbH

Stuttgart, Germany

johannes.hoetzer@vector.com

2nd Christoph Schmutzler

CARIAD SE

Ingolstadt, Germany

christoph.schmutzler@cariad.technology

3rd Julian Maidl

Vector Informatik GmbH

Stuttgart, Germany

julian.maidl@vector.com

4th Daniel Schmid

Vector Informatik GmbH

Stuttgart, Germany

daniel.schmid@vector.com

5th Patrick Markl

Vector Informatik GmbH

Stuttgart, Germany

patrick.markl@vector.com

6th Daniel Dausend

Vector Informatik GmbH

Stuttgart, Germany

daniel.dausend@vector.com

7th Matthias Traub

Vector Informatik GmbH

Stuttgart, Germany

matthias.traub@vector.com

Abstract—The software-defined vehicle (SDV) enables continuously updated software defined features. This increasing demand of new features requires a high-performance computing platform (HPC) and a hierarchical system structure based on a zonal-oriented architecture approach. The communication of the different types of ECUs (sensor/actuators, zonals, HPCs) are realized with various communication technologies like LIN, CAN and Ethernet. The processing of the messages in these kinds of heterogeneous networks is different: In the signal world, data is commonly processed cyclically, the POSIX-based HPC world is often event driven. The different message sizes and the processing behavior led to the situation that even extremely powerful multi-core ECUs are not capable to receive all data via Ethernet and processes it in time. In this paper we discuss the backgrounds and show measures to efficiently utilize POSIX based systems. We start from currently used message size distributions and send frequencies of real ECUs and optimize the system. We also show general limitations of POSIX systems and especially microkernels. For the measurements we use an Renesas R-Car H3 Board running QNX and the MICROSAR Adaptive middleware. By systematically optimizing we reduced the CPU usage from 81.6% to 24.9% in total.

Index Terms—AUTOSAR, SOME/IP, POSIX, Performance

I. INTRODUCTION

The AUTomotive Open System ARchitecture (AUTOSAR) consortium [1] is based on a long-standing and well established alliance of car manufacturers, automotive software, hardware and semiconductor suppliers and others. Main goal of AUTOSAR is to define an open and standardized software architecture for automotive electronic control units (ECUs) including a common specification, methodology and configuration. The Classic AUTOSAR standard was first released in 2004 and focused on embedded RTOS systems and microcontrollers. The rising importance of microprocessor-based ECUs with a POSIX-based systems was addressed by the Adaptive AUTOSAR standard, first released in 2016. Adaptive AUTOSAR also focus on Ethernet-based in-vehicle communication. Today, different companies provide Classic and Adaptive AUTOSAR implementations. MICROSAR Adaptive [2] is the Adaptive AUTOSAR solution from Vector Informatik GmbH.

Ethernet has now been in use in vehicles for over 16 years: BMW first introduced Ethernet based on a 100BASE-TX physical layer for flashing of ECUs in 2008 [11]. Later, Broadcom development a new Ethernet physical layer – called BroadR-Reach [9], later adopted as Open Alliance BroadR-Reach (OABR) – that allows for 100 Mbit/s bidirectional communication over a single unshielded twisted pair cable. In 2013, BMW piloted the Open Alliance BroadR-Reach (OABR) in a new generation of surround view system, a birds-view display of the vehicle. In 2015, OABR (and later 100BASE-T1) was first used for "generalized" in-vehicle Ethernet-communication, i.e., not only for specialized audio/video traffic, but also for the exchange of status and control data between several Ethernet-ECUs [12]. Since 2015, 100BASE-T1 and new multi-gigabit Ethernet standards have been adopted by other automotive OEMs [13].

Without the sake for completeness, the use of Ethernet for in-vehicle communication can be divided into different use cases and domains, starting with high-bandwidth applications first and moving towards lower bandwidth requirements:

- diagnosis and flashing of ECUs
- transmission of raw audio, video or sensor data
- communication between in-vehicle ECUs and telematic control unit (TCU)
- communication of processed sensor data for driver assistance systems (e.g., fused object data around the vehicle)
- communication of control loop data between Ethernet ECUs (e.g., using detected objects for head-up-displays)
- communication and tunneling of legacy data, i.e., communication between CAN and Ethernet ECUs via a gateway

Whilst high-bandwidth applications impose their own challenges, in this paper we focus on the last two use cases. This communication is usually of small payload but can be of high frequency. Formerly, this traffic was mostly handled by specialized Embedded ECUs. However, with the increasing trend to introduce POSIX-based computing and integration platforms that aim at replacing smaller ECUs, fine-grained

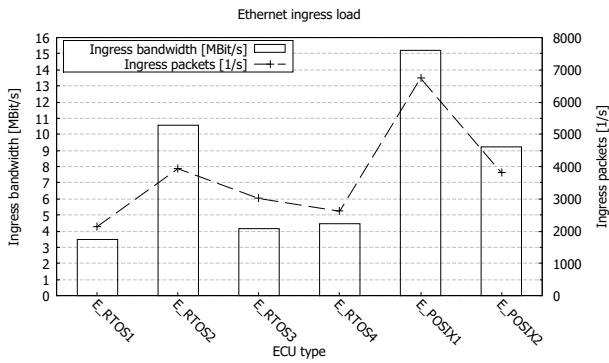


Fig. 1: Ingoing load of different RTOS and POSIX ECUs

Ethernet traffic gets increasingly relevant for POSIX-based systems. Even though this high-frequency traffic poses no problems regarding bandwidth of Ethernet-links, it can still overload POSIX-based systems, depending on the strategy and implementation of message and traffic shape handling, due to the high packet count per second. The problem of high frequency ingoing-frames is to some extent independent of the underlying protocol and can be generalized: whether data is sent directly, e.g., via UDP/IP, PDUs, SOME/IP, DDS, or communication is handled via method calls—at some point, the frequency of received frames may get too high to process it. Of course, the transmission rate can and should be limited by appropriate measures directly on the sender—limiting the outgoing rate also reduces the ingoing load of all receiving ECUs. This can be done, e.g., through debouncing by the function that produces data on the sender, or by the used communication protocol. However, because the required minimum update rate of data is highly function dependent and might not be increased, and partitioning of more and more functions in POSIX computing platforms inherently leads to an increase of communication relationships and, therefore, incoming traffic, optimizing receive performance of ECUs is also crucial.

II. ANALYSIS OF ETHERNET COMMUNICATION

In this section, we look at real-world in-vehicle control loop Ethernet communication, to evaluate the typical bandwidth and packet count an ECU needs to handle. The analysis is based on traces of Ethernet-connected ECUs of a current electrical/electronic architecture. Due to its strongly application dependent nature, control loop traffic is typically processed by upper layer software. This means that frames and included data first needs to be processed by the software networking stack and are then passed to upper-layered software components. In contrast, processing of video and audio traffic can often be hardware accelerated or is at least handled by a specialized implementation. The shown measurements do not include raw video, audio or sensor data, nor traffic between zonal ECUs (Ethernet backbone traffic), which are the main drivers for new and upcoming multi-gigabit standards [14]. Fig. 1 shows ingoing traffic load of different RTOS and POSIX ECUs. The measurements reflect a typical driving scenario. We observe that the required bandwidth is far below the line

speed of a typical 100 Mbit/s link. ECU E_POSIX1 has the highest ingoing bandwidth of approx. 15 Mbit/s and also the highest ingoing frame count of approx. 6700 frames/s. The payload size of captured traffic shows the prevalence of small grained messages that are characteristic for "traditional" in-vehicle communication: 60% of payload is ≤ 64 byte, only 8% is > 1400 byte. Overhead, e.g., due to encapsulation in a SOME/IP or UDP/IP frame, is not included. Other ECUs have fewer communication relationships and hence in general less ingoing bandwidth. However, the frame count is still high—this motivates an optimized receive implementation that can handle processing of $>> 1000$ frames/s.

III. BENCHMARK SETUP

To run the performance tests we used an Renesas R-CAR H3 board and a Linux workstation, connected by a 1 Gbit/s Ethernet link. On the R-CAR board we test three different setups: Ubuntu 22.04.4 LTS with a 6.5.0-1016-azure kernel, QNX 7.1 without any additional modification or tuning of the drivers and, lastly, MICROSAR Adaptive version r9.24.4-1, running under QNX. For QNX we used the official board support package provided by Blackberry and run the io-pkt network daemon. For all tests we used the Linux workstation as sender and the Renesas R-CAR as receiving hardware. Amongst other criteria, usage of QNX is relevant for safety-critical use cases, because it is certified to meet standards such as ISO 26262.

IV. RESULTS

In the first part the possible limits for different payload sizes for UDP are determined using an open source benchmark tool named iperf3. Based on the analysis in section II we show in the second part how AUTOSAR and the MICROSAR Adaptive middleware help to efficiently transfer messages from a real world use-case.

A. Benchmark with iperf3

Besides the theoretical performance of 1 Gbit/s we used the iperf3 benchmark [3] to derive the maximal possible throughput using UDP on the ECU described in section III. Iperf3 is design on top of `sendmsg()` and `recvmsg()` APIs [6], leading to a syscall for each message. In order to improve the syscall overhead we extended the framework to use `sendmmsg()` [5] and `recvmmsg()` [4] APIs provided by QNX and Linux (Note the addition "m" in the function names). Otherwise, QNX would not be able to reach more than 50% of the possible payload throughput on the tested hardware. This optimized APIs allow to handle multiple messages with one syscall using a vector of messages. The receive call was implemented to block until the entire receive vector of messages (`msgvec`) is full before returning. The measurements where executed 10 times by running: `./iperf3-patched -c <receiver_ip> -Z -l <packet_size> -b 1G/<msgvec_size> -udp` with the packet size varying from 1 byte to 1400 byte and the number of received messages per syscall (`msgvec_size`)

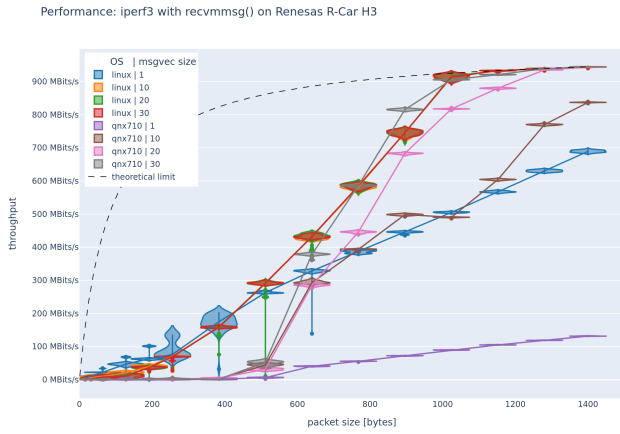


Fig. 2: Throughput of iperf3 on Linux and QNX depending on the syscall

form 1, 10, 20 to 30. The results are plotted for Linux and QNX in fig. 2 showing the average throughput over the message size and the distribution with violins. As black dashed line, the maximal throughput with respect to the payload is shown. For small packages the throughput is low and increases with the payload sizes. This behavior is expected as the header to payload overhead reduces with larger messages. For one Byte message size the throughput is between 3.9 Mbit/s and 4.9 Mbit/s for Linux and between 31.2 kbit/s and 305.1 kbit/s for QNX, depending on the msgvec size. The throughput for Linux increases faster as for QNX. This is partly driven by the overhead of the syscall, which is larger under QNX due to the microkernel architecture requiring a network daemon to handle the network traffic. With increasing msgvec size, the syscall overhead reduces, leading to a better throughput compared to smaller values. However, for larger values the effect is less pronounced. For only one received message per syscall the link can not be fully utilized and reaches only 688.9 Mbit/s for Linux and 133.3 Mbit/s for QNX, indicating that improved syscalls are fundamentally necessary for automotive ECUs. It should be noted that under Linux `recvmsg` is also able to utilize the link as the overhead of this syscall is smaller compared to `recvmsg` with only a msgvec size of one. All measurements show high CPU load, as one core is fully utilized. This also leads to a high number of lost frames for small msgvec and small message sizes.

B. Performance middleware

Building on the insights of the iperf3 measurements and the analysis in section II we created an artificial setup using SOME/IP events with a total receive rate of 12.3 Mbit/s which we optimize towards a better CPU usage. The number, size and periodicity of all events is depicted in table I and reflect the distribution of real ECUs. For the measurements we used a single receiving application which subscribed to all events. No events are sent from the application. In the MICROSAR Adaptive [2] middleware the SOME/IP daemon is sending

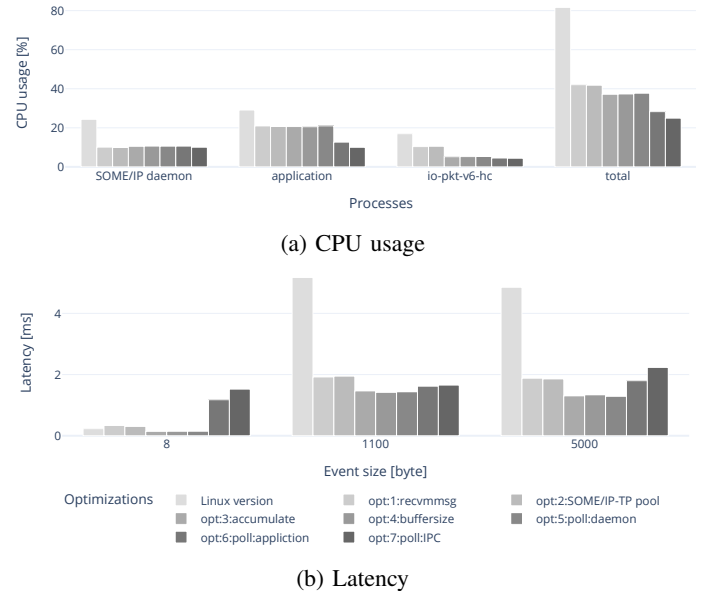


Fig. 3: Effect of the different optimizations of the MICROSAR Adaptive middleware on the CPU usage and latency.

and receiving the SOME/IP events and is dispatching it to the subscribed applications using a reactor pattern. With this design the same event needs not be transmitted multiple times to different receiving applications in the same ECU. A single port for all SOME/IP traffic is possible, which is often required for the communication with microcontroller-based systems to reduce the memory consumption. To forward the SOME/IP events from the daemon to the application process a shared memory-based ring buffer (SafeIPC) is used with an additional notification channel to wake up the receiver. To transmit the data over the network we used UDP. For events larger than an UDP frame we use the SOME/IP-TP protocol [7], which segments or combines the messages in the SOME/IP daemon. Per default a reactor thread is running doing the asynchronous reception of messages at the application. The application itself runs in an own thread decoupled from the receiving part. This design is chosen to being reactive in the application and to minimize the configuration effort. As hardware we used the setup described in section III.

In the following we show how the middleware can be optimized under QNX to efficiently handle the events described in table I. As main goal, we optimize the CPU usage with the additional constraint to not exceed a latency of >5 ms. In this case latency is defined from the moment the event is received in the network driver till it reaches the application code. All results are depicted in fig. 3 for selected message sizes and the involved processes. We started from a Linux setting using `recvmsg` and a socket buffer size of 3 MB. Using the default socket buffer size results in a high rate of the package loss, leading to meaningless results. In the first optimization (**opt:1:recvmsg**) we switched the processing mode of the SOME/IP daemon which is handling the sockets to `recvmsg` with a msgvec size of 30. With this change the CPU usage decreased from 81.6 % to 24.9 % in total. This

change impacts both the CPU usage of the io-pkt network daemon as well as the SOME/IP daemon. The application load also decreases as the events can be forwarded in chunks of the msgvec size. The latency decreases for all message size as the processes are not delayed due to the syscall and context switch overhead. In the second optimization (**opt:2:SOME/IP-TP pool**) we activate a memory pool to use pre allocate memory for SOME/IP-TP messages [7] instead of dynamic allocations. This improved slightly the latency for larger events. As the number of SOME/IP-TP messages in this scenario is low, the impact in this case is small. The third optimization (**opt:3:accumulate**) is an AUTOSAR feature to accumulate smaller messages in a larger message on the sender [8]. This reduces the CPU usage of the io-pkt as less packages are arriving, however increased it a bit in the SOME/IP daemon as the accumulated messages need to be unpacked and dispatched to the respective applications. Overall the CPU usage reduced by 3.7%. For the accumulation we used a timeout of 5 ms on the sender. This reduces the number of messages but increases the overall latency as messages are not directly sent. This feature is mainly beneficial for small events to reduce the CPU load especially on the receiver side. This feature is the only one which also effects the sender side and hence cannot always easily be applied or requires a gateway ECU which is connected to the legacy ECUs. Besides the CPU usage, also the latency reduces for all sizes as the io-pkt is reacting faster. In the fourth step (**opt:4:bufferize**) we increased the ring buffer size from 128 kB to 1 MB between the SOME/IP daemon and the application process to reduce the additional buffering, retry and segmentation overhead for larger messages in the daemon. As we only have one receiver, the same priorities and the overall load of the system is moderate, processes are scheduled fast and hence the ring buffer is not overflowing, hence we see no effect of this optimization in this setup. As default, the middleware runs in a so-called event driven mode building on top of a reactor pattern. For Linux we use `epoll` in the reactor, under QNX an own more efficient variant building on QNX primitives is used. To reduce the scheduling overhead and wakeup of the processes for each message, in the following steps we activated a polling behavior to process the messages in junks, trading latency for CPU usage. In **opt:5:poll:daemon** we activated the polling in the SOME/IP daemon with a period of 2 ms, reading all messages from the socket. In **opt:6:poll:application** we removed the receive handler from the application, meaning we are no longer being notified about new messages via a callback function running the `ReceiveHandler` thread. Instead of calling the `getNewSamples()` API from this asynchronous callback function, we poll all of them with a cycle time of 2 ms. Finally, in **opt:7:poll:IPC** we also turned off the async reactor thread in the application processes and poll the IPC channels in a 2 ms cycle, all in the context of the application thread, leading to no more context switches within the application. With all polling optimization, the CPU usage of the middleware reduces from 37.3% down to 24.9%. As expected, with turning on the polling optimizations, the latency increased

but remained below the 5 ms constraint. Note, in this work we have not tuned the operating system and the scheduler, which could further improve the latency. Also, the processing mode depends on the application design, if it's a cyclically running application, polling is more beneficial compared to an application reacting on sporadic events, e.g. from the driver. In general, the shown optimizations and their effect strongly depend on the message size, the frequency, priority of the processes and the overall system load and should always be verified for the respective project. Besides the derived use-case depicted in table I, we scaled the final optimized version up to a factor of 5 until we reached the system limit. The system is then no longer able to process the events in the application. With this scaled setup 63.2 Mbit/s and more than 31 000 event/s are handled by the middleware. The total CPU usage increases to 75% and the latency for 8 Byte increased to 5 ms. This scaling shows the upper limit for the given message size distribution, operating system, and hardware.

V. CONCLUSION

Based on the open-source benchmark `iperf3` we showed the limits for receiving UDP data on a real ECU. Especially small messages lead to a high overhead of the system, if operated in event driven mode, resulting in high CPU usage, low throughput, and high amount of message loss. Even for a powerful ECU the usage of small messages results in a low maximum throughput of 12.3 Mbit/s of UDP data. Building on analysis of real-world network traffic, we showed how AUTOSAR Adaptive with SOME/IP as well as the MICROSAR Adaptive middleware enable an efficient usage with respect to CPU usage, still able to handle the traffic and keeping the latency below 5 ms. AUTOSAR and SOME/IP features are used to accumulate small messages as well as segment larger ones. By changing the event driven processing of the reactor pattern to a periodic one by cyclically polling the API, we could significantly reduce CPU usage. Also, the change of the receive API to `recvmsg` strongly reduced the number of syscalls. Under QNX, due to its microkernel architecture, this also means similarly less context switches to the network daemon. The polling-based approach shows a significant effect on CPU usage, whilst still meeting latency requirements. All optimizations show that for small messages the overhead of syscalls involving other daemons (i.e., io-pkt), scheduling and context switches is quite high and should be as low as possible. Further improvements are possible by tuning the operating system, priorities and scheduler. We also scaled the scenario up to a factor of 5 until we reached the system limit.

In general building on asynchronous frameworks and other middlewares like DDS or GRPC will result in comparable issues for fine-grained, high-frequency traffic and need to be tuned in a similar way. MICROSAR Adaptive, AUTOSAR and SOME/IP provided means to tune the middleware to efficiently utilize the ECU for the respective use-cases required for a SDV.

REFERENCES

- [1] AUTOSAR, <https://www.autosar.org/>, accessed 19.03.2024
- [2] MICROSAR Adaptive, <https://www.vector.com/de/de/produkte/produkte-a-z/embedded-software/adaptive-microsar/>, accessed 19.03.2024
- [3] iperf3, <https://iperf.fr/>, accessed 19.03.2024
- [4] recvmmsg, <https://man7.org/linux/man-pages/man2/recvmmsg.2.html>, accessed 19.03.2024
- [5] sendmmsg, <https://man7.org/linux/man-pages/man2/sendmmsg.2.html>, accessed 19.03.2024
- [6] The Open Group Base Specifications Issue 7, 2018 Edition, IEEE Std 1003.1-2017 (Revision of IEEE Std 1003.1-2008)
- [7] AUTOSAR, SOME/IP Protocol Specification, AUTOSAR Foundation R22-11, 2022
- [8] AUTOSAR, Specification of Communication Management, AUTOSAR Adaptive Platform R22-11, 2022
- [9] Broadcom Corporation, "BroadR-Reach© Physical Layer Transceiver Specification For Automotive Applications", March 2012.
- [10] "IEEE Standard for Ethernet Amendment 1: Physical Layer Specifications and Management Parameters for 100 Mb/s Operation over a Single Balanced Twisted Pair Cable (100BASE-T1)," IEEE Std 802.3bw-2015, 7 March 2016, doi: 10.1109/IEEESTD.2016.7433918.
- [11] A. Maier, "Ten+ years automotive Ethernet" Automotive Ethernet Congress 2015, March 2015.
- [12] K. Matheus, "Evolution of Ethernet-based automotive networks: faster and cheaper." 8th Ethernet&IP@Automotive Technology day, Oct. 2018.
- [13] T. Koenigseder, "How did the automotive Ethernet vision of 15 years ago turn out today?" Automotive Ethernet Congress 2023, March 2023.
- [14] K. Matheus, "Status of automotive high-speed standardization for sensors and displays." Automotive Ethernet Congress 2024, March 2024.

APPENDIX

TABLE I: SOME/IP events and send periods

event sizes	# events	period ms	event/s	Byte/s
1	3	10	300	300
4	3	10	300	1200
8	9	10	900	7200
32	27	20	1350	43200
64	39	20	1950	124800
100	18	40	450	45000
200	12	40	300	60000
400	6	40	150	60000
600	8	40	200	120000
1100	8	40	200	220000
2000	4	80	50	100000
4000	4	80	50	200000
5000	12	100	120	600000
			6320	1581700

Optimal Onboard Controls for Multiple Electrified Automotive Applications

Dr. Mariano SANS*

*Vitesco Technologies France, Toulouse
(Tel: +33 5 8299.8900; e-mail: mariano.sans@vitesco.com)

Abstract: Intensive developments in new electrified automotive applications lead more and more to high dimensional inputs/outputs and multi-criteria optimization challenges. These applications include new and complex components as Power Electronics, Bi-directional Chargers, Fuel Cells, etc., in the purpose of Energy Management, Battery Management, Thermal Management, etc., not forgetting constraints induced by the Life Cycle Analysis (LCA) or Life Energy Analysis (LEA) for better sustainable industrial design. State-of-the-art solutions in onboard computers of vehicles usually use classical control strategies like PID, digital compensators, or some optimal control strategies like ECMS (Equivalent Consumption Minimization Strategy), but not performant enough face to high multi-dimension systems. Some new and alternative Model-based Optimal Control technics are described in this paper, based on the Pontryagin Maximum Principle (PMP) applied on predictive scenarios, with innovative adaptations to consider physical non-linearities, multi-level constraints, saturations on outputs or states, frequency limitations, self-calibration of tuning parameters, self-adaptation of internal models, and real-time triggering of strategies according to disturbances.

Several applications are presented, giving promising performances, around 5% to 10% of energy savings, using potentially real-time compliant algorithms, able to include some global long-term optimization of Life Cycle Engineering (LCE) criteria as potential prospects in such innovation projects developments.

1. INTRODUCTION

Targeting new challenges as energy savings, CO₂ emissions reduction, and Life Cycle Engineering for contribution to limit global climate changes makes important to reconsider the way to design appropriate algorithms that must react in real-time conditions into individual or collective automobile transportation systems. Electrified vehicles have got additional degrees of freedom (hybridization with additional electrical motors, additional power sources like batteries, super caps or fuel cells, autonomous cars...) that must be manipulated in real driving conditions for mixed comfort, safety, and energy criteria to be optimized in more severe conditions (increase of voltage, new safety conditions, adaptation to traffic, etc.).

New issues become to be an enlarged optimization problem of a complex dynamic system to lead from one initial state to another final state under severe constraints.

From this definition, from actual knowledge in modelling physics of vehicle systems and sub-systems, and from mathematical definition of the criteria to optimize, advanced algorithms can be considered, based on specific methods to solve equations, and to minimize integral criteria on given scenarios like standard cycles and real driving records.

Optimal control design can be split in different steps:

- definition of a **representative model**, i.e. a set of equations describing the main physical behavior to be considered.
- definition of the **time scenario** on which the expected criteria have to be optimized, with the boundary conditions (initial and final targeted states).
- definition of the **criteria to optimize** (maximize and/or minimize), as a cumulated sum of calculated variables.
- definition of the **constraints to be respected** (lower or upper limits, in values or gradients).
- **tuning of calibration parameters**, to reach the boundaries conditions.

This general scheme process can be summarized as below:

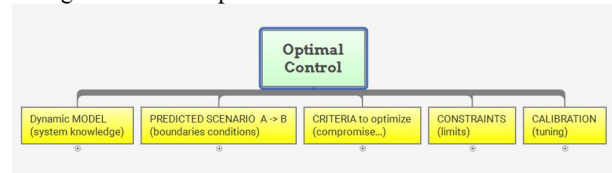


Figure 1: Optimal Control general scheme

The purpose of this paper is to describe and promote different improvements in the PMP algorithms to provide the optimal control strategy of a vehicle to minimize multi-criteria during different driving scenarios. Typical applications focus on electrified vehicles, in different use-cases, with description

along following chapters of the main strategy adaptations enabling to solve or compensate different kind of complexities, like non-linearities, multi-level constraints, saturations in inputs or outputs, frequency limitations, self-adaptive parametrization, adaptive triggering under disturbances, or integer combinational optimization for high-dimensional systems based on Linear Programming strategies.

2. PMP model-based algorithms

2.1 Basic equations

The Pontryagin Maximum Principle (PMP) is clearly described in literature as in [1], [2], [3], [4], [5] for theoretical presentations, and [6], [7], [8], [9] for some specific Automotive applications, showing the pertinence and efficiency of such mathematical approach to get the optimal solutions for complex systems to be controlled in an optimal way, even under rigorous constraints.

Based on equations of dynamical behavior to predict, criteria to estimate, and constraints to check and satisfy, the PMP method provides among the best operating strategies by minimization of the Hamiltonian function calculated with respect to the admissible control input values. Manipulation of equations in basic PMP algorithm are summarized below (see [1], [3]):

$$\text{Considering the state-equation: } \dot{x} = f(x, u) \quad (1)$$

with x = state vector, u = input vector, f = state equation and criteria to be minimized:

$$J = \int_{t_0}^{t_f} g(x, u) \cdot dt + \phi(x_f) \quad (2)$$

with $g(\cdot)$ = instantaneous criteria equation, $\phi(\cdot)$ = final criteria penalty, all expressions being non-linear in general,

If an optimal strategy $u^*(t)$ exists on the $[t_0, t_f]$ time interval, then following optimality conditions must be respected:

Co-states λ exist, as:

$$H = g(x, u) + \lambda^T \cdot f(x, u) \quad (3)$$

$$\text{with } \frac{d\lambda(t)}{dt} = -\frac{\partial H}{\partial x} \quad (4)$$

leading to the optimal control strategy u^* solution as:

$$u^*(t) = \text{Arg}_{\min} H(u(t)) \quad (5)$$

for given $x(t)$ and $\lambda(t)$ at each instant.

Additionally, co-states λ must converge to the given gradient expression of the final criteria penalty ϕ :

$$\lim_{t \rightarrow t_f} \lambda(t) = -\frac{\partial \phi(t)}{\partial x_f} \quad (6)$$

From this basis, different discussions may arise, as developed below.

2.2 Changing Software configuration

We can then consider the PMP as an algorithm including and simulating inside the *direct* Model of the system to be optimized, i.e. calculating the outputs vs. the inputs by the means of the $f(\cdot)$ and $g(\cdot)$ equations. We can call this specificity as a “MIS”, i.e. having a “Model in the Software”,

which is not the case at all in usual algorithms in automotive applications, that either calculate the *reverse* of the system model (calculating the inputs according to the required outputs from ‘inverted’ algebraic formula, which is not always accurate enough), or don’t use any dynamic model at all, only having parameters to be tuned according to the expected performances, like the PID approach. Only Model-based predictive controllers (MPC or MBPC) are also using direct models providing the predicted effects of control inputs considered in the optimization process. But optimization is only assumed from strategies like the PMP, as used in this purpose.

2.3 Complexity to cope with

PMP is a strong theory basis, and several concrete applications have proven its efficiency. But some issues may remain nevertheless, depending on the complexity of the global system and scenario on which to optimize it (see [2], [3]).

A first difficulty may reside in the solving of the Hamiltonian minimization itself, in case of strong non-linearities of discontinuities in the model, or high dimension in states with internal mutual influences, or due to significant model errors, variations in time or ageing effects, or because of constraints and saturations on inputs and outputs that make solving processes much more complex, etc.

Another difficulty may arise in the tuning of initial co-states values included in the Hamiltonian function, in the purpose to enforce the global boundary conditions expected on initial and final states of the system to be controlled. This tuning may become for instance too sensitive for too long scenarios like a complete standard automotive driving cycle as the “WLTC” or “WLTP” (Worldwide harmonized Light vehicles Test Cycle or -Test Procedure, 30 minutes long), or under unpredictable disturbances, etc.

Different solutions are then proposed in following chapters.

3. PMP Adaptations

3.1 Model adaptation to real system behavior

PMP strategy is a model-based strategy. It becomes then important to ensure the validity of the model and of the optimal strategies coming from it. In this purpose, an “observer” has been integrated in the real-time software, reacting from the comparison between real output measures and estimated values of the system variables. This solution is advantageously combined with the already present ‘direct’ model that the PMP is based on, as discussed previously.

These adaptations can be applied to both the states gradient equations $f(x, u)$ and/or to the criteria definition $g(x, u)$, with simple additive or subtractive terms that allow these models to converge and give a better accuracy to the PMP optimization.

It comes, in a simple version:

$$\dot{x} = f(x_{mes}, u^*) + K_f \cdot \epsilon_x \quad (7)$$

with $\epsilon_x = x_{mes} - \hat{x}_n$

x_{mes} = actual measurement of state value x
 \hat{x}_n = recurrent estimation of state value from gradient equation:

$$\hat{x}_n = \hat{x}_{n-1} + [f(\hat{x}_{n-1}, u^*) + K_f \cdot \epsilon_x]. dt \quad (8)$$

A similar method is applied to adapt the criteria expression $g(x, u)$, considering that in most cases the cumulated criteria can be measured (for instance, the total fuel or energy consumption). It comes:

$$\hat{J}_n = \hat{J}_{n+1} + g(\hat{x}_n, u^*). dt \quad (9)$$

leading to: $\hat{g} = g(\hat{x}_n, u^*) + K_g \cdot \epsilon_g$
with: $\epsilon_g = J_{mes} - \hat{J}_n \quad (10)$

Gains K_x and K_g main be tuned in a standard way, according to the dynamics of the system and the desired convergence response time for the adaptation, avoiding any instable mutual influence.

Therefore, the optimal strategy of input u^* is pushed closer to the real optimal solution.

3.2 Adaptive shooting method

A final criteria penalty is usually depending on the final states at the end of the optimization scenario (portion of a cycle, or portion of a real driving sequence). A typical case is the battery state of charge, to be got close to a specific final target, for different reasons or purposes (for instance reaching a minimal level before recharging to avoid ageing effects on the battery, or reaching a sufficient state of charge on a hybrid vehicle before entering a city zone implying pure electrical mode). The PMP theory imposes then the co-states to converge to the partial derivation of this penalty vs. the final states, i.e. $\frac{-\partial\phi}{\partial x_f}$.

But this dependency coefficient may usually depend on the past dynamic trajectory, and then must be adapted according to the criteria results for different tunings of co-states.

The final states x_f may be measured (or estimated), associated to the corresponding measured criteria J , and can be set from different tunings of the initial λ coefficients, giving an estimation of the gradient of the final penalty criteria:

$$\frac{\partial\phi}{\partial x_f} = \frac{\Delta J}{\Delta x_f} \quad (11)$$

for a given variation of the initial co-state $\Delta\lambda_{init}$.

It comes: $new \lambda_{init} = previous \lambda_{init} + K_\lambda \cdot \left(\lambda_f + \frac{\Delta J}{\Delta x_f} \right)$ (12)
until the final $\lambda_f(t_f)$ converges to the estimated gain $\left(\frac{-\Delta J}{\Delta x_f} \right)$.

This adaptation is tuned by the convergence gain K_λ , and may bring a quite good robustness to the PMP strategy, as shown in the following application of the energy consumption minimization on a serial-hybrid car NISSAN Serena on real driven tests (see Figure 2):

The vehicle is connected to a predictive “e-Horizon” system providing the *most probable path* in a several minutes horizon. That leads to minimize the global energy consumption by

recurrent retuning of the initial costate λ_o to target the penalty differential gain $\frac{-\partial\phi}{\partial x_f}$ assimilated to the estimated ratio $\frac{-\Delta J}{\Delta x_f}$ at the end of each prediction segment, as shown in Figure 2:

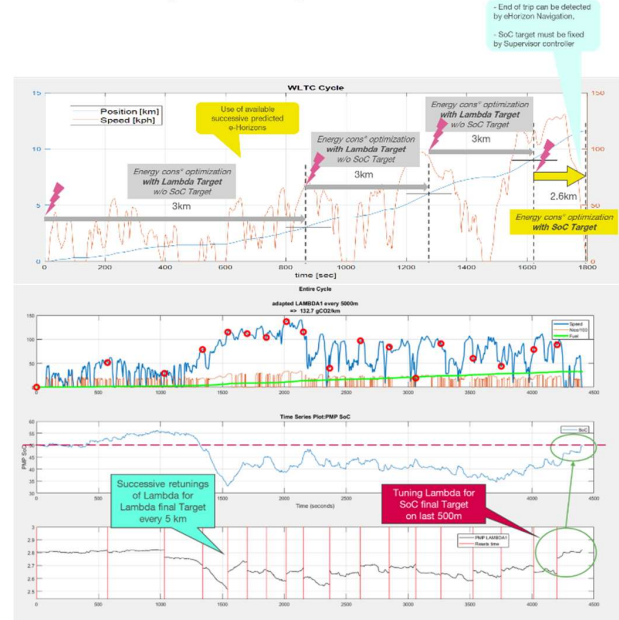


Figure 2: Adaptive Shoots application to real driving cycles

In complement, the effective final state target is set only on the last path horizon segment before reaching the final destination.

Simulations on WLTC cycle of a parallel hybrid passenger Honda “DHT” (“Dual Hybrid Transmission”) have proven that total energy amount got from this adaptation strategy is very close to the theoretical minimal value given by an off-line optimization like the Dynamic Programming (DP), also compared to “Cost Based” (ECMS type) and Reinforcement Learning (RL) strategies, as shown in following Figure 3:

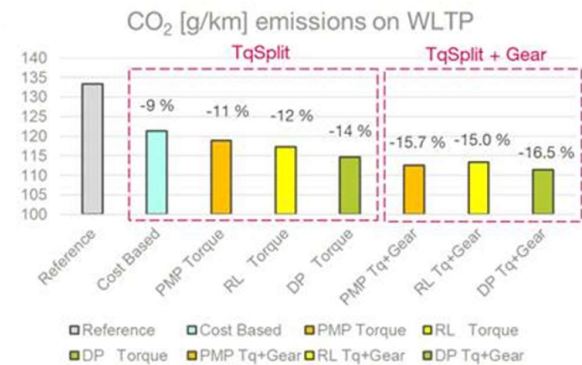


Figure 3 : CO2 emissions on WLTP on Honda DHT vehicle (ref. 134 g/km)

Two functions were implemented : a single “Torque Split” strategy and a combined “Torque Split + Gear” strategy, to get an iso-battery State of Charge (SoC) at the end of the cycle, by the means of torque repartition between combustion engine and electrical motor, combined with an optimal gear selection.

Several cycle portions have been used to control the co-states to successive targets as explained above.

As seen, an improvement of nearly 15% vs. reference has been reached, with around 6% better than the ECMS solution.

Considering that technics like RL need some long training phases, that are not necessary for the self-tuned PMP, that ECMS remain less robust to disturbances (see [9]), and that DP (based on Bellman principle – see [3]) are not real-time compliant, then we get significant advantages in robustness, efficiency and integrability with such on-board PMP solutions, as well on real driving cycles as on standard normative cycles.

3.3 Constraints management

The question of the constraints management in optimization processes is quite critical, in particular for limits (max or min) to be considered on states or outputs, leading to more complex algorithms able to consider *a priori* the possible effects to be observed *a posteriori* on the state outputs due to control inputs.

General equations of constraints on states may be expressed as:

$$h(x) \leq 0 \quad (13)$$

Since states or outputs are the effects (or consequences) of the inputs, the requested limitations on states or outputs signals need to manage indirect *a posteriori* checks, much more difficult as for direct *a priori* limitations on inputs signals.

Long and complex theories have been developed in literature (see [1], [2]), quite difficult to implement in real-time software.

A more “pragmatic” method is then proposed in this paper, based on a special adaptation of the Karush-Kuhn-&-Tucker (KKT) parameters associated to the equations of constraints, as according to the PMP theory, to keep the minimal value of the Hamiltonian function always inside the admissible domain of inputs fulfilling the requested constraints on outputs.

The idea is to define an admissible domain Δ_{ad} of control inputs, using a prediction of the states (noted \tilde{x}) on a given tuned horizon Δh (for instance: on several seconds to consider a temperature behavior). The prediction \tilde{x} is pre-calculated for all possible control inputs (for instance from nominal min to max values), and all non-admissible predicted outputs indicate non-admissible inputs, leading to the definition of a restricted admissible domain, respecting the constraints on predicted outputs:

$$\tilde{x} = x + f(x, u) \cdot \Delta h \quad (14)$$

Then the ‘predicted’ constraint equations become function also of u input :

$$h(\tilde{x}) = h(x + f(x, u) \cdot \Delta h) = \tilde{h}(x, u) \quad (15)$$

Then we get: $\Delta_{ad} = \{u \text{ as } \tilde{h}(x, u) \leq 0\}$

In this new domain Δ_{ad} , the PMP is applied without KKT coefficients, providing -according to the PMP theory- the optimal control solution minimizing the Hamiltonian function calculated inside the restricted admissible domain Δ_{ad} .

Then the KKT coefficients are introduced so that this optimal solution among the restricted admissible domain becomes the global optimal solution among the total possible domain, by a

mathematical translation of the Hamiltonian function thanks to the appropriate KKT coefficients.

In the case of inequality constraints on states, the KKT coefficients μ are introduced in the Hamiltonian function to get a complete Lagrangian function L as follows:

$$L = g(x, u) + \lambda^T \cdot f(x, u) + \mu^T \cdot \tilde{h}(x, u) \quad (16)$$

with $\tilde{h}(x, u) = (\tilde{x} - x_{max})$ for $\tilde{x} < x_{max}$
or $\tilde{h}(x, u) = (x_{min} - \tilde{x})$ for $\tilde{x} > x_{min}$

It becomes then possible to calculate the μ coefficients to push the optimal value u^* given by equation (5) to become at this instant the solution of the optimality condition $\frac{\partial L}{\partial u} = 0$, so that the global minimum of L coincides then with the optimal point given previously from the H function inside the restricted admissible domain.

For linear models where functions H and L are of class C2, a linear solving system can usually be used to get the appropriate μ coefficients, as detailed below:

since: $L = H + \mu^T \cdot \tilde{h}(x, u)$
then it comes: $\frac{\partial L}{\partial u} = \frac{\partial H}{\partial u} + \mu^T \cdot \frac{\partial \tilde{h}}{\partial u} = 0 \quad (17)$

For a single constraint, i.e. a scalar μ coefficient, it comes:

$$\mu^* = -\left(\frac{\partial H}{\partial u}\right) / \left(\frac{\partial \tilde{h}}{\partial u}\right) \quad \text{to be calculated for } u = u^*.$$

This calculation can be extended to the case of multi-dimensional systems, with n inputs \underline{u} and m constraints \underline{h} , that supposes to find m KKT coefficients $\underline{\mu}$, leading to a matrix equation resolution by a matrix pseudo-inversion, depending on if more inputs than constraints or vice-versa:

Equation (17) can be written as : $A \cdot \underline{\mu} + \underline{b} = 0$

with dimension of $A = n \times m$ and dimension of $\underline{b} = n \times 1$

Then it comes : $n > m \Rightarrow \underline{\mu}^* = -(A^T \cdot A)^{-1} \cdot A^T \cdot \underline{b}$

$$n < m \Rightarrow \underline{\mu}^* = -A^T (A \cdot A^T)^{-1} \cdot \underline{b}$$

$$n = m \Rightarrow \underline{\mu}^* = -A^{-1} \cdot \underline{b}$$

The μ^* coefficients when $\neq 0$ allow to saturate the corresponding constraints, equivalent to ‘counterforces’ preventing each constraint to be over-passed.

This transformation ensures to keep all the necessary PMP optimality conditions still valid in all the restricted admissible input domains under such constraints, as typically the condition on the costates gradient, as follows:

$$\frac{d\lambda}{dt} = -\frac{\partial L}{\partial x} = -\frac{\partial H}{\partial x} - \mu^{*T} \cdot \frac{\partial \tilde{h}}{\partial x} \quad (18)$$

As an example, following Figure 4 shows some simulated results of a PMP control with KKT coefficient management, applied to control the coolant temperature inside given limitations in the parallel hybrid vehicle Honda DHT, whiles battery state of charge is recharged, and global energy consumption is minimized on a more stringent driving cycle

with low speed and several stops, as here during a ‘‘Traffic for London’’ cycle:

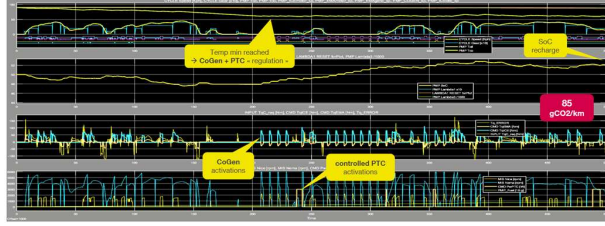


Figure 4: Optimal control of energy under temperature constraints in a parallel hybrid vehicle

The temperature depends on both fuel engine running point and on an electrical heater (‘‘PTC’’) plug in the coolant circuit. Multiple criteria (fuel consumption and battery state of charge monitoring) are managed at the same time the temperature state is maintained above a certain minimal threshold (around 60°C), by an automatic decision strategy of PTC activations and engine restarts, including co-generative phases (‘‘CoGen’’) where engine is just providing energy to the battery, when necessary, at vehicle stops. Total fuel consumption is nevertheless minimized to a low CO2 emissions value, around 85g/km for this passenger car, thanks to the respect of all PMP optimality conditions.

A scheme of the PMP adaptation for constraints management can be shown below in Figure 5, indicating the innovative parts (colored) added to the state-of-the-art algorithms (in grey):

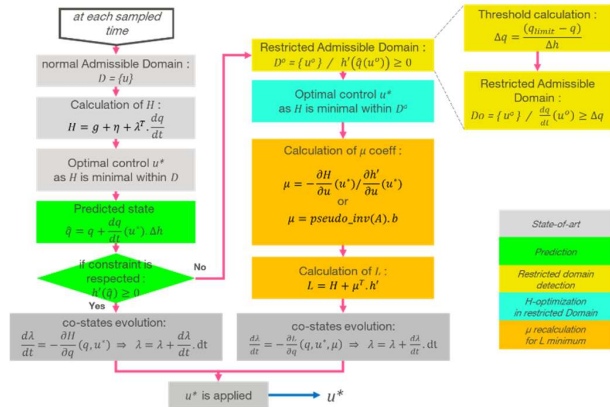


Figure 5: Constraints management in PMP strategy

3.4 High dimension systems management

In case of high dimensions in number of inputs (for instance N on/off actuators to be controlled together, as on a multiple-modules battery or fuel cell, or on multiple branches in onboard chargers for EV), the Hamiltonian function may become dis-continuous, and its minimization may lead to long numerical research of optimal values among a huge number of admissible values.

In this case an adaptation of Linear Programming strategy can be associated to the PMP, to solve it with good accuracy within fast calculations.

Considering many digital inputs (like on/off switches) $\{u_k = \pm 1\}$ for $k = 1, \dots, N$, for $N > 20$, the Hamiltonian cannot be calculated in all cases. A specific Mixed-Integer Linear Programming (MILP) algorithm is then associated to the PMP to find the minimal argument for H.

It comes : $u^* = Arg_{min}(H)$ = solution of a MILP function to find the minimal value among 2^N combinations.

This process then can be applied with a numerical derivation of the Hamiltonian function around the optimal control value u^* , to get the evolution of the co-states λ according to :

$$\frac{d\lambda}{dt} = -\frac{\partial H}{\partial x} /_{at u^*}$$

Real applications applied on a ‘‘Switched Battery’’ system used in an Electrical Vehicle have shown the capabilities of this technics to find the optimal combination of N actuators to optimize a dynamic system.

In this example, the PMP control is applied on a 36-modules battery, finding at each instant the optimal combination among the 2^{36} maximum possible combinations, to minimize the energy criterion based for instance on total ohmic losses while maintaining the equilibrium of modules states of charge, during discharge or recharge phases.

Following Figure 6 shows some simulation results where the different states of charge converge together after a dispersed initialization, with implicit permutations of successive activations, automatically performed by the PMP+MILP algorithms to minimize the global criteria on losses and ageing effects.

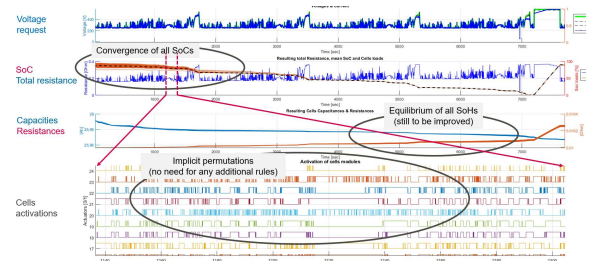


Figure 6: Control of a 36-modules Switched Battery

4. CONCLUSIONS

Coping with several non-linear phenomena in physical systems to be controlled and optimized in real-time, a set of adaptations of optimal control based on PMP have been proposed in this paper.

Each method is described by its solving algorithm scheme, combined to the PMP equations, applied to different use-cases.

Results are presented, for different purposes as energy minimization on predicted driving conditions, self-adaptation of internal models, under constraints to be respected, or for multi-dimensional systems, giving acceptable real-time compliant solutions in embedded controllers, open to any discussion for possible improvements.

This brings definitively significant space for promotion of this mathematical optimization method and for its integration into on-board real-time operating strategies, able to cope with higher and higher level of complexity in real systems, under real constraints on inputs or outputs, under disturbances or ageing effects leading to deviations in models, or with possible high number of actuators to combine. Automotive applications motivated by the unavoidable trends in electrification of powertrain should get the best advantages of it.

5. REFERENCES

- [1] Emmanuel Trélat : “Contrôle optimal, théorie & applications.”
Université Pierre et Marie Curie (Paris 6)
Collection "Mathématiques Concrètes", revised edition
2013
- [2] Hartl-Sethi-Vickson : “A survey of the Maximum Principles for Optimal Control problems with State Constraints.”
- [3] Simone Baldi : “Adaptive Optimal Control.”
School of Mathematics - Southeast University (SEU)
Jiulonghu Campus - Library 514, Nanjing 211189, China
- [4] D.Vrabie, O.Pastravanu, M.Abu-Khalaf, F.L.Lewisa :
“Adaptive optimal control for continuous-time linear systems based on policy iteration”
Automatica, Volume 45, Issue 2, 2009, p 477-484
- [5] Olivier Pietquin : "De l'Apprentissage Statistique pour le Contrôle Optimal et le Traitement du Signal"
Université Paul Sabatier - Toulouse III, 2011
- [6] Delprat-Guerra-Büch : “Global Optimization in the power management of a Fuel Cell Hybrid Vehicle (FCHV)” Université Polytechnique Hauts-de-Fran
- [7] A.Sciaretta, A.Vahidi : “Energy-Efficient Driving of Road Vehicles – Towards Cooperative, Connected and Automated Mobility.”
Book – ISBN: 978-3-030-24127-8
- [8] Kamal-Kawabe : “Eco-driving using real-time optimization.”
European Control Conference - June 2015
- [9] Martin Sivertsson, Christofer Sundström, Lars Eriksson :
“Adaptive Control of a Hybrid Powertrain with Map-based ECMS”
IFAC Proceedings – Vol 44, Issue 1, 2011, p2949-2954

Towards Compact Surface Languages for Specific Modelling Aspects in EAST-ADL

Imad Berrouyne¹, Alessio Bucaioni¹, Federico Ciccozzi¹, and Henrik Lönn²

¹Mälardalen University, Västerås, Sweden

²Volvo Group Truck Technology, Gothenburg, Sweden

Abstract

The EAST-ADL is an Architecture Description Language for automotive embedded systems. It offers a comprehensive modelling solution for an integrated system, addressing diverse aspects including but not limited to variability, timing, and safety. Nevertheless, the challenge lies in the intricate nature of specifying these aspects. Both because the expressiveness adds complexity to syntax and semantics and because they are intertwined with the foundational concepts within the EAST-ADL. In this paper, we propose an approach to inject these aspects using a constraints-based surface language. Such a language offers a compact and optional description layer for annotations of the EAST-ADL.

Keywords: EAST-ADL, Embedded Systems, Automotive Systems, Feature Modelling, Variability

1 Introduction

The EAST-ADL is an established Architecture Description Language that addresses the complexities of embedded components in automotive systems [6]. With built-in feature modelling, it offers a comprehensive solution also integrating concerns related to variability [1]. However, the challenge arises in effectively specifying variability constraints within the EAST-ADL, particularly given its intricate relationship with foundational concepts. Indeed, specifying variability using the core constructs is complex and time-consuming which means that efficiency and correctness is at risk. We propose an approach to inject variability concerns into EAST-ADL models using a constraints-based surface language that operates as a loose, separate, and optional layer, offering an additional level of abstraction for constraining EAST-ADL models. Such a language provides a more flexible and modular means of incorporating variability into system architectures, providing a more compact and tractable syntax for defining basic variability. The interpretation of this surface language is conducted in conjunction with an underlying EAST-ADL core model. This integration ensures that the variability constraints are seamlessly woven into the broader modelling framework, preserving the integrity and coherence of the overall system specification. This contribution represents a first step towards offering various compact surface languages for specific modelling aspects, such as, e.g., timing and safety.

The remainder of this paper is structured as follows. Section 2 presents a running example describing a Windscreen Wiper using a EAST-ADL model along with constraints. Section 3 gives an overview of the related work. Section 4 provides our proof-of-concept solution based on a surface language and an augmented EAST-ADL model. Finally, Section 5 presents the conclusion and future work.

2 Running example

Figure 1 depicts the windscreen wiper functionality that we use as a running example. It includes three windscreen wiper variants *WiperCtrlStd*, *WiperCtrlAutoReturn* and *WiperCtrlAutomatic* based on a *DesignFunctionPrototype* specified within a core EAST-ADL model along with the constraints they must satisfy during variability resolution. As a proof-of-concept, we target three frequently used types of variability constraints: ModelYear, Brand, and Class. For instance, the *WiperCtrlAutomatic* should only be included when ModelYear is 2016, Brand is X and Class is Heavy Duty.

Product variability is typically large and follows complex rules. Our goal is to provide a

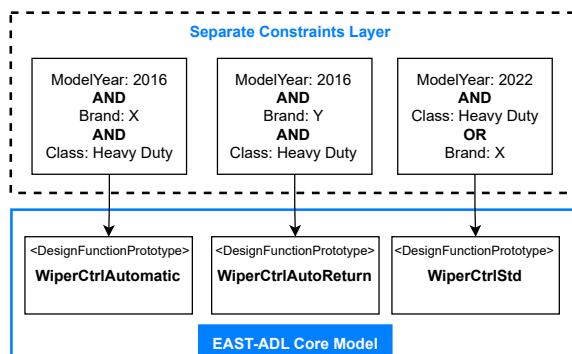


Figure 1: Windscreen wiper running example

means to specify it in a clearly expressed and separate manner. Throughout this paper, we will show how to specify these three variants and their constraints using a compact surface language, and how they can be injected into an EAST-ADL core model.

3 Related work

Existing modelling languages for automotive systems, such as AUTOSAR and SysML, generally support the description of multiple aspects, such as variability, in addition to the core concepts. AUTOSAR primarily addresses the standardization of software interfaces and communication between different software components in automotive systems [2]. It has means to express variability, but it is for the software-related elements, and it lacks a dedicated and compact syntax for capturing variability. SysML is an extension of the UML that is tailored for system engineering [9]. Variability is expressed through constructs like Block Definition Diagrams and Parametric Diagrams. However, challenges include potential diagrammatic complexity in large-scale systems and a lack of standardized approaches, leading to potential inconsistencies in model interpretation. Some studies addressed the ability to explore various aspects of modelling languages. Zhang et al. [8] introduces a DSL, named EATXT, for the specification of EAST-ADL models using textual format. The approach permits to describe an entire EAST-ADL model but lacks a means to examine the model's aspects using new abstractions. Grönninger et al. [2] suggest using model views to address the inherent complexity of representing SysML variability. These views are meant to focus on specific aspects of the entire model. Other works use model-driven techniques to enhance the management of different aspects of software product lines such as test script generation [3] and configuration files [7].

4 The proposed approach

The motivation for this paper comes from the observation that several concerns that can be addressed in the EAST-ADL are disregarded because their modelling requires a large manual effort. Further, the expressiveness and modularity of the language sometimes hide the meaning of the model. The gains of our approach are separation of concerns by extending the EAST-ADL without altering the standard, and the description of specific modelling aspects in a textual format. We achieve the latter by means of a complementary surface language that offers a limited but compact notation for the EAST-ADL extensions. The complete model still persists in the full-featured standard EAST-ADL model and is generated out of the combined surface language and core model. The enabler for this approach is the inherent separation between core constructs and extensions. The aim is to obtain an EAST-ADL model encapsulating variability through a surface language. We present the architecture of our proposed approach along with its two-step process in the following.

4.1 A two-step process

The proposed approach revolves around a two-step process consisting in 1) specifying a target aspect in a compact surface language and 2) model generation using a model transformation procedure. In the present work, we focus on the aspects of variability as a proof-of-concept. Figure 2 depicts an overview of the process. The green boxes are concepts related to the proposed surface languages, while the blue boxes represent elements strictly related to the EAST-ADL standard. The model transformation procedure results in the integration of surface language aspects into the core EAST-ADL model and leads to the generation of an augmented EAST-ADL model.

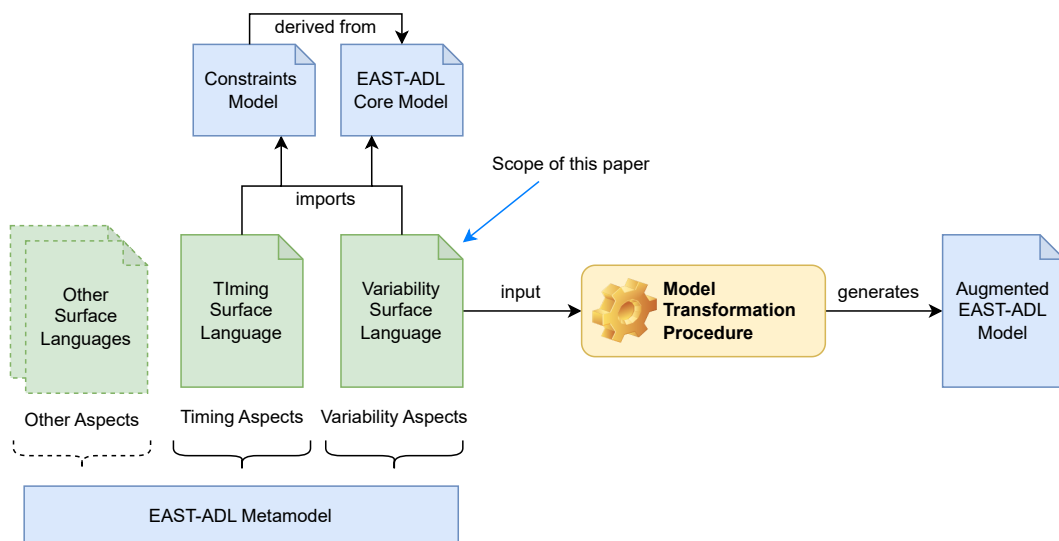


Figure 2: Process overview

In that respect, we aim to effectively disentangle variability concerns from the EAST-ADL, thereby promoting a more lucid approach to engineering such a concern. The initial phase of the proposed process involves the precise delineation of constraints using the surface language while the subsequent stage entails interpreting these constraints bound to the underlying core EAST-ADL model.

4.2 Step 1: Description of modelling aspects

The description of modelling aspects in this case study relies on a constraints-based surface language which works in conjunction with a core EAST-ADL model and a model encompassing the variability features. We assume that these features are derived from the core model *a priori* by engineers. To express the constraints for the present running example, our surface language depends on the metamodel defined in Figure 3. This metamodel consists of the root class **SurfaceModel** which is composed of **Include** instructions. These instructions permits to include an **EAElement** (e.g., *WiperCtrlAutomatic*– *DesignFunctionPrototype* is also an *EAElement*) in an augmented EAST-ADL model under certain constraints. An **Include** references a EAElement from a core EAST-ADL model and contains a **ConstraintsGroup**. A **ConstraintsGroup** consists of a one or many constraints bound using the logical operators **AND**, **OR** or **XOR**. A **Constraint** references the **Feature** that the augmented model must be constrain with.

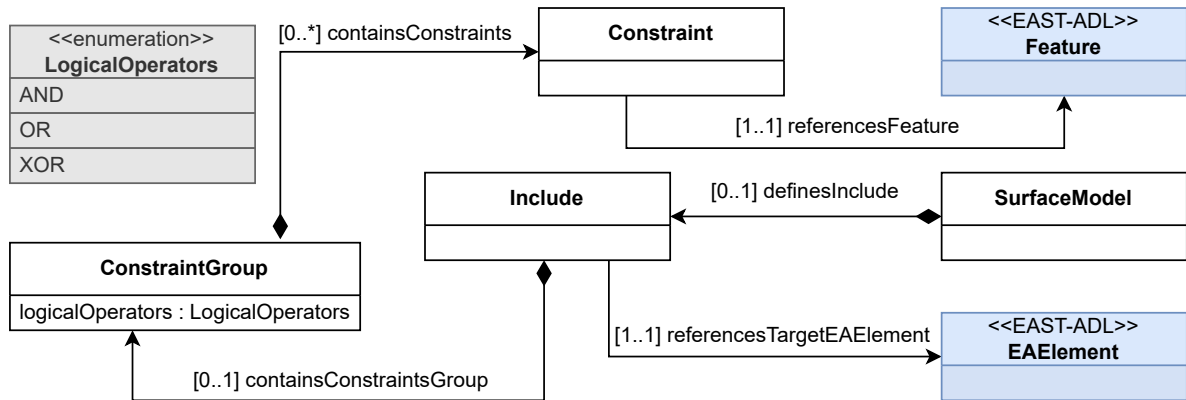


Figure 3: Metamodel of the surface language

Practically, we can describe these modelling aspects using a Domain-Specific Language (DSL) of offering a straightforward and decoupled manner to express variability using constraints. In the present example, it targets the specific cases where one wants to ensure that the final model meets certain criteria in terms of variability. However, such a DSL is meant to be compact, which limits its ability to address variability from a wider viewpoint. Thus, it is not intended to replace the built-in feature modeling or product line concepts of EAST-ADL. Listing 1 presents a potential syntax of the DSL that we applied to the running example.

```

INCLUDE WindscreenWipersPackage/WiperCtrlAutomatic IF VariabilityConstraints/ModelYear="2016"
AND VariabilityConstraints/Brand="X" AND VariabilityConstraints/Class="Heavy Duty"
INCLUDE WindscreenWipersPackage/WiperCtrlAutoReturn IF VariabilityConstraints/ModelYear="2016"
AND VariabilityConstraints/Brand="Y" AND VariabilityConstraints/Class="Heavy Duty"
INCLUDE WindscreenWipersPackage/WiperCtrlStd IF VariabilityConstraints/ModelYear="2022"
AND VariabilityConstraints/Class="Heavy Duty" OR VariabilityConstraints/Brand="X"
  
```

Listing 1: Specification of the running example using the surface language

In this example, we create three **INCLUDE** instructions. The first applies to *WiperCtrlAutomatic*, the second to *WiperCtrlAutoReturn*, and the third *WiperCtrlStd*. For instance, the first **INCLUDE** targets the *WiperCtrlAutomatic* contained in the *WindscreenWipersPackage* from the core model. It has three constraints which are all retrieved from the *VariabilityConstraints* model: *ModelYear="2016"*, *Brand="X"* and *Class="Heavy Duty"*.

4.3 Step 2: Augmented model generation

As this is a preliminary work, this second step is still under development. Nevertheless, we consider that from a conceptual perspective our goal is to populate a detailed variability model according to the content of the surface language. This implementation could be achieved using three methods. The first method could use a model-to-model transformation, consisting of weaving the surface language model with the core EAST-ADL model. The second method could rely on a model-to-text transformation by generating an EAST-ADL model incorporating the target aspects as an EAXML file based on a template. The third method could leverage the idea behind Blended Modelling [5], where we would synchronize the core EAST-ADL model in real time according to the specified aspects. The outcome

yields an augmented EAST-ADL model originating from distinct abstractions. This augmentation is achieved through the application of a surface language, allowing for the independent articulation of aspects without necessitating an in-depth exploration of the inherent intricacies of the EAST-ADL. Importantly, this process ensures the preservation of the original model's integrity and adherence to the standard EAST-ADL metamodel.

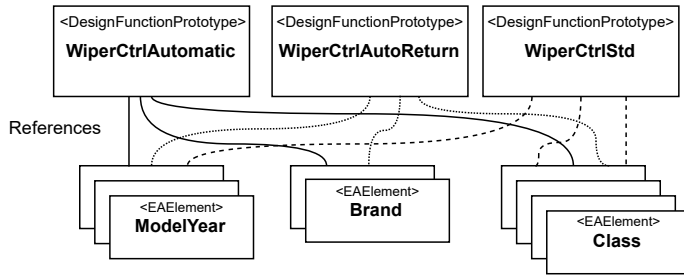


Figure 4: Augmented EAST-ADL model overview; all lines (solid, dashed, dotted) refer to references.

the mapping rules bi-directional and thus exploit them for validation of the transformed model. This process will be iterative and aims at targeting most elements of the EAST-ADL metamodel. In the case of our running example, these methods would lead to the model shown in Figure 4. We refer to the relationship between the *DesignFunctionPrototype* in the core EAST-ADL model and the injected variability aspects as a *Reference*. We use such broad terminology as we must investigate thoroughly the kind of reference that would apply and under what conditions. Nevertheless, our preliminary investigation for variability aspects and the feedback of some expert engineers suggests that such a *Reference*, conforming to the semantics of the surface language, would be generally feasible given the inherent versatility of the EAST-ADL.

5 Conclusion

The presented approach introduces a variability surface language as a proof of concept for exploring various aspects of the EAST-ADL using new abstractions. These abstractions, defined separately, give engineers a tailored manner to express their concerns. Surface languages facilitate the identification of additional aspects within the EAST-ADL, thus separating concerns, improving its expressiveness, and broadening its adaptability to meet the diverse requirements of the automotive industry. In future work, we plan to target other aspects such as safety and timing, explore more descriptions constructs and implement a comprehensive transformation procedure for generating the augmented EAST-ADL model.

Acknowledgment

The work in this paper has been supported by the Swedish Knowledge Foundation (KKS) through the Modev project, by the Excellence in Production Research (XPRES) Framework and by the Swedish Governmental Agency for Innovation Systems (VINNOVA) through the iSecure project.

References

- [1] Muhammad Waseem Anwar, Alessio Bucaioni, and Federico Ciccozzi. Timing-aware variability resolution in east-adl product line architecture. In *30th Asia-Pacific Software Engineering Conference*, December 2023.
- [2] AUTOSAR Consortium. *AUTomotive Open System ARchitecture (AUTOSAR) Specification*. AUTOSAR Consortium, 2022. Release 19-11.
- [3] Alessio Bucaioni, Fabio Di Silvestro, Inderjeet Singh, Mehrdad Saadatmand, Henry Muccini, and Thorvaldur Jochumsson. Model-based automation of test script generation across product variants: a railway perspective. In *2021 IEEE/ACM International Conference on Automation of Software Test (AST)*, pages 20–29. IEEE, 2021.
- [4] Zheng Cheng, Massimo Tisi, and Rémi Douence. Coqtl: a coq dsl for rule-based model transformation. *Software and Systems Modeling*, 19(2):425–439, 2020.
- [5] Federico Ciccozzi, Matthias Tichy, Hans Vangheluwe, and Danny Weyns. Blended modelling-what, why and how. In *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, pages 425–430. IEEE, 2019.
- [6] EAST-ADL Association. *EAST-ADL Specification*. EAST-ADL Association, 2023. Available at <https://www.east-adl.info/Specification.html>, Version 2.2.
- [7] Enxhi Ferko, Alessio Bucaioni, Jan Carlson, and Zulqarnain Haider. Automatic generation of configuration files: an experience report from the railway domain. *J. Object Technol.*, 20(3):4–1, 2021.
- [8] Jörg Holtmann, Jan-Philipp Steghöfer, and Weixing Zhang. Exploiting meta-model structures in the generation of xtext editors. In *MODELWARD*, pages 218–225, 2023.
- [9] Object Management Group. *Systems Modeling Language (SysML) Specification*. Object Management Group, 2023. Version 2.6.1.

Short Paper: Development and Evaluation of a Prototyping Platform for the Simulation, Transmission, and Real-Time Analysis of Realistic AUTOSAR Security Event Traffic

Thomas Bitterlich
T-Systems International GmbH
Munich, Germany
Thomas.Bitterlich@t-systems.com

Dr. rer. nat. Maximilian Engelsberger
Vector Informatik GmbH
Stuttgart, Germany
Maximilian.Engelsberger@vector.com

Dr. rer. nat. Grit Pientka
T-Systems International GmbH
Frankfurt am Main, Germany
Grit.Pientka@t-systems.com

Abstract— The contribution proposes a new approach of a prototyping platform simulating realistic AUTOSAR security event traffic, based on real-world attack patterns. Furthermore, their transmission between Fleethead- and SIEM-cloud systems, and their analysis within backend security services and in real-time is investigated. This advances the evaluation of technical realizations of automotive Intrusion Detection Systems, helps to gain new insights with the handling of realistic attack scenarios, and thus enables the gradual realization of the UNECE R155 regulation.

Keywords— *Vehicle Cyber Security, Intrusion Detection System, AUTOSAR, UNECE R155, Automotive SIEM, Security Operation Center, Cyber Security Management System, Service-oriented Platforms, Distributed Architectures*

I. MOTIVATION AND GOALS

Due to the rise of cyber security incidents over the past years, the UNECE R155 regulation [1] requires vehicle manufacturers to demonstrate that cyber security risks are identified, evaluated, and mitigated starting from July 2024. Road vehicles need to be continuously monitored during their operational phase to detect attacks on single vehicles as well as the entire fleet, and appropriate countermeasures must be put into place.

One approach is having an in-vehicle intrusion detection system (IDS) based on AUTOSAR IDS [2,3,4] which detects security events and reports them to a security information and event management system (SIEM) in the backend for further analysis. This allows to identify spy-outs, attack attempts, actual attacks, and eventually to derive appropriate mitigation measures.

In this contribution based on the results of a cooperation between Vector Informatik GmbH and T-Systems International GmbH, we describe the design and the prototypical realization of a platform that allows to simulate AUTOSAR security events from a vehicle fleet, which represent realistic attack patterns seen in the field. This security event traffic is transferred between a Fleethead-VPC (where the simulated fleet and the OEM's backend head is deployed) and a SIEM VPC (where the SIEM components are deployed) using an API Gateway. Finally, in the SIEM VPC, different strategies for streaming data analytics are employed to verify that realistic AUTOSAR security events can be used to identify and match known attack patterns.

The goals of this specific approach (which is work in progress) are

- to develop an understanding of real-world attack patterns,
- to simulate realistic attack patterns with the help of AUTOSAR security event mappings representing those patterns intermixed with nominal background noise,
- to demonstrate and understand the implications of data volumes, their noise/peak character, and timings,
- to evaluate and verify solutions for the transmission of this type of security event data, and
- to evaluate and verify real-time analysis and detection strategies for known attack-patterns within AUTOSAR security events.

II. STATE OF THE ART

AUTOSAR (Automotive Open System Architecture) is a standardized software framework and open E/E system architecture for mobility applications. Currently, there are three specifications released which focus on IDS-related topics. First, the general AUTOSAR requirements on Intrusion Detection Systems [2] which describes the components of a distributed onboard IDS. Second, the AUTOSAR specification of Intrusion Detection System Manager [3] which describes the functionality, API, and configuration of that basic software module. Third, the AUTOSAR Specification of Intrusion Detection System Protocol [4], which describes the format, message sequences, and semantics between the onboard AUTOSAR IDS components.

On the other hand, there are several datasets available, including known attack patterns simulated, or carried out on vehicles and captured for scientific usage such as of HCRL [6]. Further approaches for the simulation and analysis of cyber attacks on vehicle fleets are as follows: Malik and Sun [7] use a threat model to analyze and identify the most significant cyber attacks on connected and autonomous vehicles. The focus of their CARLA-based simulations is to analyze the impact of common attack scenarios on the physical world (e.g. car accidents).

Iqbal and Ball [8] use the Eclipse MOSAIC framework to model two typical road scenarios and the messaging between the vehicles and infrastructure. The model demonstrates the impact of two cyber security attacks (replay and bogus information) and generates datasets for machine learning. The approach focuses on vehicle ad-hoc networks.

Katsikeas et. al. [9] use the Meta Attack Language to develop vehicleLang, a domain-specific language which is used to codify common attack logics for the domain of automotive systems and with respect to their IT infrastructure. They use common attack patterns to generate test cases if they can be modelled with the proposed language. The approach is limited on modeling and does not focus on the simulation aspect.

To the best of our knowledge, there is currently a lack of prototyping platforms which are able to a) generate realistic AUTOSAR security event traffic based on forensic analysis of known attack patterns, b) evaluate and verify technical solutions for data transfer from the simulated vehicle fleet to the SIEM, and c) evaluate and verify analysis and detection strategies for known attack patterns within AUTOSAR security event traffic in real-time.

III. NEW APPROACH OF A PROTOTYPING PLATFORM FOR THE SIMULATION, TRANSMISSION, AND ANALYSIS OF REALISTIC AUTOSAR SECURITY EVENT TRAFFIC

A. Real-World Attack Vector and Simulation of Realistic Security Event Traffic

An important aspect of the described approach is the simulation of realistic AUTOSAR security event traffic. To this end, an actual attack vector, which is seen in the field, is analyzed. The chosen example is a CAN injection attack enabling a keyless car theft with a small injection device. Our insights are based on the forensic analysis documented by CANIS Automotive Labs [3]. This attack vector is chosen because it is a well-documented example of a real-world cyberattack on the internal systems of a single vehicle.

Prior to the actual attack, the car is physically manipulated in such a way, that it becomes possible to electrically attach an injection device to the vehicle's chassis bus. The whole attack can be divided into four major phases, during each of which a set of digital traces are generated. Each of the digital traces occurring in these phases are exemplarily mapped to AUTOSAR security events and used to define a corresponding sequence of security events representing the

attack pattern. Based on that, a Python script is written, which allows to generate this pattern including some variations and combine it with background noise of nominal security events as part of the fleet simulation (see Fig. 1).

B. Transmission of High-Volume Security Event Traffic Between Fleethead- and SIEM VPC

A basic assumption of the implemented platform is that the backend of an automotive IDS, as part of a Vehicle Security Operation Center (VSOC), can be divided into two functional domains: First, the domain which is related to the technical communication with the fleet, with optional security event transformation or refinement tasks. Second, the domain which is dedicated to classical tasks of a SIEM such as analysis, detection, and reporting. As one possible realization, the two VSOC domains are mapped on two VPCs: The Fleethead- and the SIEM-VPC. This allows a technical and organizational subdivision of the related tasks. To this end, a fully managed, reliable, and highly scalable reception infrastructure (e.g. AWS API Gateway) is set up which enables the transport of high volumes of security events from/to different and mutually decoupled producers/consumers. The reception system is also able to accept different types of messages. This is used to realize the security event transmission as well as the transport of master data, such as event- and ECU-details, from the Fleethead VPC to downstream systems in the SIEM VPC. Currently, the data flow to the vehicle is not investigated. **Fehler! Verweisquelle konnte nicht gefunden werden.** illustrates the overall prototyping platform.

C. Real-Time Analytical Approaches to Identify Attack Patterns in Security Event Traffic

To analyze the IDS events in real-time in the SIEM VPC, stream analytics services (e.g. AWS Kinesis) are evaluated. Having established that enough relevant security events can be transferred to the backend, rule based and machine learning (ML) approaches for attack detection are tested with the aim of identifying the most universal signature or best algorithm for a given attack. Such signatures currently appear to be the most straightforward way to transfer insights gained from backend analyses back to the in-vehicle IDS systems for local attack detection. Additionally, assuming that appropriate mechanisms for software updates in the vehicles exist, realizations of vehicle-based attack detection algorithms could be evaluated as well.

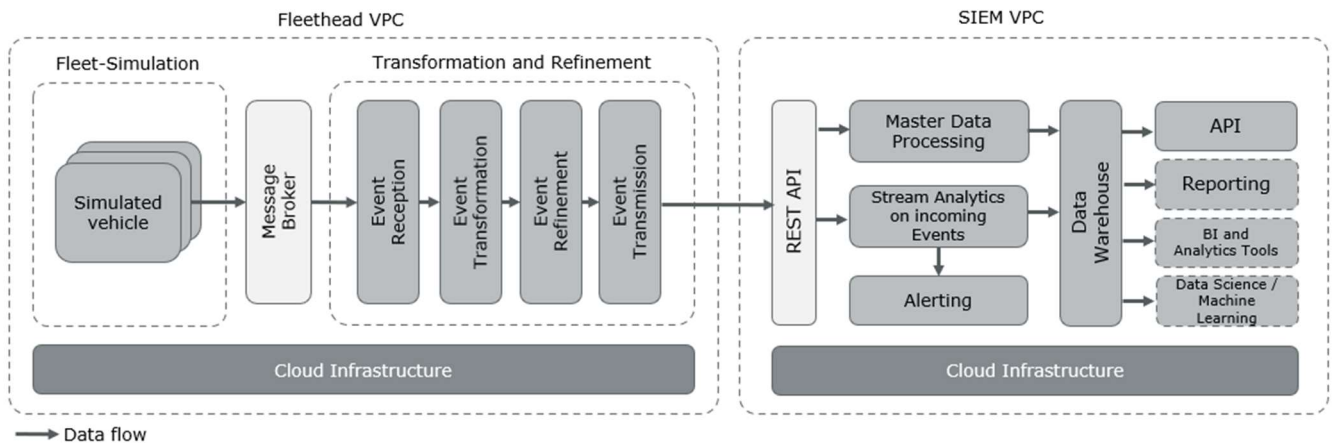


Figure 1: System architecture of the prototyping platform consisting of two domains: First, the Fleethead VPC, and second, the SIEM VPC.

Furthermore, we plan to investigate which additional data sources can aid attack detection in the vehicles as well as in the backend. Additional backend data could turn out to be particularly relevant for analyzing fleet attacks and classifying the types of affected vehicles using e.g. tree-based classification algorithms.

All results of the backend analyses are recorded, and alerts are generated if an attack has been detected. The findings are made accessible in a user-friendly application which also offers further analysis and reporting capabilities as required by UN R155.

IV. CURRENT RESULTS AND FURTHER PLANS

The contribution describes the development and ongoing evaluation of a prototyping platform for the simulation, transmission, and analysis of AUTOSAR security events. Starting from an architectural design, a working prototype of the evaluation platform has been realized. This includes the implementation and integration of all important parts: (1) Fleet simulation, (2) event transmission and (3) stream analysis.

A first real-world attack simulation has been implemented on top of the platform. This comprises the execution of attack campaigns including approximately 5300 security events per vehicle consisting of actual attack events as well as background noise. Further, first rule based real-time analytical algorithms have been realized and have been proven to reliably detect the attacks within the stream of random noise events. To study the runtimes of the algorithm with the minimal signature, 50 runs of the analytical algorithm were executed. The average runtime on the attack campaign is $\tau = 1.941$ s (median: $\tilde{\tau} = 1.934$ s, standard deviation: $\sigma = 0.035$ s). This translates to an average processing time of 368 μ s per event and thus indicates the suitability of the algorithm for real-time analyses.

Upper bounds for the execution time of the different stages in the Fleethead VPC are given in table 1. Fleet simulations (including simulated OTA transmission) are intentionally delayed simulating realistic system constraints. The number of events does not exactly scale with the number of vehicles due to some pseudo-random degrees of freedom in the noise- and attack-simulation. The throughput on fleet simulation side decreases with an increasing number of vehicles which are simulated on a given number of virtual machines. This is a bottleneck and most-likely caused by the limitations of the VM resources. This behavior can be improved by an optimized deployment strategy to allow higher numbers of fleet sizes simulated time-efficiently.

The SIEM VPC can handle events at rates up to 1000 events/s. This configuration parameter proved sufficient for the investigations performed so far. At this reception rate,

analyzing and persisting a run with 100 vehicles takes on average $\tau = 7.4$ s (min: $\tau = 3.6$ s, max: $\tau = 29.2$ s, standard deviation: $\sigma = 3.1$ s) indicating that the bottleneck currently is in the Fleethead VPC.

Further plans are a) to accomplish further evaluations of the performance of the evaluation platform itself and b) to evaluate different analytical approaches for real-time attack detection in the SIEM VPC. In this way valuable clues for the design of next-generation in-vehicle and backend-based ID(P)S – including possible strategies for real-time attack prevention – can be derived.

ACKNOWLEDGMENT

This work was funded by the Federal Ministry of Economic Affairs and Climate Action (BMWK), following a decision of the German Bundestag in the context of the SofDCar project (grant agreement 19S21002K).

REFERENCES

- [1] UNECE, UN Regulation No. 155 - Cyber security and cyber security management system, <https://unece.org/transport/documents/2021/03/standards/un-regulation-no-155-cyber-security-and-cyber-security>, last accessed 2023/05/09.
- [2] AUTOSAR RS Intrusion Detection System - Requirements on Intrusion Detection System, https://www.autosar.org/fileadmin/standards/R20-11/FO/AUTOSAR_RS_IntrusionDetectionSystem.pdf, last accessed 2023/06/01.
- [3] AUTOSAR SWS Intrusion Detection System Manager – Specification of Intrusion Detection System Manager, https://www.autosar.org/fileadmin/standards/R20-11/CP/AUTOSAR_SWS_IntrusionDetectionSystemManager.pdf, last accessed 2023/06/01.
- [4] AUTOSAR PRS Intrusion Detection System - Specification of Intrusion Detection System Protocol, https://www.autosar.org/fileadmin/standards/R20-11/FO/AUTOSAR_PRS_IntrusionDetectionSystem.pdf, last accessed 2023/06/01.
- [5] Tindell, K., CAN Injection: keyless car theft, CANIS Automotive Labs, <https://kentindell.github.io/2023/04/03/can-injection/>, last accessed 2023/06/01.
- [6] HCRL (Hacking and Countermeasure Research Lab): Datasets for intrusion detection for automobile, <https://ocslab.hksecurity.net/Datasets>, last accessed 2022/06/07.
- [7] Malik, S., Sun, W., Analysis and Simulation of Cyber Attacks Against Connected and Autonomous Vehicles, 2020 International Conference on Connected and Autonomous Driving (MetroCAD), last accessed 2024/03/20.
- [8] Iqbal, S., Ball, P., Simulating Malicious Attacks on VANETs for Connected and Autonomous Vehicle Cybersecurity: A Machine Learning Dataset, arxiv.org, last accessed 2024/03/20.
- [9] Katsikeas, S. Johnson, P., Hacks, S., Lagerström, R. Probabilistic Modeling and Simulation of Vehicular Cyber Attacks: An Application of the Meta Attack Language, ICISSP 2019, last accessed 2024/03/20.

Run ID	Fleet Size	#Events	Total Data Volume (MB)	Fleet Simulation (including simulated OTA-transmission)			Transformation and Refinement	
				Total Execution Time (s)	AVG Noise Throughput per vehicle (Events/s)	#Vehicles per VM	Execution Time (s)	AVG Noise Throughput (Events/s)
22	1	2742	0.68	280	9.5	1	280	9.5
23	10	30586	7.6	420	8.4	10	420	72
24	100	303340	75.1	920	5.4	10	880	343

Table 1: Measurement results of the execution times and event throughputs of the different stages

Timing Architecture Model for Embedded Systems Anomaly Detection

Peter Heller and Jürgen Mottok

Laboratory for Safe and Secure Systems (LaS³)
Technical University of Applied Sciences Regensburg
93053 Regensburg, Germany
{peter2.heller, juergen.mottok}@oth-regensburg.de

Abstract—By using execution timing behaviour to discover anomalies, embedded systems can be monitored at various architectural layers. Different methods for deducing sane system execution behaviour based on available event or timing data are proposed in the current literature about security-related anomaly detection of embedded systems. With our work, we evaluate several strategies and discuss problems with accessible metrics and architectural components used for feature development. An embedded system’s architecture layers serve as the basis for a common classification scheme that makes it possible to combine timing- and event-based metrics into a single timing architecture layer model. Then, using metrics and architecture components, our suggested model is applied to several anomaly detection techniques and utilized to compare existing methods. Our mapping leads us to the conclusion that most detection models are restricted to single system layers (i. e., communication or application code) and use a small number of accessible architecture levels. Our existing model allows us to combine various time and event metrics, but we also want to develop new features for embedded anomaly detection that can be used across all system layers (code, scheduling and communication).

Keywords—*anomaly detection, time series data, embedded systems software, architecture layer model*

I. INTRODUCTION

Due to ongoing regulatory developments, preventive measures against malware will be an integral part of the product lifecycle within the European Union in the foreseeable future. With the two directives 2023/1230 [1] and NIS 2 [2] issued by the European Parliament, manufacturers and suppliers will be obliged to implement appropriate security measures for their products. One objective of these regulations is to counteract tampering and to prevent or at least mitigate malicious intrusion of safety-relevant control systems. Real-time capable systems exhibit measurable side effects on the remaining system when subjected to changes regarding tasks or other critical code sections (see [3]). Analysing execution timing offers a way to implement anomaly detection by evaluating changes to known system behaviour based on a selection of indicators at different architecture levels, and thus to identify suspicious activity.

Available research on anomaly detection provides us with various approaches for detecting changes in system behaviour. The problem with current solutions, however, is that it is difficult to compare the different methods with one another. Some approaches use event-based metrics from the operating system or a communication stack, while other models use timing measurements at the instruction level to detect deviations from the specified behaviour. Because no common model exists to date, one objective of this paper is to derive a suitable classification to map available metrics from the literature into a single scheme. Scheduling theory

for embedded real-time systems uses models and architecture frameworks to characterize the execution behaviour. By combining an architectural model and suitable notations for different workloads from scheduling theory, we introduce a model that can combine time- and event-based metrics from different architecture levels in a common representation. We want to use this representation to merge existing features for anomaly detection in a hybrid approach that leverages data based on a collection of metrics captured from different architecture levels.

In summary, the focus of this work is to address the following research question:

Q1. Can available timing and event metrics for embedded systems anomaly detection be mapped into a common classification scheme based on their system architecture?

The following Section II starts with a general introduction of the literature on embedded systems anomaly detection with security related context, where we focus on RTOS based systems. We discuss the general problem of using time-series data and give a brief overview of the current state of the art. Section III then presents the timing architecture layer model for developing a hybrid approach on multiple architecture levels, and categorizes existing literature from Section II based on the aforementioned model. The final Section IV briefly summarizes the key findings and the next steps in our research process.

II. RELATED WORK: ANOMALY DETECTION USING SYSTEM TIMING AND EVENT METRICS

The current literature on embedded systems presents a challenge due to the variety of methods employed, architectural levels involved, and underlying metrics applied in anomaly detection for time-series data. To improve our understanding, we want to focus on two important aspects based on anomaly detection literature: The applied metrics (runtime-data and events) and architectural components of currently available approaches for embedded systems. First, how available metrics are used and what type of data those metrics are based on. Secondly, which layers of the system architecture were used and provided to be useful for the development of new anomaly detection features. To narrow the available literature on anomaly detection using time-series data, two selection requirements were used: (i) The Data source for anomaly detection is based on embedded system traces. (ii) Selected publications should have a security context. Context in this regard implies that the objective for anomaly detection is security-relevant for the system, or that the verification process of the created anomaly detection mechanism is targeting tampering of the target system. The purpose of this

selection is to ensure that the range of relevant metrics for the subsequent comparison have common ground.

In general, the available literature can be split into two different categories based on the applied metrics: *event-based* and *time-based* approaches. Events are observable changes in system state and are generated during system execution. The goal of recording events is to determine if a sequence of events, the frequency of events or temporal dependencies show measurable deviations from normal behavior. Depending on the architectural origin of an event (i.e. application- or instruction-level), it can be observed either from software or hardware. Events can be described as a tuple of system defined values (see [4]) denoted as $e = \langle v_1, v_2..v_x \rangle$ holding event-specific information, which in turn is highly dependent upon where the specific event is generated (network-stack, scheduler, application code, instruction-level). Optionally, a timestamp can be incorporated to denote chronological dependencies between single events for time-based approaches. In this case, the timing information needs to be obtained and processed either on the target system or measured by using an external time base. Depending on system complexity and hardware architecture, we can observe and record different event streams at varying levels of granularity regarding overhead and intrusiveness. Therefore, we can not always observe a single event, but a subset of available data.

Embedded real-time systems implement deterministic scheduling models to design and verify their intended functionality, and thus exhibit recurrent behaviour and distinct execution patterns. Three identified approaches [4]–[6] use Inter Arrival Curves (IACs) or modified modelling techniques with similar properties to infer system state based on a selection of events. IACs characterize the activation pattern of individual event streams, by limiting upper and lower bounds of event occurrences for a given time window. A combination of different activation functions based on arrival curves allows modelling the dynamic behaviour of different event-based systems. Ezeme et al. [5] and Torres [6] have shown how recurrent timing and pattern detection can be implemented for (online) system monitoring utilizing the aforementioned event-based metrics. Salem et al. [4] use a different approach by aggregating multiple event streams into sequence-based arrival curves (IACs) which are then used to derive suitable features for anomaly detection. Hoffmann et al. [7] utilize a similar approach with multiple event streams, but on a different architecture level. Their work focuses on performance metrics that are captured at CPU execution level using system performance counters. Lu and Lysecky [3] implement different models (range-based, distance-based and SVMs) utilizing only timing parameters at the lowest architectural levels to detect changes in software behaviour when subjected to different types of malware. In this case, the hardware trace port is used to extract core registers and measure timing with an external time base. By exposing several signals from the hardware trace port, cache, and pipeline effects can be observed and fed into the detection algorithm. With their hardware-based approach, they can measure execution time down to the instruction level and capture specific timing parameters they are interested in.

Given our available literature, we can apply a preliminary classification of the individual parameters. First, high-level software events, which can be captured from the operating system, a scheduler, or the communication stack of a target.

Due to the nature of these events, the streams, and traces contain a high level of noise from other event sources. Second, low-level events, which are based on performance counters or hardware peripherals and allow detection of anomalies based on core execution behaviour at the deeper system levels. Third, execution time, which can be determined at different architecture levels and with different methods, but usually requires measurement. All of those metrics use different data types and originate from different system architecture levels. What is interesting to us is a) the variety of available metrics and b) the limited use of the available system architecture components, which we want to address in the following Section III. For this purpose, we want to introduce our timing architecture model to map parameters and architecture components into a common classification scheme. We want to apply our model to develop new features for anomaly detection, allowing us to extend detection models to all available system architecture layers. To the best of our knowledge, there is no known study or paper that maps available time- or event-based metrics into a common model or scheme targeting embedded systems at the time of writing.

III. MAPPING TIME AND EVENT METRICS BASED ON ARCHITECTURE LEVELS

System architecture layers can be used as a descriptive tool to decompose and analyse metrics within a given system, which is why we choose this approach as a tool to compare available literature. We are interested in understanding how different methodologies for embedded devices use the existing system architecture to implement effective anomaly detection with event streams and time series data. For this purpose, we based our model on an existing layering model for timing analysis of embedded real-time systems [8] and extended their work by applying a workload-based decomposition from the automotive domain [9]. This extension of the model allows us to further differentiate existing approaches below the application level, so we can map available anomaly detection methods based on architecture and metric usage.

Based on our preliminary evaluation of available literature, we found that event-based metrics are used more frequently instead of runtime measurements. Therefore, we need to consider both timing and event-based metrics during our mapping procedure. For this purpose, our current model provides two degrees of abstraction, called *layers* and *levels*, to represent architecture and timing/event properties we are interested in. The available model provides three high-level architecture layers: a *Communication Layer*, a *Scheduling Layer* and an *Application Code Layer* to represent the generic structure of an embedded system. Layers are used to represent specific key timing properties of an embedded system, such as *Core Execution Time (CET)*, *Response Time (RT)*, and *Round Trip Time (RTT)*. Levels are used to assign a source of events or timing parameters to a system component. They allow further differentiation of where different timing parameters, events, and other interesting metrics originate and where their area of effect is located inside the architecture. For a single layer, subcomponent timing is used to calculate each key parameter based on the assigned levels. For further reference, Table I shows an overview of all nine architecture levels, abstraction layers and descriptions for relevant timing and event metrics as well as the key parameter for each Layer.

TABLE I
SYSTEM ARCHITECTURE LAYER MODEL FOR DECOMPOSING TIMING AND EVENT METRICS BASED ON ARCHITECTURE ORIGIN

Name	Level	Layer	Key Parameter	Description
Instruction	1			Smallest measurable quantity for tracing (single instruction, pipeline events, caching, fetch)
Basic Block	2			Continuous sequence of code with a single entry and exit point (branch — jump)
(Sub) Function	3	Code	CET	Decomposed task job sequences (notification, locks, signals, resource access)
Top-Level Function	4			Runnables and task-jobs, workload management (job start, job stop, job dispatch)
Task, ISR	5			Task start/stop, task preemption, application specific ISR, background task jobs
CPU-Core	6	Scheduler	RT	Singlecore scheduling effects: synchronization, spinlocks, memory-management, scheduling ISRs
Processor	7			Multicore scheduling effects: synchronization, spinlocks, memory-management, scheduling ISRs
ECU	8	Comm.	RTT	Interprocessor level: bus interfaces between system internal devices (SPI, I2C ...)
Network	9			External level: Networked Signals/Events, Network-Latency, Roundtrip Time

A. Timing Architecture Layer Model

While the original layering model for timing analysis does not provide a formal notation, other works [9], [10] can be used to enhance the mapping of available events and timing parameters to the architecture layers. For our use case, we remove the lowest available level (OP-Code/Micro-instruction, L0) from the original model [8], since single machine instructions are the smallest measurable quantity we can effectively capture and process using available tracing methods.

Considering the existing model, the key parameter to the code layer is the Core Execution Time (CET) of the associated task(s) or application. The CET reflects the actual time a specific task, a function, or a sequence of instructions executes without overhead through preemption or scheduling. Any timing parameters and associated events necessary to compute the raw execution time of application code are accounted for within the first five layers of the architecture model. For this purpose, we introduce a workload-based decomposition that can be employed to partition application code of an embedded system into three different levels of granularity. Levels 3 to 5 can be illustrated as a set of tasks implementing an event queue executing requested workloads, as shown in Figure 1. Depending on the event queue, the active application task handles specific system events, that are scheduled to execute certain processing workloads (jobs). At each event execution cycle, queue entries are released and used to schedule a number of jobs for the active task. Events and timing parameters related to level 5 of our architectural model are captured in the scheduler and task loop of the active task. This level is used to observe timing and behaviour at task level granularity, i.e. when a single task is scheduled or pre-empted. Overhead through execution time for maintenance, monitoring and tracing functions, application specific ISRs and background jobs executing in the idle task are included at this level. Similarly, events (i.e. logs or syscalls) generated through monitoring, profiling and tracing are associated to this level. Level 4 consists of timing and events generated through executing scheduled workloads for a single task. This level is used to trace how workloads consisting of a single task job or a continuous sequence of individual jobs behave.

Different authors define a *job* or a *runnable* as a collection of code that performs higher level functionality (see [9], [10] and [11]). Since runnables or task jobs implement higher-order functions, we needed another level of abstraction to break down application code below this level. This would allow us to characterize functions, events, and portions of code needed to perform complex operations. For this reason,

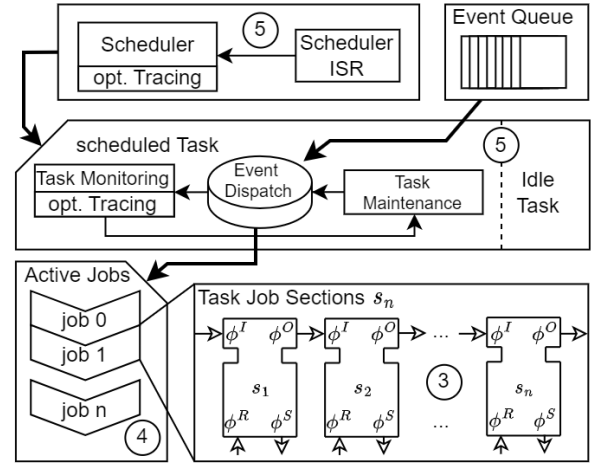


Fig. 1. Overview of a generic embedded application with architecture and event/timing components: scheduling and active task correspond to level 5, task jobs to level 4 and decomposed sections are assigned to level 3.

we introduce the definition of a *task job section*, which can be understood as a section of code that contains a sequence of instructions defined with an execution time model and signal vectors $(\phi^I, \phi^O, \phi^R, \phi^S)$, per definition of [9]. For our use case, we want to extend this definition to be applied at the granularity of single jobs instead of the entire task, hence the naming task job section. Signals are input or output events for any given task, including the executing task, which are used to control or synchronize subsections of jobs and other runnables (tasks). Signals can generally be distinguished into two different categories: signals required for computation and signals required for synchronization. First, the input and output signal vectors (ϕ^I, ϕ^O) are computational signals because they provide the data being used and transformed during processing within a single task job activation. These signals define the interface of a given task job section, since they enforce the required input and output parameters needed for a specific section of code. The second category can be described as event signals, which are denoted as requested signals (ϕ^R) and supported signals (ϕ^S) . Requested signals (ϕ^R) are required input signal vectors, that are consumed at the start of each section. This type of signal is used to provide events (i.e. notifications or data from other tasks) to a sequence of code and can be used for synchronization purposes. Supported signals (ϕ^S) are consumable events or notifications generated by the executed section of code, which are required by other tasks or subsequent task activations.

Timing based on task job sections at the granularity of available signals makes up level 3 of our architecture model.

This level allows analysing execution runtime by tracing single signals passed between tasks and the scheduler/operating system, as well as timing based on the execution model of each task job section. Generally, these signals are used to exchange data between tasks or for resource management implemented by the scheduler.

The second level (2) is used to assign metrics that can be captured at basic block granularity. In this case, any sequence of code between branch/jump instructions can be instrumented or measured. On the first level (1), the timing or activation of a single instruction is the primary feature. This type of measurement allows for single cycle granularity when capturing available metrics, either by runtime measurement of a single instruction or by observing instructions through a trace-port as an event stream. Since our mapping needs to cover timing, event and counter-based metrics, that can be observed through specialised hardware, we need to include caches, retired instructions, memory access and pipeline effects at this level.

The key parameter, within the scheduling layer, is the Response Time (RT) for the scheduled task set. This layer introduces execution time and latency from the scheduler, to account for timing variations outside of application code. Level 6 represents timing and events that are present at single-core task scheduling level. This level is used to capture timing effects introduced by scheduling and resource management handled centrally throughout the system. Runtime caused by locking mechanisms, timing overhead introduced by the scheduler, or scheduling ISR are accounted to this level. Level 7 extends the definitions of level 6 to multicore scheduling and overhead that is introduced by sharing cores, resources, and workloads. The distinction between the two levels is used to separate the impact caused by multicore processors, instead of mixing metrics together.

The final communication layer is used to introduce timing and events related to external communication and focuses on the Round Trip Times (RTT) for available interfaces. For illustrative purposes, we further differentiate between interprocessor (level 8) and external communication (level 9). At the interprocessor level, system-bus timing and packet handling events within system boundaries are handled. The external level handles events and signals leaving the system boundaries, e. g. network connections to other systems.

B. Application of the Timing Architecture Model

We introduced our timing architecture layer model to map available metrics provided by related work. Our goal was to create a common representation to compare several embedded anomaly detection techniques, based on metrics and architectural components. We saw that two publications [3], [7] implement detection models utilizing timing and events at levels 1 to 3. In addition to the lower level timing aspects, timing variation and events due to cache and pipeline behaviour are also considered in both approaches. All other works [4]–[6] are focused on high-level metrics (5,8,9) and implement their approaches without any additional input from the underlying architectural layers. Table II shows a quick outline of the different parameters used by current approaches, as well as our applied mapping.

From the existing publications, we can infer that communication and code layer parameters are being preferred as a source for different anomaly detection methods. Specific

TABLE II
APPLIED SYSTEM ARCHITECTURE LAYER MODEL

Source	Used Parameter	Architecture Level	Architecture Layer
[3]	Execution Time, Cache	1,2,3	Code
[5]	Inter Arrival Curves	5,8,9	Code, Comm.
[4]	Inter Arrival Curves	5,8,9	Code, Comm.
[6]	Inter Arrival Curves	5,8,9	Code, Comm.
[7]	Slack, Interrupts and Event-Counters	1	Code

timing parameters from timing theory (CET, RT) are found in two of the available publications as part of input vectors for heuristic models or machine learning-based detectors. Event-based systems tend to use the sequence information of observed event streams at the code and communication layers to infer the state of the underlying system. Based on the architectural model, we can determine that the scheduling layer is not actively used in any available publication. While tasks are monitored, instrumented and traced at the code level, resource management, workload sharing and locking features are underutilized. For communication-based systems, the application should require heavy use of resource management and locking features, making them good candidates as features for anomaly detection in this context.

IV. CONCLUSION AND OUTLOOK

Our goal was to evaluate how anomaly detection for embedded systems is performed and what architecture components are used to implement novel detection algorithms. We introduced a timing architecture layer model for mapping publicly accessible timing and event metrics based on architecture components into a common classification scheme. The proposed model was introduced as a comparison tool, since current research uses various metrics and architecture layers to implement anomaly detection methods. We applied our model to the available anomaly detection literature, which was selected based on embedded security context, and show how different timing and event related metrics are utilised. We found that there is an under-utilization of parameters present in the scheduling layer of our architecture model. We also believe that a more in-depth comparison of time and event metrics could be a valuable contribution.

Our work targets embedded devices with strict timing requirements, like soft and hard real-time embedded systems or RTOS based systems. Since their system operation has specific requirements on execution timing and response, even slight changes can have measurable impact on the timing characteristics. For our future work, we want to evaluate whether anomaly detection on all available system layers (code, scheduler, and communication) is a feasible approach based on data available from embedded systems or whether single level approaches are sufficient. In terms of application within the security domain, another area of interest is remote attestation protocols. For our future work, we would like to determine, if time-based anomaly detection could be extended to all available system layers for attestation protocols. With our current work, we want to evaluate novel features, so we can determine which architecture component provides the best use for effective anomaly detection and parameter selection.

ACKNOWLEDGMENT

The presented work is part of the research project KRITIS Scalable Safe and Secure Modules (KRITIS³M), which is funded by the Project Management Jülich (PtJ) and the German Federal Ministry for Economic Affairs and Climate Action (BMWK) under funding code 03EI6089A.

REFERENCES

- [1] Council of European Union, “Regulation (EU) no 2023/1230 on machinery and repealing Directive 2006/42/EC of the European Parliament and of the Council and Council Directive 73/361/EEC,” Official Journal, 6 2023, L 165, p. 1.
- [2] —, “Directive (EU) 2022/2555 of the European Parliament and of the Council of 14 December 2022 on measures for a high common level of cybersecurity across the Union, amending Regulation (EU) no 910/2014 and Directive (EU) 2018/1972, and repealing Directive (EU) 2016/1148 (NIS 2 Directive),” 12 2022, L 333, p. 80.
- [3] S. Lu and R. Lysecky, “Data-driven anomaly detection with timing features for embedded systems,” ACM Trans. Des. Autom. Electron. Syst., vol. 24, no. 3, apr 2019. [Online]. Available: <https://doi.org/10.1145/3279949>
- [4] M. Salem, M. Crowley, and S. Fischmeister, “Anomaly detection using inter-arrival curves for real-time systems,” in 2016 28th Euromicro Conference on Real-Time Systems (ECRTS), 2016, pp. 97–106.
- [5] M. O. Ezeme, Q. H. Mahmoud, and A. Azim, “Hierarchical attention-based anomaly detection model for embedded operating systems,” in 2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), 2018, pp. 225–231.
- [6] R. T. Labrada, “Multi-signal anomaly detection for real-time embedded systems,” 2020, master thesis, University of Waterloo.
- [7] J. L. C. Hoffmann, L. P. Horstmann, and A. A. Frohlich, “Anomaly detection in multicore embedded systems,” 2019 IX Brazilian Symposium on Computing Systems Engineering (SBESC), pp. 1–8, 11 2019.
- [8] P. Gliwa, Embedded Software Timing: Methodik, Analyse und Praxistipps am Beispiel Automotive. Springer Fachmedien Wiesbaden, 2021. [Online]. Available: <https://doi.org/10.1007/978-3-658-26480-2>
- [9] M. Deubzer, “Robust scheduling of real-time applications on efficient embedded multicore systems,” Ph.D. dissertation, Technische Universität München, 2011.
- [10] M. Alfranseder, “Efficient and robust dynamic scheduling and synchronization in practical embedded real-time multiprocessor systems,” Ph.D. dissertation, Technische Universität Clausthal, Clausthal, Dec 2016. [Online]. Available: <https://doi.org/10.21268/20161207-083032>
- [11] S. K. Baruah, M. Bertogna, and G. C. Buttazzo, Multiprocessor Scheduling for Real-Time Systems, 2015th ed., ser. Embedded Systems. Springer International Publishing, Apr. 2015.

Signal integrity challenges of complex high-speed serial links up to 25 Gbps in an aeronautic environment

Soazig LE BIHAN
THALES Avionics
Merignac, FRANCE
soazig.lebihan@thalesgroup.com

Marc GATTI
THALES Avionics
Merignac, FRANCE
marc-j.gatti@fr.thalesgroup.com

Tristan DUBOIS
IMS Bordeaux
Talence, FRANCE
tristan.dubois@u-bordeaux.fr

Adil EL ABBAZI
THALES Avionics
Merignac, FRANCE
adil.elabbazi@fr.thalesgroup.com

Jean-Baptiste BEGUERET
IMS Bordeaux
Talence, FRANCE
jb.begueret@ims-bordeaux.fr

Abstract: Developments in printed circuit technologies have evolved over the years, enabling increased circuit density and finer engravings. This progression has also led to higher data rates, increased clock speeds, reduced switching times, and lower power consumption, all within increasingly limited spaces. Signal integrity (SI) is a crucial aspect in the design of electronic boards, as multiple factors can impact signal quality, including signal attenuation, impedance matching, crosstalk, and jitter. Signal attenuation arises from dielectric and conductive losses, which must be carefully considered in the analysis of signal integrity.

Therefore, a more precise approach is necessary to model these effects in simulation, taking into account the actual electrical properties of the dielectric and copper of the printed circuit board (PCB). Moreover, analyzing and optimizing every potential discontinuity has become an essential part of SI analysis, as it can result in reflection and insertion losses along the trace. The achievement of an automated process becomes crucial for every new aerospace design. Several boards are currently being developed for military and civil aerospace projects, incorporating high-speed links (SerDes with Ethernet) up to 25 Gbps and more and very high densities, while operating under severe environmental conditions (EMC, thermal, vibration...).

Keywords—Signal integrity, High-speed links, SerDes, Models, Roughness, Glass weave skew, Optimization, Equalization

I. INTRODUCTION

The increasing demand for high-performance computing and faster connectivity in aeronautical domains, such as radar, electronic warfare, and data and image processing, requires a rigorous study of signal integrity. This is particularly crucial in modern high-speed communication, with higher data rates and faster rise times. This improves the reliability and accuracy of the transmitted data, making communications more reliable.

Signal integrity analysis should no longer assume that the dielectric is homogeneous and the copper is a flat surface. It must take into account its actual structure within the Printed Circuit Board (PCB) and its anisotropic and frequency-dependent properties. While rules of thumb and models have been developed to address these issues, special care needs to be taken regarding the impact of these factors on the signal, especially in light of the evolution of new technologies.

Signal integrity is a crucial factor to consider when designing and routing electronic boards. The design must follow rules and specific routing methodologies to ensure the reliability of data transmission between components across the discontinuities of the PCB, such as FPGA BGA (Ball Grid Array), DC blocking capacitors, connectors and vias. These discontinuities can result in signal attenuation, crosstalk, jitter, and impedance mismatch. Careful optimization is needed to maximize the signal's transparency. As the optimization of the PCB has reached its physical limits, signal processing techniques have been implemented within the transceiver to account for potential signal degradation.

Accurate modeling and simulation have become increasingly important in guaranteeing signal integrity. This ensures that a board operates properly from the initial tests and helps to minimize manufacturing costs and to avoid iterations. In order to enhance productivity, the optimization process of a new high-speed serial link must be more flexible and automated to accommodate potential design changes thanks to mathematical algorithms.

II. PCB TECHNOLOGY

As data rates and PCB complexity increased, considering the real structure of the conductive and insulating layers became necessary. This involves taking into account the effects of roughness and the influence of materials like fiberglass on signal integrity.

A. Copper roughness

Conductive losses on a PCB are caused by factors such as surface roughness, etching factor, skin effect and metal losses. The manufacturing process of copper foil on a PCB can be performed through several methods. The base copper electroplated (ED) is the most commonly used. ED copper is created by electroplating copper ions from an electrolyte bath onto a rotating drum. The copper on the drum side is smooth, while the one on the solution side (mat) is rougher. The drum side is often bonded to the PCB core for high-speed links (Reverse Treated Foil). This roughness is classified from roughest to smoothest (III, VLP, S-HTE, VSP) [1]. Based on the class of the roughness profile, the copper surface can lead to higher frequency losses due to an increased skin effect. These losses tend to increase when the trace is long and

narrow, which is often the case in High-Density Interconnect (HDI) designs.

Up to 10 GHz, the impact of roughness on signal loss is negligible. However, as data rates increase, it has become necessary to consider its impact for a more precise characterization of circuit performance. The presence of roughness introduces inaccuracies in losses, dielectric constant, and phase delay values [2]. Depending on the manufacturer, the roughness profile may vary. The dielectric properties change when considering the surface roughness. The copper roughness decreases the spacing between two conductors, resulting in increased capacitance and the dielectric constant's value (Dk).

It is essential to incorporate roughness into simulations in order to obtain reliable results. Mathematical expressions have been developed to model roughness and can account for its impact by calculating a correction factor K_{SR} in the insertion losses formula (IL) [3] as follows:

$$IL_{total}(f) = IL_{diel}(f) + K_{SR} * IL_{conductor}(f)$$

To approximate the actual electroplating process, the Cononball-Huray model provides an approach to the roughness model as a configuration of stacked spheres forming a hexagonal pyramid (Fig. 1). By identifying the number of spheres and their radius, it is possible to calculate its area and determine K_{SR} , based on the data provided by the manufacturer. The introduction of a new material must be characterized by profilometry, for instance, to ensure the accuracy of the roughness data provided, especially after chemical treatments. The measurement of an HVLP copper revealed a roughness of approximately 1um for the Roughness Average (Ra), compared to a manufacturer-supplied value assumed to be less than 0.43um.

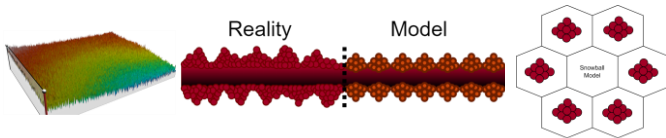


Fig. 1. Correlation between surface roughness measured by profilometer and the Cononball-Huray model

There are various alternative models available for calculating K_{SR} . It became necessary to compare the benefits of these models with each other, as well as with the actual measurements, to determine which ones provide the most realistic and reliable results for high-speed designs up to 25 Gbps and more.

B. Dielectrics

A printed circuit board (PCB) is made of insulating and conducting layers. Previously, insulating layers were commonly referred to as FR4. However, because the losses are mainly dominated by dielectrics at high frequencies, design margins and material properties have become more crucial.

The selection of a dielectric is a key factor in ensuring product reliability, as it needs to meet specific constraints for robustness in the electromagnetic environment and extreme conditions in aeronautics.

Thermal parameters, such as glass transition (Tg), time to delamination (Td), and coefficient of thermal expansion (CTE) in the x/y and z axes, play a significant role. Electrical parameters, including dielectric constant (Dk) and dissipation factor (Df), affect losses, propagation speed, and impedance. Additionally, mechanical parameters such as peel strength, water absorption, Young's modulus, or Poisson's ratio are essential considerations in designing a new stackup. Each new dielectric must be qualified to verify its properties and ensure reliance on the manufacturer's data [4].

The choice of stackup, and more specifically, the type of glass used, becomes crucial in ensuring the robustness and reliability of high-speed signals. The dielectric commonly used in printed circuits is made of woven glass fibers encapsulated in resin (Fig. 2). The electrical properties of Dk and Df depend on the ratio of these two materials. An increase in resin content results in a decrease in the Dk value and an increase in the Df value.

The inhomogeneity of the material can lead to disparities in the propagation velocities within the differential pair. This effect, called "glass weave skew", can generate mode conversions, skew, resonances, and losses caused by impedance mismatches along the trace [5]. This will lead to potential bit errors, a more closed eye, and contribute to deterministic jitter generation.

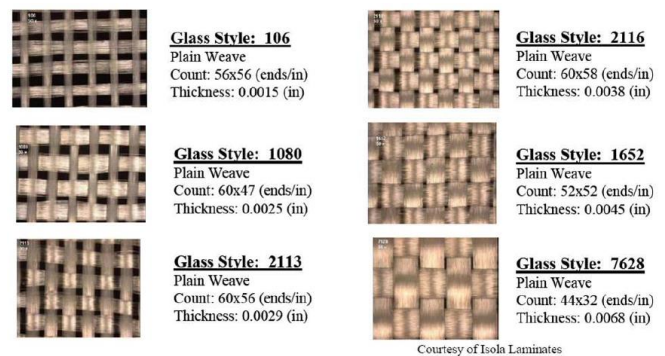


Fig. 2. Glass Type (Source: ISOLA)

There are various choices available for dielectrics, and selecting the right one can be challenging due to variations in thermal, mechanical, and electrical properties. Therefore, understanding the capabilities and requirements of the system, including factors such as performance and operational conditions (EMC, temperature, humidity and vibrations), is crucial [6].

Finally, PCB stackup needs to be compliant with microvia technology, essential for HDI designs and high-speed signals.

The MEGTRON6 dielectric facilitates the development of high-frequency stack-ups involving multiple laminations and varying levels of microvias. This material is available in both laminates and prepregs. However, the industry is driven to explore alternatives in different countries due to the demand for secondary source materials, performance considerations, and cost factors.

A trade-off must be made in order to satisfy the need for performance while remaining competitive in the market by

reducing the cost of the product. The specific choice of stackup could lead to higher costs and longer supply and manufacturing lead times. Alternatively, it might require placing an order for a minimum quantity of PCBs in order to justify launching manufacturing at the supplier.

III. OPTIMIZATION APPROACH

A. Discontinuities optimization

Discontinuities such as vias, connectors, DC blocking capacitors, BGA fanouts ... can significantly degrade the signal along the channel (Fig. 3). Poorly matched impedance can lead to unwanted signal reflections, energy losses, and electromagnetic interference. The analysis and optimization of 3D transitions are important steps in high-speed serial link analysis. Each of them must be carefully studied in order to minimize the overall impact of each transition.

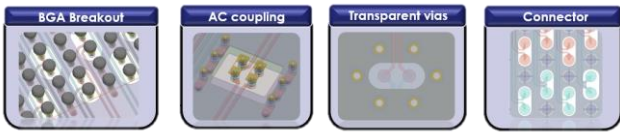


Fig. 3. Optimization of discontinuities for high-speed links under HFSS

To route high-speed serial links, it is essential to follow basic rules, such as using stripline structures between ground layers, to ensure immunity to electromagnetic interference. The high density of the BGA often requires routing signals across multiple internal layers, which necessitates the use of microvias and via-in-pads to minimize trace length, which are recommended for high-frequency signals.

A cutout of the reference planes under the capacitor or BGA in the path of the high-speed signal helps to reduce reflections. The dimensions and number of these cutouts must be meticulously studied. Every through-hole structure, such as vias or connectors, must use antipads with optimized antipads to avoid any electrical contact. This optimization work must take into account manufacturing constraints, costs, as well as the limitations of routing density that affect available space.

All discontinuities are simulated under a HFSS 3D solver in order to obtain the return loss results for all links (Fig. 4) for the daughter board. They adhere to the threshold set by the 25 Gbps IEEE 802.3 standard.

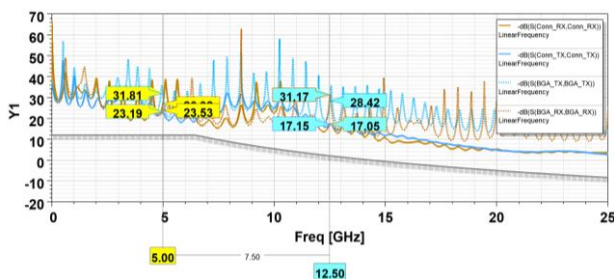


Fig. 4. Tx (blue) and Rx (brown) return losses inverted for the whole channel (dashed line for BGA side and solid line for connector side)

Cascaded link of S-parameter models enabled the evaluation of the impact of individual discontinuities on the channel's overall performance. It was observed that including

an extra S-parameter caused an average increase of -3dB in reflection losses.

B. Equalization

To meet the growing demand for high-performance PCB design, integrated circuits are constantly pushing the limits of clock speed and data rates. As the optimization of the PCB has reached its physical limits, signal processing techniques have been implemented within the transceiver to compensate for potential losses, jitter, reflections caused by poor impedance matching, etc.

Equalization balances and adjusts the signal by amplifying or attenuating it, based on the frequency. Depending on the objective, different types of equalization with specific treatments and application locations are used to compensate for channel effects, resulting in a more open eye diagram. Equalization becomes necessary as data rates increase in order to enhance the performance of the design. Below 5 Gbps, equalization on the transmitter side is usually sufficient. Beyond that, receiver-side equalization may be required [7, 8].

C. Process automation methodology

The optimization process is divided into two parts: optimizing the physical link routing and adjusting internal equalization parameters. The first part aims to meet protocol standards in order to subsequently enhance equalization results. However, when designing a new high-speed link, numerous routing changes may arise, necessitating a reevaluation of the optimization process. Manual optimization for introducing a high-speed design should remain exceptional. To enhance productivity and reduce the time required for new designs, the optimization process needs to be automated.

The primary goal is to provide a library of optimized PCB structure models according to the desired protocol and data rate. These 3D models should be flexible and not pose technological challenges for manufacturers. Using mathematical algorithms such as metamodels [9] or those based on natural behavior [10, 11], structure models will be automatically generated for a given stackup and routing class entered as constraints to be met. The algorithm's aim is to select the best candidate based on the specified objectives, such as limits for insertion loss, reflection, and crosstalk provided by standards. A dynamic link is created between the 3D structures and the algorithms through solutions offered by the Ansys suite. These algorithms will also facilitate the extraction of material properties such as roughness, Dk, and loss tangent (Df) from measurements.

The second part of automation will make it easier to select equalization parameters within the transceiver. High-speed serial link standards are long and complex to understand. Similarly, the IBIS AMI models are far from easy to use and time-consuming. Finding the optimal equalization settings in simulations can be a complex and time-consuming task. The goal will be to achieve the most open eye diagram possible while minimizing over/undershoots and meeting the required timing between signals. Each equalization technique must be properly configured to achieve the optimal combination of

parameters without degrading the signal quality through excessive optimization.

IV. TEST VEHICULE FOR VERIFICATION AND SIMULATION CORRELATION

Simulation enables the identification and resolution of potential issues in advance, thereby minimizing costs and time in the early stages of the design. Various routing configurations can be tested and compared to validate initial assumptions. However, simulations do not fully represent reality, necessitating validation of the simulated results through measurement. Measurement serves as a reference, accounting for manufacturing capabilities and tolerances that may affect signal quality. Assumptions made in simulation parameters and setup configuration must be verified.

Additionally, the test vehicle facilitates experimentation with new technologies, such as the use of three levels of stacked microvias, validating the introduction of new high-speed dielectric materials under the severe constraints of aerospace applications. Measurement also requires de-embedding techniques, involving the addition of connectors, traces, and microvias to access the desired structure. While additional elements are measured, their effects must be subtracted from the measurement to align with the simulation. Different de-embedding techniques are compared to move the reference plane further away from the cable ends, penetrating the PCB to characterize each structure. Despite the common use of Thru-Reflect-Line (TRL) calibration for de-embedding S-parameter data, its dependence on different structures raises the risk of calibration failure when fixtures are not identical. Alternative methods, less complex yet equally accurate, are available [12]. Various techniques exist to obtain real dielectric and copper properties after manufacturing. The Beatty structure [13] retrieves Dk and Df values, while a simple line and a mathematical algorithm incorporating the Huray model can determine copper roughness.

The objective is to establish a correlation between simulations and measurements to provide a reference and build confidence in simulation results, ultimately aiming to rely solely on simulation rather than costly new test vehicles. This will validate stackup, routing design, and high-speed link configuration parameters depending on each protocol and data rate.

V. CONCLUSION

To ensure the performance and reliability of electronic components and circuits, it is necessary to accurately model their behavior and simulate their performance under extreme environmental avionics conditions. Accurate modeling and simulation can be a complex task, especially as electronic components become more complex and operate at higher frequencies. Meeting these challenges requires an innovative and robust analytical methodology, combining careful design, testing, and analysis, along with continuous research and development of advanced techniques.

Several boards are currently being developed for military and civil aerospace projects, incorporating high-speed links up to 25 Gbps for very high densities, while operating under

severe environmental conditions (EMC, thermal, vibration...). Our new approach will be tested to validate its effectiveness and to determine its suitability for future projects. This will involve benchmarking of EDA tools and comparing existing and new models, which will be validated by measurements while taking into account the global electronic development cost of ownership and the proposed approach performance trade-off.

The final objective is to provide an automated and tools equipped methodology for the optimization of a new high-speed serial link using mathematical algorithms and metamodels.

REFERENCES

- [1] B. Simonovich, "Heuristic Modeling of Transmission Lines due to Mixed Reference Plane Foil Roughness in Printed Circuit Board Stackups", Lamsim Entreprises, June 2021
- [2] E. Bogatin, "The Quest for Smoother Copper May Have Reached Its Limit", Signal Integrity Journal, Feb 2020
- [3] B. Simonovich, "Practical Modeling of High speed Channels Based on Data Sheet Input", Lamsim Entreprises, EDI CON, 2017
- [4] B.Hargin, "The Printed Circuit Designer's Guide to Stackups", IConnect007, 2022
- [5] B. Hargin, "Preventing glass-weave skew", Siemens
- [6] P. Chevallier, F. KARPUS, V. RANANJASON, A.C. TANGUY "Instructions d'implantation des cartes imprimées hyperfréquences", Thales, March 2022
- [7] A. Blankman, "Advanced Course on Serial Data Jitter Measurements Jitter University Webinar Series part 7", Teledyne LeCroy, Nov 2022
- [8] C. Filip, "Overview of Channel Equalization Techniques for Serial Interfaces", Siemens
- [9] T. Most, J. Will, "Sensitivity analysis using the Metamodel of Optimal Prognosis", Nov 2011
- [10] T. M. Shami, A. A. El-Saleh, M. Alswaiti, Q. Al-Tashi, M. Amen Summakieh, S. Mirjalili, "Particle Swarm Optimization: A Comprehensive Survey"
- [11] C. Fleurent, Accromath, Editorial – vol. 14.2, Summer-Fall 2019.
- [12] S-J. Moon, X. Ye, R. Smith, "Comparison of TRL Calibration vs. 2x Thru De-embedding Methods", 2015
- [13] H. Barnes, J. Moreira, "Analysis of test Coupon Structures for the Extraction of High Frequency PCB Material Properties", 2013

Short Paper: The Security Analysis of a BLE Connected Health Device

Paul L. R. Olivier¹, Florent Galtier¹, Guillaume Auriol^{1,2}, Vincent Nicomette^{1,2}

¹LAAS-CNRS, ²Université de Toulouse, INSA

Abstract—IoT devices represent a prime target for security threats. Unfortunately, effective security practices are not widespread as they should be, in particular concerning the health sector. This paper conducts a security analysis of a connected blood pressure monitor, revealing six significant vulnerabilities. We carry out four attack scenarios to highlight the dangers they pose to its users.

Index Terms—Security, Health, Bluetooth Low Energy

1. Introduction

Wireless communications offer numerous benefits, particularly in enhancing user interaction with IoT devices. The seamless connectivity provided by technologies like Bluetooth Low Energy (BLE) has become a key selling point for manufacturers, facilitating a wide array of applications. For instance, it enables the integration of companion applications on smartphones, allowing for additional features and functionalities. This integration also enables manufacturers to optimize the device hardware to offload demanding computing tasks to smartphones or remote servers. This, in turn, improves the overall efficiency of the device. Moreover, the wireless connection allows manufacturers to introduce new features post-deployment without the need to wait for every functionality to be fully implemented before shipping the devices. This approach significantly reduces the time-to-market, ensuring a responsive product development cycle.

Despite these advantages, the connection link and the information it carries must be properly secured. Implementing robust security measures is a non-trivial task, as demonstrated by the multitude of attacks on wireless protocols. The impact of such attacks can be devastating, posing serious threats to user privacy and system integrity.

This underlines the necessity of external audits on the final product to identify potential security vulnerabilities that may have been overlooked during the development phase. Various frameworks were developed with this goal in mind [2]. Unfortunately, we still observe today poor security practices in implementing wireless communication for IoT devices.

In this paper, we conduct a security analysis of a connected blood pressure monitor. It is a health device for people with medical conditions such as hypertension, hypotension, diabetes, and other cardiovascular conditions. It can also be beneficial for people interested in prevention or lifestyle monitoring, such as pregnant women, athletes, fitness enthusiasts and seniors. Overall, such a device targets a large portion of the population and its compromise may present a risk for the user's health.

Through our study, we demonstrate the security implications resulting from an insecure implementation of the

Bluetooth Low Energy protocol. In particular, we show that the lack of authentication and integrity checks leads to device firmware tampering.

We summarize our contributions as follows:

- Reverse engineer a health device monitoring the blood pressure and conduct a security analysis of its components.
- Exploit the 6 vulnerabilities in its BLE implementation and firmware Over-The-Air update.

2. Background

Bluetooth Low Energy (BLE) is a lightweight variant of Bluetooth, dedicated to devices needing low energy consumption. Every BLE-based application using the *connected mode* is built on top of the *ATT* (Attribute Profile) and *GATT* (Generic Attribute Profile) layers [5]. Both layers define a client / server model, providing a generic solution to exchange data between devices. Specifically, an *ATT* server is a database of *attributes*. Each *attribute* is composed of an identifier, a type and a value. An *ATT* client is able to interact with this database using some requests. For example, a *Read Request* allows the client to read a given *attribute*, while a *Write Request* allows modifying the value of an *attribute*. The *GATT* level provides an additional layer of abstraction to define some *services* including *characteristics* and creates generic profiles for a given type of device.

3. Analysis

The blood pressure monitor performs three distinct measures: the diastole, the systole and the pulse. It also displays if irregularities were observed during the measurements. The device works by inflating a cuff around the wrist, measuring the pressure exerted by blood on the artery walls as the cuff deflates. This type of device is particularly useful for people with medical conditions related to the heart or interest in monitoring their lifestyle in a preventive way.

The device can be used alone, however the companion app helps keeping track of the records. The synchronization happens wirelessly when the application connects to the device via BLE. The user also has the possibility to enter the measurements manually in the companion app.

Our primary goal is to identify potential **vulnerabilities in the implementation of the wireless protocol**. It is essential to assess the device's ability to secure the data it handles, particularly when it is related to the health. Next, **analyzing the insight of the firmware** allows a better understanding in the vulnerabilities present on the device. This involves recognizing the instruction set used,

recovering the program load address, and identifying the memory layout including the I/O mapping. This can be achieved through intercepting the firmware update image for instance. Finally, the most impactful goal is to **gain arbitrary code execution** on the device. This not only serves to validate the identification and exploitation of vulnerabilities, but also demonstrates their security implications. Furthermore, it aids in comprehending and mitigating the associated risks.

Considering the goals mentioned earlier, we divided our analysis in four aspects. We started our analysis with a network reconnaissance and the companion app reverse. This allowed us to pinpoint the first flaws in the device related to its BLE implementation. Then, we looked at the hardware to get a better picture of the device’s internals. With this knowledge, we focused on the firmware to get closer to code execution. While we haven’t fully achieved this goal, the presented results provide encouraging prospects for future work.

Throughout the steps of our analysis and the discovery of novel insights, we systematically searched online for publicly available information.

3.1. Network

The device uses BLE to synchronize its records with the companion app. Using an Android Smartphone and a computer with the Mirage framework [2], we analyzed the structure of the GATT server on the device, along with its communications. The GATT server exposed several services and characteristics such as the firmware version, name and model. We identified three main manufacturer specific services for records synchronization, control data and firmware update.

We have examined the events that occur when the application connects to the device. For this purpose, we set up a Man-in-the-Middle (MitM) between the smartphone and the device. We identify two events triggered on the connection. First, the application sends the name of the current user, to which the device acknowledges good reception. The message has a total size of 20 bytes: two bytes flag, the entered name and the padding with null bytes. Second, the application queries the GATT server for information about the device status. In particular, to determine if an update is available for the device.

Then, if unsent records are stored, the device sends them in the form of `Handle Value Notifications` on a dedicated characteristic. The record messages contain a flag, the measured values (systole, diastole, pulse), a boolean indicating pulse irregularity, and the date when it was taken.

The MitM was possible because neither authentication nor encryption are employed during the communication. The application requests first to pair the device. However, we did not observe any standard pairing such as described by the BLE specification [5]. Instead, the application registers the device from its link-layer address. Moreover, it seems the device does not log any registration information, allowing anyone to connect and access saved records without restrictions. We found that the use of the term “pairing” in the application is confusing, as it could suggest this security mechanism is implemented in the communications, while it is not.

3.2. Companion Application

The blood pressure monitor comes with a companion app designed for recording measures and keep track of user health over time. For the analysis we focus on the latest version (2.2.2.5) dating from October 2023.

The APK contains two firmware images corresponding to the BLE chip. Further details about these images are provided in Section 3.4.

For GATT services and characteristics, we use the UUID to identify code sections related to the device communication. The majority of those sections are contained in the `BleService` class. We identified the code responsible for managing the firmware update, handling the device records and executing various control actions (e.g., modifying the username).

By manually inspecting the reversed code, we identified an out-of-bounds read during the parsing of health record messages. The main cause is the control exerted by the flags in the frame header over the size read, leading the application to read beyond its actual size. It may expose sensitive information or compromise the companion app’s integrity.

3.3. Hardware

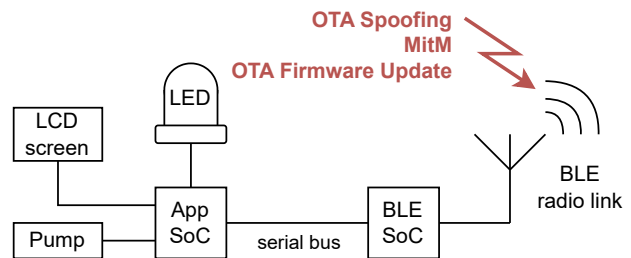


Figure 1. System architecture overview

The hardware is composed of a main PCB board which controls the LED, the LCD screen and the pump, and handles the wireless communication. In this analysis, we are more interested in the two main System on Chip (SoC). The application SoC is SH79F6488P and manufactured by Sino Wealth while the BLE SoC is a CC2541 F256 from Texas Instruments. Both SoC implements different derived versions of the 8-bits 8051 microcontroller.

The board offers several test points. Using the datasheet of the application SoC, we managed to trace its JTAG pins. However, after analyzing its signals on an oscilloscope, we hypothesize that it may be disabled.

Next to the BLE SoC, we identified the serial bus used to communicate between both SoC. A simple custom protocol is used, and we managed to partially reverse it. Each packet is formed of a header, the payload and, at the end, the checksum of all bytes except the first, modulo 8 bits. The header is composed of a preamble (indicating if it is a request or a response), an operation code and the payload size. The operation may be initiated by any of the two SoC. We managed to identify the codes corresponding to initiating a wake-up call, responding with device information (including firmware revision and model number), sending the username, and transmitting the measures (systole, diastole, pulse) along the time.

3.4. Firmware

Because we couldn't get any access to the running program on the hardware, we focused the firmware analysis on the two binaries embedded in the Android companion app. The binaries follow the format described in the Texas Instruments BLE Over-the-Air Download (OAD) [4]. The images start with 2 bytes CRC-16, the image header and continue with the firmware content. The image header follows a public format [4] composed of 2 bytes CRC-shadow, 2 bytes user-defined image version number, the image length on 4 bytes, and the 4 bytes user-defined image identification. In summary, the OAD mechanism is composed of three images: the boot image manager (BIM) and two different images called "A" and "B". The BIM maps into the 8051 interrupt vectors to intercept all resets and jump to a valid image, B or A. The image A is cut in half where the upper segment maps to $0x800$ and runs the proprietary OAD Target BLE Profile which is in charge to handle the firmware update. The image B maps to $0x4000$ and implements the BLE stack.

We explored the firmware update procedure as a noteworthy entry point for code execution. On each connection, the companion app checks whether the firmware requires an update, according to the number of the running version. The outline of the procedure is described by Texas Instruments [4]. The companion app sends the image header to the device, and upon acceptance, initiates the transfer of the firmware image. On completion, the device validates the image's CRC-16 against the one sent at the beginning and finally resets the connection. Despite being explained in the documentation [4], no encryption or signature are used for the OAD mechanism.

At the time of writing, we have not extracted or located the firmware used by the application SoC.

The firmware analysis is an ongoing task. We aim to focus on better understanding its internal work to be able to patch it and get code execution on the BLE SoC (Section 4).

3.5. Vulnerabilities

During the security analysis, we identify 6 vulnerabilities in the different components of the device.

Communication Protocol: the BLE implementation on the device suffers from vulnerabilities on the pairing, authentication and communication protocols.

- The *pairing* mechanism is *not* implemented at all, and therefore fails to establish trust between the two devices. It is responsible for providing authentication, enabling key distribution and negotiating the shared secrets.
- *No authentication* is used: both the companion app and the device do not authenticate to each other. This enables spoofing attacks for both sides.
- The *communication is not encrypted*. Data exchanged during communication is in clear text and can easily be obtained.
- The *communication does not present any integrity protection*. Data exchanged during the communication can be manipulated.

Firmware Update

- *No encryption* is used during the OAD mechanism despite being available.
- *No signature* is implemented for protecting the firmware integrity.

4. Attack Scenario

We describe 4 over-the-air attacks to demonstrate the severity of the vulnerabilities presented in Section 3.5.

The system model is similar to what was described in Section 3. It is composed of the blood pressure monitor and the companion app communicating over BLE. We do not assume the victim has the monitor device around the wrist during the attacks, because it records the measures to download them later to the companion app.

The attacker only knows public information advertised by the blood pressure monitor over BLE such as the BLE address. For the OTA spoofing and MitM, we assume the attacker does not have physical access to the target devices, hence cannot tamper the devices' operating system and firmware. For the OTA firmware update, the attacker needs a way to put the device in the update mode (long press on/off button).

The attacker has the following four goals:

- 1) Spoofing the blood pressure monitor to the companion app.
- 2) Spoofing the app to the blood pressure monitor.
- 3) Establishing a MitM between the blood pressure monitor and the app
- 4) Pushing a custom firmware on the BLE SoC

4.1. Attacks

OTA Spoofing. These attacks are straightforward as no security mechanism is implemented.

To impersonate the companion app, the attacker simply connects to the device using BLE. It is then possible to read the different GATT services, and the records made by the device.

In the same way, the attacker can advertise fake blood pressure measures knowing the BLE address, and forge fake records to poison the history in the companion app.

MitM. Since it is trivial to impersonate both the device and the companion app, a MitM attack is no more difficult. Multiple approaches can be considered, but the easiest is BTLEJuice [1].

The idea consists of first connecting with the blood pressure monitor to make it stop advertising, and then using a second BLE device spoofing its address to wait for a connection request from the companion app. Once the MitM established, the attacker can easily modify the transmitted data, such as the device records to tamper the user's health history.

OTA BLE Firmware Update. To update the firmware, the attacker needs a way to put the device in the firmware update mode. A solution for that is to long press the on/off button on the device. From that, the attacker can connect to the device and push the firmware update with the desired modifications (e.g., backdoor). Neither encryption nor signing is present in the firmware images.

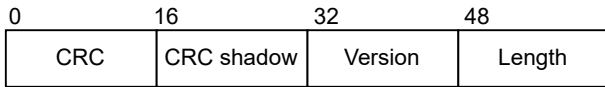


Figure 2. Firmware update image header structure

4.2. Attacks’ Impact

The severity of these attacks are high for several reasons. First, these attacks are cheap and low efforts. No security mechanisms are used to protect the link between the blood pressure monitor and the companion app. Moreover, the firmware update attack potentially allows full execution on the BLE SoC. To a greater extent, the device interacts with humans and report their health status for storing records. These records may later be used to decide whether the users should take specific medical treatment. Therefore, one could use these attacks to falsify the records, or even change the device’s behavior with a malicious update. This may lead to erroneous diagnosis and dangerous medical decisions for the user. Finally, the absence of cryptography can lead to breaches of the user’s privacy, especially concerning their medical condition.

5. Experiments

The experiments were carried out using the Braun wrist blood pressure monitor iCheck 7 BPW4500WE running the firmware 1.0.15, an Android smartphone with the companion app version 2.2.2.5, and a computer with two Bluetooth interfaces.

The Mirage framework [2] includes modules to launch standard wireless attacks such as MitM and device spoofing. In particular, we used the `ble_master`, `ble_slave`, and `ble_mitm` modules.

Application Spoofing. Any device supporting BLE can connect to the device and run commands, as no authentication and encryption are implemented in the device. For instance, we were able to connect and read the records with a BLE-debugging application such as *nRF Connect For Mobile*.

Device Spoofing. First, we cloned the GATT server of the device to replicate all the services and characteristics that the companion app may need or request. Then, Mirage is used to generate a new instance of the GATT server to which the companion app will connect.

MITM. We spoofed both roles to perform the BTLEJuice [1] MitM attack. In addition, we implemented a scenario to alter the packets containing health records, thereby tampering with the companion app history.

Firmware Update. We extended the application spoofing attack with a Mirage scenario for faking the firmware version and uploading our modified variant following the OAD process. To confirm our success, we modified strings in the firmware accessible via the GATT server, particularly the device information such as the version and model. We also had to adjust the leading CRC-16 to align with our changes before uploading the firmware. Indeed, as represented on Figure 2, each firmware file begins with a CRC, computed over the whole file [4].

6. Countermeasures

The presented attacks rely on the lack of basic security mechanisms such as authentication, integrity and encryption. The main countermeasure against the spoofing and MitM starts with the implementation of a secure pairing procedure¹. Additionally, other protections described by the Bluetooth Core Specification [5] Volume 3 Part H and Volume 6 Part E, should be applied. The firmware update process should follow the BLE OAD guidelines [4] to use firmware image encryption and signing.

7. Conclusion & Future Work

We performed a security analysis of a connected blood pressure monitor and showed that unfortunately, it suffers from several serious vulnerabilities. Some of them arise from the fact the device does not implement the security measures recommended by the BLE specification [5]. Sadly, this is not an isolated case in the more general world of IoT devices. Indeed, as shown in several works [6], [7], a huge majority of recent devices do not implement secure pairing, and by extension encryption, or contain vulnerabilities. This issue, especially in health-related devices, is a big concern for the users’ safety. Moreover, it has been shown through the last years that security has not been a priority for manufacturers, and despite evolution in the standards, most devices remain unsecure to this day. A way to solve this problem would be to make it compulsory to implement appropriate security mechanisms, for instance by the mean of a certification that all connected objects should undergo.

To further our analysis on this device, we plan to continue the analysis of the BLE firmware to reach the third goal on code execution. For instance, inserting a backdoor or a function which modify the records before sending them by BLE to the companion app. We also have to recover and analyze the application firmware responsible for displaying the measures on the LCD screen and commanding the pressure exercised on the wrist by the pump. Another possibility is to look for deeper vulnerabilities in the firmware by using dynamic analysis techniques. For example, the serial bus could be fuzzed, or the firmware could be rehosted [3] in an emulator allowing full control over its state.

Responsible Disclosure. In adherence to responsible disclosure practices, the vulnerabilities identified during the course of this research were reported to the respective vendor on two separate occasions, in January and March 2024. Despite these efforts to engage in constructive dialogue towards mitigating potential security risks, no acknowledgments were received from the vendor.

8. Acknowledgment

We want to thank Sebastien Di Mercurio for his precious help in the hardware analysis. This work has been partially supported by the French National Research Agency under the France 2030 labels (Superviz ANR-22-PECY-0008 and REV ANR-22-PECY-0009).

¹. It is recommended to avoid the “Just-Works” version which relies on a fixed key.

References

- [1] Damien Cauquil. Btlejuice: The bluetooth Smart MitM Framework, 2016.
- [2] Romain Cayre, Vincent Nicomette, Guillaume Auriol, Eric Alata, Mohamed Kaaniche, and Geraldine Marconato. Mirage: towards a metasploit-like framework for iot. In *2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE)*, 2019.
- [3] Andrew Fasano, Tiemoko Ballo, Marius Muench, Tim Leek, Alexander Bulekov, Brendan Dolan-Gavitt, Manuel Egele, Aurélien Francillon, Long Lu, Nick Gregory, et al. Sok: Enabling security analyses of embedded systems via rehosting. In *Proceedings of the 2021 ACM Asia conference on computer and communications security*, 2021.
- [4] Texas Instruments. BLE Developer's Guide for Over-the-Air Download for CC254x Version 1.2.
- [5] Bluetooth SIG. Bluetooth Core Specification v5.4.
- [6] Pallavi Sivakumaran, Chaoshun Zuo, Zhiqiang Lin, and Jorge Blasco. Uncovering vulnerabilities of bluetooth low energy iot from companion mobile apps with ble-guuide. In *Proceedings of the 2023 ACM Asia Conference on Computer and Communications Security*, 2023.
- [7] Chaoshun Zuo, Haohuang Wen, Zhiqiang Lin, and Yinqian Zhang. Automatic fingerprinting of vulnerable ble iot devices with static uuids from mobile apps. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019.

Short paper: Towards Designing a Cybersecurity Testbed for Critical Industrial Control Systems

Benedikt Pletzer and Jürgen Mottok
Laboratory for Safe and Secure Systems (LaS³)
Technical University of Applied Sciences Regensburg
93053 Regensburg, Germany
{benedikt.pletzer, juergen.mottok}@oth-regensburg.de

Abstract—The rising threat of cyberattacks on industrial control systems results in an increasing demand for cheaper and more capable defense mechanisms. Our research group is therefore concerned with the development of distributed intrusion detection systems (IDS) for industrial control systems, implementing a defense in depth approach. Developing machine learning based IDS solutions is dependent on the availability of training data as well as a test environment. A common solution for these requirements are cybersecurity testbeds. This work-in-progress paper concerns the construction of a cybersecurity testbed also suited for the development of IDS with a holistic approach to monitoring information technology (IT) and operational technology (OT) networks of critical infrastructure and industries.

Keywords—Cyber security, virtual engineering and simulation, diagnosis, privacy and data protection, cybersecurity testbed

I. INTRODUCTION

The critical infrastructure sector has become increasingly dependent on the use of interconnected systems to control and monitor the operations of their facilities. This development, that has taken place over the last decades, has opened a path for malicious actors to disrupt their operations and compromise data integrity, thus putting public safety at risk. These attacks have a wide range of objectives, ranging from monetary gains over the accidental targeting of critical infrastructure by automated large scale attacks to the disruption of vital services or industries, perpetrated by state-sponsored entities. The list of potential victims is equally diverse as the attacker's potential objectives, from the electric energy grid over the transportation of other energy carriers like oil and gas to the water infrastructure. There are several prominent instances of said attacks that have become public in the past decade. This includes the alleged perennial organized attack on two American nuclear facilities, for which two Russian government officials have been charged by the United States Department of Justice [1], [2]. Another notable and even more consequential incidence was the ransomware attack on the colonial pipeline that supplies the US east coast with almost 50 % of its fuel demand. The attack forced the operator to shut down the pipeline, causing an immense impact on the US economy. The Colonial Pipeline company had to cave in to the extortion attempt and paid more than 4 million US dollars in order to restore the pipeline's operation [3]. These attacks do not only affect large providers of critical infrastructure but also small to medium-sized plants. An example of such an institution becoming a victim of cyberattacks is the Riviera Beach Water Utility incident that occurred in 2019. The operational technology (OT) of the city's water works were indirectly infected as the city's information technology (IT) was targeted with a malicious email. The computers controlling the plant's pumps and water quality testing were incapacitated as a result of the spread of ransomware [4].

The rising threat stemming from cyberattacks, demonstrated by the earlier described incidents and many more, is prompting legislative bodies and government agencies to regulate the cybersecurity of critical infrastructure providers. The German government has therefore decided to strengthen the Federal Office for Information Security (BSI) in questions of cybersecurity by enacting the IT-Sicherheitsgesetz 2.0 (IT-Security law 2.0) [5]. This law gives several new competences to the BSI including but not limited to:

- the extension of the BSI's jurisdiction to include companies of special interest to the public safety of the German state
- obligations to report cybersecurity incidents to the BSI
- the obligation to implement an intrusion detection system (IDS) in critical networks

The obligations described in the IT-Security law are only applicable to critical infrastructure providers that offer their services to more than 500,000 people. This figure, however, is expected to be incrementally lowered in the coming years, thus generating the demand for IDS suitable for deployment in smaller facilities. The development of these systems requires an environment for the generation of training data, verification, and validation of IDS without jeopardizing the safety of a real critical infrastructure facility. The development of such a testbed, based on the IT and OT of a medium-sized water supplier, is therefore the main concern of this publication.

The testbed proposed in this work aims to improve the state of the art of simulation based cybersecurity testbeds by:

- integrating superordinate Levels of a facility's network
- implementing a wider variety of industrial network protocols
- creating a water infrastructure testbed with increased attention to detail regarding the simulated process
- providing a water infrastructure testbed with a more complex network topology

This publication is structured as follows: Section II provides an overview of similar publications, which is used in Section III to identify existing research gaps in the field of cybersecurity testbeds focusing on industrial control infrastructure. Subsequently, our proposed approach to address the previously identified research gaps is described in Section IV, followed by a roundup of our intermediate results in Section V. The paper is then concluded, in Section VI, by describing future work and outlining potential applications of our proposed security system.

II. RELATED WORK

Cybersecurity testbeds can generally be divided into two groups: testbeds mirroring the target system using a down-scaled physical system and virtual testbeds based on simulations. These two approaches can be combined with hardware

in the loop (HIL) configurations to extend a virtual testbed. Physical testbeds have some advantages like the usually more realistic emulation of the targeted system, the possibility to connect IO devices, and the possibility to simulate attacks involving physical access to hardware or communication lines. Our argument, however, is that a virtual testbed is more suited for research purposes. The main criterion for this opinion is the lower cost and higher portability of said systems, allowing for fellow researchers to reproduce the testbed with ease. Virtual testbeds can be expanded upon more easily, especially if software like the Graphical Network Simulator 3 (GNS3) [6] is used. GNS3 allows for simple creation and monitoring of virtual networks, which facilitates the extension by adding additional virtual devices. The software also allows for the integration of real hardware if it can't be simulated.

Another aspect that has to be considered when designing a cybersecurity testbed are the sections of a facility that should be reflected in the testbed. For this purpose, the ANSI/ISA-95 model [7] is used to categorize an institution's network topology into Levels zero to four. Level 0 refers to field devices like sensors or actuators. These devices are usually connected to Programmable Logic Controllers (PLC) via I/O ports and are therefore not part of virtual networks provided by GNS3 or similar software. PLCs and other controllers are part of Level 1. Level 2 refers to devices that have a supervisory control function. In water supply providers, this usually includes a supervisory control and data acquisition (SCADA) system. The Level above supervisory controls, Level 3, is the manufacturing operations management (MOM), which is tasked with coordinating and optimizing processes within a facility. The top Level deals with enterprise operations like management, human resources and strategic decision-making and planning. The remainder of this Section provides a summary of similar testbeds described in literature. Key features of the described testbeds are shown in Table I.

De Brito and De Sousa [8] propose a testbed based on the SCADA and control systems of a nuclear power plant. The system was designed with the principles of relying on open-source and free software if possible and the aspiration to model a complex system with an accurate simulation. The network topology, however, is of limited scope and does not contain devices above Level 2 of the ANSI/ISA-95 model. The simulated nuclear power plant is controlled by a single PLC, which is running the open-source software OpenPLC. Besides the power plant simulation and the controller, there is a SCADA system and an attacker connected to the same switch. The SCADA system, the controller and the simulated nuclear power plant communicate via the Modbus/TCP protocol over a virtual network constructed using GNS3.

Thornton [10] describes a virtualized SCADA laboratory controlling a simulated gas pipeline via Modbus/TCP. The testbed consists of a physics simulation, control logic and a SCADA system, the latter two are implemented in Python. The physical system is simulated with a high degree of realism using the proprietary software Matlab/Simulink. The testbed is designed with the teaching and research of cybersecurity in mind. The authors provide a suite of tools to simulate attacks (device scan, denial of service, set point modification) on the network. The alteration in the network traffic caused by the attacks can be monitored using the open-source IDS tool SNORT [16]. De Brito and De Sousa [8] note that the implementation of the control logic in Python might

reduce the otherwise high degree of realism.

Texeira et al. [11] propose a SCADA system testbed intended to be used for research regarding machine learning in cybersecurity. The authors simulate a water tank connected to two pumps and a control valve. The system communicates with a PLC, a data logger, and an HMI via the Modbus/TCP protocol. The testbed has a simple network topology as well as a simple physical system simulated with a low degree of realism. Depending on the machine learning algorithms used, this might be a limiting factor if used for developing machine learning based IDS. The low attention to detail of the simulation might be explained by their intention to generate training data for the detection of reconnaissance attacks, which are not affected by the complexity of the physics simulation.

Figueroa-Lorenzo et al. [12] designed a testbed to test new security functionalities for the Modbus/TCP protocol based on a role-based access control mechanism. The network topology of the system has some advanced features that are omitted in other works, including virtual local area network (VLAN) and routing. These are, however, used to implement the aforementioned security features and do not reflect higher Levels in the ANSI/ISA-95 model.

A comparison of key aspects of the aforementioned publications can be seen in Table I. This table is the basis for the subsequent identification of aspects we aim to improve over other cybersecurity testbeds.

III. IDENTIFICATION OF RESEARCH GAPS

One striking similarity of the previously described testbeds is the omitting of the two top Levels in the ANSI/ISA-95 model, namely the devices connected to MOM and the enterprise level operations of a facility or company. These two Levels, however, should not be neglected when dealing with cybersecurity threats, as shown in the attack on the Riviera Beach Water Utility [4] and the planned attack on two nuclear power plants [1], [2]. This statement is especially true if the development of a comprehensive security solution with a security in depth approach is the primary objective. Therefore, we aim to develop a cybersecurity testbed that encompasses all relevant aspects of an industrial facilities network infrastructure, from devices that directly control physical processes to workstations used for higher level planning and business tasks. Constructing such a testbed should facilitate the development of defense mechanisms capable of detecting potential threats at all stages of a potential attack, from the delivery of malicious software to actual actions on physical processes.

Another aspect to be improved is the degree of realism that is achieved by the simulation of the physical process, as well as its processes' complexity. Both of these criteria have been covered in testbeds that emulate different kinds of physical processes like nuclear power plants as shown in [8] and [13]. The testbeds that emulate processes related to water supplies, however, are overall simpler and simulated with less detail. Another contribution of our work will therefore be a water supply related testbed that emulates a process that is more complex with a higher simulation accuracy. A testbed with these properties might facilitate the development of IDS that are capable of detecting manipulations in the application data communicated over the network.

Implementing the two former requirements, namely increasing the realism of the simulated physical process and

TABLE I
COMPARATIVE ANALYSIS OF KEY FEATURES OF RELATED TESTBED APPROACHES

Publication	System	Protocols	Complexity of Network Topology	Realism (Physical process)	Network Level (ANSI-95)
[8]	nuclear power plant	ModbusTCP	low-medium	high	L1, L2
[9]	not specified	ModbusTCP	low	-	L1, L2
[10]	gas pipeline	ModbusTCP	low-medium	medium	L1, L2
[11]	water tank	ModbusTCP	low-medium	low	L1, L2
[12]	generic	ModbusTCP	medium-high	-	L1, L2
[13]	nuclear power plant	not specified	low	high	L1, L3
[14]	water tank	ModbusTCP, Modbus	high	real water tank	L0, L1, L2
[15]	motor control	Profinet S7comm	medium	real motor	L0, L1, L2

the integration of devices on higher ANSI-95 Levels, entails improving on our last identified shortcoming, which is the size and complexity of implemented network topologies. This could be improved in all simulated testbeds listed before, to better fulfill our needs.

Most previously described testbeds seem to focus on the ModbusTCP protocol. The protocol is commonly chosen because it can be implemented using open-source software. Training functionalities like IDS on datasets that only contain one protocol, however, might limit the transferability of models trained on the dataset, especially in a field with as much diversity in terms of protocols as the industrial control sector. Integrating additional protocols is therefore a worthwhile endeavor, at the very least, to determine whether model architecture or training of machine learning based IDS depends on the protocols present in the network. The final research gap identified in this paper is therefore the limited variety of protocols implemented in previously published industrial cybersecurity testbeds.

IV. PROPOSED APPROACH

This section will outline key aspects of our approach and illustrate the reasoning behind the following design decisions. The core functionality of cybersecurity testbeds is the generation of network traffic that resembles the traffic that could be observed in a real system. It is therefore natural that one of the first decisions to be considered is the manner in which the network topology of a real system would be replicated. This could be achieved by connecting real hardware using physical switches and routers. Our approach, however, is based on focusing on simulations to achieve the key functionalities of the testbed, facilitating the replication of our work by other researchers. The network simulation platform chosen for this testbed is the open-source software GNS3 [6]. It allows for the flexible creation and emulation of complex network topologies, which can be extended by integrating real hardware through the host device's network ports. Figure 2 shows how physical devices, like PLCs, could be connected to the testbed.

To implement the control functionalities in the testbed, the open source solution OpenPLC is chosen. The PLC software can be deployed on a variety of hardware platforms including the widely used Raspberry Pi as well as Arduino microcontrollers for less compute heavy tasks. But more importantly, it is also compatible with containerization tools like Docker to enable a resource efficient integration in the testbed. Combining these two traits makes OpenPLC well suited for our approach of generating a virtualization centered testbed that can be extended with real hardware if necessary. OpenPLC also provides an IDE that allows the development of PLC software in the programming languages described in

IEC 61131-3 [17]. The most notable limitations of OpenPLC are the small in variety of supported protocols (Modbus/TCP and DNP3) and its early development stage. According to the developers, it is not yet ready to be used in production environments.

Emulating higher Levels of the ANSI/ISA-95 model [7] is achieved by integrating virtual machines running the operating system Microsoft Windows, which is typically used for tasks involving office software or other planning tasks. These devices can be separated from devices representing Level 2 and below using VLANs or subnets created by virtualized routers. Devices operated by staff are often the first ones to be affected by an attack, as techniques like social engineering create access to vulnerabilities that could not be exploited in other systems. They are also often connected to the internet directly, exposing them to a variety of threats. The main advantage of this approach is that it might allow an IDS trained using the testbed to detect patterns connected to malicious software spreading from a facility's IT to its OT network. These devices can be replaced with Linux virtual machines running open-source office software if there is no need to replicate Windows specific attacks.

As already described in Section II, most previously published cybersecurity testbeds covering water infrastructure rely on relatively simple simulations containing only a few components with a low degree of accuracy. Pumps are usually simulated by assigning fixed flow rates to them when they are switched on. This results in linearly rising/falling fill level in water if the pump is turned on/off. Sensor characteristics like signal noise are omitted, and so is hardware-specific behavior like the response of pumps to being switched on or off. Our approach comprises adapting the behavior of pumps when ramping up to conform with a PT2 system, as described by Shankar et al. in [18]. The measured signals are also superimposed with noise to match the values that can be observed in real applications more closely.

The final proposed element of our testbed approach is the implementation of further protocols to strengthen the transferability of models trained on our testbed to real applications. The OpenPLC software chosen to implement control functionalities in the testbed is, as previously stated, limited to the Modbus/TCP and DNP3 protocols. The additional protocols to be added in the first instance are OPCUA, Profinet, and MQTT because they are commonly used. These extensions are planned to be realized using the interceptor pattern [19], allowing for a straightforward implementation without the need to modify the source code of OpenPLC. This approach improves modularity by preventing a tight coupling to OpenPLC. We expect the last three design decisions to yield a testbed that is more representative for a wider range of real life systems. Thus aiding with the transfer of findings

to other testbeds as well as real life systems.

In summary, we propose the development of a cybersecurity testbed applicable to critical infrastructure with the following design principles in mind:

- virtualization and simulation focused design
- usage of open-source software whenever possible
- representing Level 2 and upwards of the ANSI-95 model
- improved simulation accuracy and complexity of simulated system
- expansion of the protocols that can be utilized in the testbed

V. INTERMEDIATE RESULTS AND INTENDED USE

Because work on our testbed is still in progress, some of the design features described in Section IV are not yet implemented. This includes the incorporation of additional protocols using the interceptor pattern, as well as the addition of real hardware equipment and other simulated components. The improvement of simulation detail is already implemented. Figure 1 shows the simulated throughput of a pump as depicted in the SCADA element of the testbed. The flow rate is overlaid with a normally distributed error, with a variance depending on the current flow rate as well as its derivative. These adaptations are made to create a signal that better matches recordings made in a facility of one of our industry partners. The step response of the pump is modelled using PT2 system [18] calculated using the Euler method. The SCADA system also visualizes other system parameters

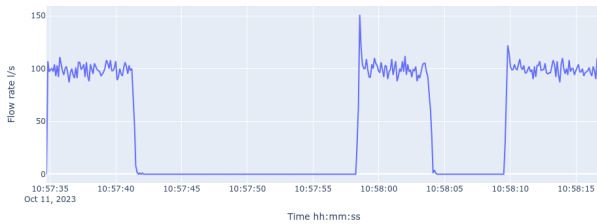


Fig. 1. Flow rate through simulated pump shown by SCADA system.

like fill levels. The provisional topology of the system in GNS3 is shown in Figure 2.

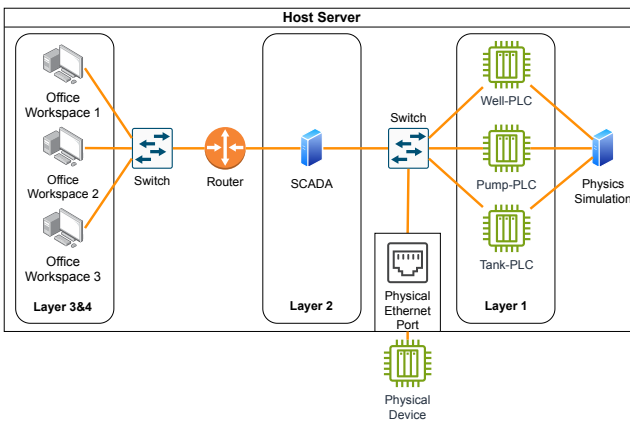


Fig. 2. Network topology of the testbed.

The overarching objective of our research project is the development of security functionalities for industrial control infrastructure that are supposed to be deployed on dedicated and cost-effective embedded devices. These devices are to be positioned on strategic edges within a facility's

network topology in order to form one distributed system. The testbed described in this publication may contribute to the development of said system in several ways. Our plan is to utilize it as a tool to generate data suitable for the training and validation of machine learning based algorithms capable of detecting potential intrusions, manipulation attempts, or other malicious actions within a facility's network. We plan on realizing this by performing cyberattacks using an automated attack framework, concurrently developed as part of our research project, and recording the resulting traffic. The distributed nature of the proposed systems might facilitate detecting malicious behavior on multiple stages of a potential cyberattack. Examples for these stages would be lateral movement within a facility's IT network, VLAN hopping to traverse security boundaries between OT and IT networks, or the actual manipulation of control systems to disrupt operations or harm the system. This defense in depth approach should raise the odds of detecting cyberattacks, especially if these attacks are complex multi-stage processes.

The primary subject of our research are critical infrastructure providers, and more specifically in this publication water suppliers. The results of our project, however, might also be applicable to other industrial networks. Systems designed corresponding to the holistic approach to intrusion detection and cybersecurity that this publication argues for, might find application in all facilities implementing networks organized in a layered structure resembling the ANSI/ISA-95 model. This includes, besides the apprehensively mentioned critical infrastructure providers, the majority of larger manufacturing plants. Protecting these assets from cyberattacks has also been a growing concern over the past decade, prompting companies to invest in their cybersecurity infrastructure. Investing in cybersecurity is voluntary for most companies, although, as already mentioned in Section I, companies of special interest to the state of Germany can be obliged to follow the same regulation. The BSI can consider companies to be of special interest due to their economic importance or their role in defense industries. All large players in the automotive, chemical, or aviation industry might thus be affected by the same legislation as critical infrastructure providers in the future.

VI. CONCLUSION

This publication describes the development of a cybersecurity testbed replicating the operations of a water supplier's IT and OT infrastructure. Our approach is based on the following design considerations: our testbed is created as a virtualization and simulation focused system in order to facilitate the scalability of our work and its replication by fellow researchers. We also utilize open-source software whenever possible for the same reason. Our approach also encompasses improving our testbeds capability in comparison to other published water infrastructure testbeds by providing a more complex and detailed physical simulation. Furthermore, we propose the inclusion of devices in the network topology that can be attributed to the Levels three and four of the ANSI/ISA-95 model. This allows us to replicate attack scenarios like the lateral movement within a facilities IT or attempts to overcome security mechanisms like VLANs or subnets. Our final contribution to the current state of simulated cybersecurity testbeds is the incorporation of further industrial protocols.

ACKNOWLEDGMENT

The presented work is part of the research project *KRITIS Scalable Safe and Secure Modules* (KRITIS³M), which is funded by the Project Management Jülich (PtJ) and the German Federal Ministry for Economic Affairs and Climate Action (BMWK) under funding code 03EI6089A.

REFERENCES

- [1] M. Graves, “United States of America v. Evgeny Viktorovich Gladkikh,” 2021.
- [2] D. Skinkard and M. Lesko, “United States of America v. Pavel Alekandovich Akulov, Mikhail Mikhailovich Gavrilov, and Marat Valeryevich Tyukov,” 2021.
- [3] J. Beerman, D. Berent, Z. Falter, and S. Bhunia, “A Review of Colonial Pipeline Ransomware Attack,” in *2023 IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing Workshops (CCGridW)*. Bangalore, India: IEEE, May 2023, pp. 8–15.
- [4] A. Hassanzadeh, A. Rasekh, S. Galelli, M. Aghashahi, R. Taormina, A. Ostfeld, and M. K. Banks, “A Review of Cybersecurity Incidents in the Water Sector,” *J. Environ. Eng.*, vol. 146, no. 5, p. 03120003, May 2020.
- [5] “Zweites Gesetz zur Erhöhung der Sicherheit informationstechnischer Systeme,” in *Bundesgesetzblatt Jahrgang 2021 Teil I Nr. 25*, May 2021, pp. 1122–1138.
- [6] “GNS3 —The Software that Empowers Network Professionals.”
- [7] “ISA95, Enterprise-Control System Integration.”
- [8] I. B. De Brito and R. T. De Sousa, “Development of an Open-Source Testbed Based on the Modbus Protocol for Cybersecurity Analysis of Nuclear Power Plants,” *Applied Sciences*, vol. 12, no. 15, p. 7942, Aug. 2022.
- [9] B. Ciprian Nicolae, “SCADA virtual test environment development,” *Electrotehnica, Electronica, Automatica*, vol. 58, no. 4, Oct. 2011.
- [10] J. Z. Thornton, “A Virtualized Scada Laboratory for Research and Teaching,” Master’s thesis, Mississippi State University, 2015.
- [11] M. A. Teixeira, T. Salman, M. Zolanvari, R. Jain, N. Meskin, and M. Samaka, “SCADA System Testbed for Cybersecurity Research Using Machine Learning Approach,” *Future Internet*, vol. 10, no. 8, p. 76, Aug. 2018, comment: E-Preprint.
- [12] Figueroa-Lorenzo, Añorga, and Arrizabalaga, “A Role-Based Access Control Model in Modbus SCADA Systems. A Centralized Model Approach,” *Sensors*, vol. 19, no. 20, p. 4455, Oct. 2019.
- [13] F. Zhang, H. A. D. E. Kodituwakku, J. W. Hines, and J. Coble, “Multilayer Data-Driven Cyber-Attack Detection System for Industrial Control Systems Based on Network, System, and Process Data,” *IEEE Trans. Ind. Inf.*, vol. 15, no. 7, pp. 4362–4369, Jul. 2019.
- [14] H. Hindy, D. Brosset, E. Bayne, A. Seam, and X. Bellekens, “Improving SIEM for Critical SCADA Water Infrastructures Using Machine Learning,” in *Computer Security*, S. K. Katsikas, F. Cuppens, N. Cuppens, C. Lambrinouidakis, A. Antón, S. Gritzalis, J. Mylopoulos, and C. Kalloniatis, Eds. Cham: Springer International Publishing, 2019, vol. 11387, pp. 3–19.
- [15] O. Pospisil, P. Blazek, K. Kuchar, R. Fudziak, and J. Misurec, “Application Perspective on Cybersecurity Testbed for Industrial Control Systems,” *Sensors*, vol. 21, no. 23, p. 8119, Dec. 2021.
- [16] M. Roesch, “Snort - lightweight intrusion detection for networks,” in *Proceedings of the 13th USENIX Conference on System Administration*, ser. LISA ’99. USA: USENIX Association, 1999, pp. 229–238.
- [17] “IEC 61131-3:2013 Programmable controllers - Part 3: Programming languages,” Feb. 2013.
- [18] V. A. Shankar, S. Umashankar, S. Paramasivam, and H. Norbert, “Real time simulation of Variable Speed Parallel Pumping system,” *Energy Procedia*, vol. 142, pp. 2102–2108, Dec. 2017.
- [19] D. C. Schmidt, F. Buschmann, and K. Henney, Eds., *Pattern-Oriented Software Architecture*, ser. Wiley Series in Software Design Patterns. Chichester [England] ; New York: Wiley, 2000, – v. 2. Patterns for concurrent and networked objects – v. 3. Patterns for resource management / Michael Kircher, Prashant Jain – v. 5. On patterns and pattern languages
Vol. [1] cataloged separately. LCCN: 96196873: Pattern-oriented software architecture : a system of patterns / Frank Buschmann ... et al. 1996 Vol. 5 written by Frank Buschmann, Kevlin Henney, and Douglas C. Schmidt.

Predictive Maintenance and Control of Memory for Availability in Safety Systems

Ashish Kumar
CTO Functional Safety
NXP Semiconductors
Bengaluru, India
ashish.kumar_2@nxp.com

Zbynek Mynar
BL SCE
NXP Semiconductors
Roznov, Czechia
zbynek.mynar@nxp.com

Arjun Muddaiah
MME Functional Safety
NXP Semiconductors
Bengaluru, India
arjun.muddaiah@nxp.com

Abstract—Memory management and availability of memory is critical for safe functioning of automotive or industrial systems. The advent of autonomous systems makes availability a critical element to achieve fail-operational state. Such complex and critical systems need to adhere to functional safety standards. Predictive maintenance and control are important aspects which helps to achieve it. This paper will discuss the current state-of-the-art of memory management and present two new architectures to ensure memory availability without any impact to the system operation.

Keywords—memory management, bit error, predictive, maintenance, control

I. INTRODUCTION

In modern complex embedded systems, reliability of memory has a significant impact on the long-term availability of systems. Memory faults are one of the most common faults in current embedded systems. Highly effective techniques used in the protection of memory against faults have the downside of being expensive due to complexity of systems and increase in memory sizes. Error Correction Code (ECC) is the most commonly used technique for runtime maintenance of memory, which can detect and correct errors in m -bit of data. But ECC alone cannot be the answer to all the relevant memory faults, as any occurrence of permanent fault in a particular memory region make its availability null and void. Memory is also deemed unavailable if a high number of correctable memory errors occur frequently in a given memory location. Such errors cause a significant impact on functional-safety-based systems, which are forced to move into a safe-state in the absence of memory availability.

For autonomous or high Safety Integrity Level (SIL) systems, when the memory location is deemed unavailable, the application would be degraded or put into a safe state, which is not desirable. This paper proposes two architectures, which improve memory availability in case of the presence of correctable and non-correctable memory errors and which are agnostic to the application during both development and operation.

II. MEMORY AVAILABILITY

Memory is a key component in embedded systems and importance of availability is vital for the system to operate safely and securely. Memory can be volatile and non-volatile in nature. Memory serves to keep information in the form of code, configuration, runtime data for processing, storage, diagnostics, and efficient retrieval of access like cache. To improve memory availability, predictive maintenance and predictive control are essential components [1].

III. PREDICTIVE MAINTENANCE

Predictive maintenance is based on digital analysis of available data and subsequent prediction of future failure of systems, sub-systems, or components. Structure of the general predictive algorithm assumed in this article is shown in Fig 1. There are many types of predictive maintenance algorithms [1], [7]. This paper concentrates on the following classes related to memory availability:

- *Corrective or run-to-failure maintenance*: The goal is to restore the functionality when a fault is detected. This type is well-established in the memory domain with the most common ECC technology used to detect and correct n number of erroneous bits of a memory.
- *Risk-based maintenance*: These algorithms aim to address risk-sensitive systems and components to ensures efficient allocation or replacement of resources to ensure safe operation.
- *Preventive maintenance*: Preventive algorithms focus on reducing the likelihood of failure or breakdown by preventing potential issues from arising. This involves monitoring of historical data and past failures to establish well-structured maintenance to avoid such failures to re-occur.

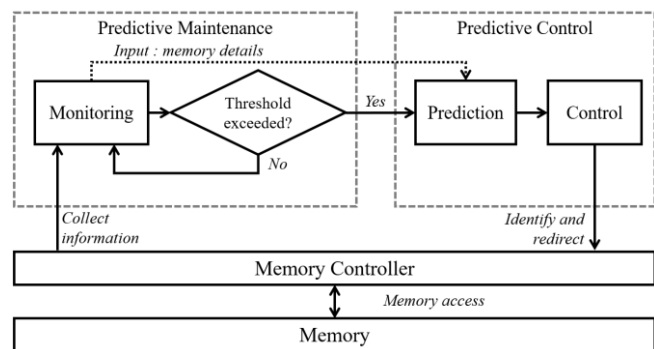


Fig 1. Concept of predictive memory maintenance and control

This paper will take advantage of preventive maintenance and risk-based maintenance principles for the memory availability to contribute to the corrective maintenance state-of-the-art. For the corrective or run to failure maintenance, there are many algorithms and measures which predict mean time to failure of memories, e.g. using memory reliability models [2], [3], [4]. This paper will take advantage of existing error detection and reporting/monitoring measures of the memory management. In this paper, we will limit to the following monitoring measures for simplification of our proposal:

- *Aging factor*: Number of read and write accesses to the memory.

- *Errors reported*: Number of both correctable and uncorrectable single-bit and multi-bit errors.

The first monitored set is the aging factor, which monitors number of accesses made to specific memory area. This set aids to determine the remaining lifetime of given memory location as it is limited by memory technology. The second monitored set is the reporting of correctable and uncorrectable errors associated to the memory. This set aids to determine of the remaining lifetime of the memory and likelihood of the memory being unavailable. For example, the DRAM memories which report correctable errors are 70-80% more likely to develop uncorrectable errors [5], [6]. Based on risk-based maintenance and preventive maintenance principles, the two monitored sets of data are the key inputs to predict the risk of failure and subsequently prevent the data present in the faulty memory from being unavailable to the system. In a simple case, the risk can be evaluated based on relation of monitored data with configured upper threshold of recorded events. If potential risk is beyond the acceptable limits, the predictive control algorithm is triggered to prevent upcoming failure and related unavailability. The predictive maintenance flowchart is shown in Fig 2.

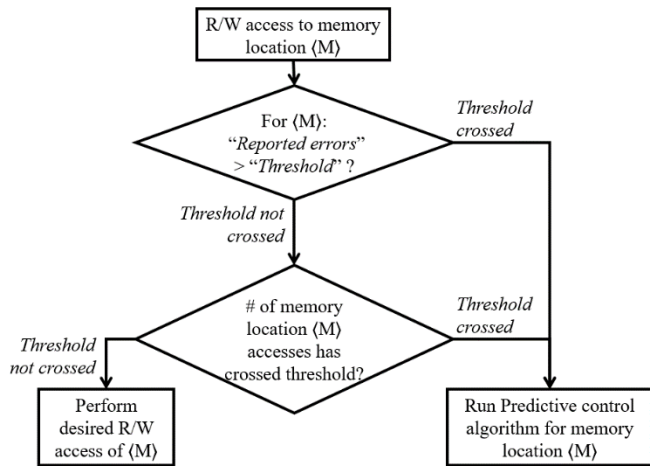


Fig 2. The predictive maintenance algorithm flowchart

IV. PREDICTIVE CONTROL

In terms of memory management, the predictive control is an algorithm, which predicts the next available and viable memory to avoid recurrence of failure, which restricts availability. Such predictive control algorithm is triggered by predictive maintenance algorithm identifying that the failure could reoccur posing a critical risk for the availability of the function as shown in Fig 2. A number of similar algorithms or systems were proposed in the past. Such proposals usually monitor number of correctable memory error reported and remap memory upon reaching configured threshold of captured events to replace faulty memory. These types of algorithms are applicable for mass storage devices, non-volatile memories, semiconductor device caches, etc. [10].

This paper offers two algorithm proposals to achieve predictive control for memory management, reusing and extending on the existing technology.

A. Proposal with memory controller

The first proposal concentrates on the Memory Controller (MC), which is assumed to be associated to a memory of distinct types, technologies (e.g., like NAND, DRAM, etc.), and volatility. The main purpose of the MC is to manage the data exchange between the controller (e.g., CPU, DMA, etc.) and the memory. The MC contains the necessary logic to perform read and write access, refresh the memory if applicable, synchronize clocks with controller or bus and so on. In this proposal, the MC maintains the above-described operational memory, but also the additional back-up memory which is used when memory region in the operational memory is predicted to fail. To check health of back-up memory itself, and viability of the proposed predictive control algorithm, the Memory Built in Self-Test (MBIST) or similar technique can be used at start-up or shutdown phase (unless the MBIST algorithm is implemented in non-destructive manner, in which case such check can be done at runtime).

The proposed *predictive control* algorithm is depicted in Fig 3. It consists of the following steps:

- The MC shall ensure any new request for the memory location say M1 is cached. If not in cache, the MC will instruct the cache controller to keep a cache copy. This copy of data ensures back-up of the data in case an uncorrectable error or multi-bit error occurs in which the correction technology cannot correct it.
- The MC confirms the requested memory M1 has exceeded the threshold of the monitored error and aging factor data. If within the desired threshold, it will continue to perform the requested operation. Else it will move to the next step implementing the *predictive control* algorithm.
- The MC evaluates and identifies the back-up memory location say M2 from the free pool. The free pool can be part of the existing memory.
- Once the free location is identified, the MC will start the redirection process. During this process, access to this location will be done only on the cached copy and not to the actual memory location. The MC configures an address redirection of the affected memory location to the new memory location. In essence, the system will continue to address the faulty memory location, but MC will internally redirect to the back-up memory. The system/application is agnostic to this redirection.
- Once the redirection process is completed, the MC instructs the cache controller to perform a cache writeback so that new memory location has the data refreshed up to date.

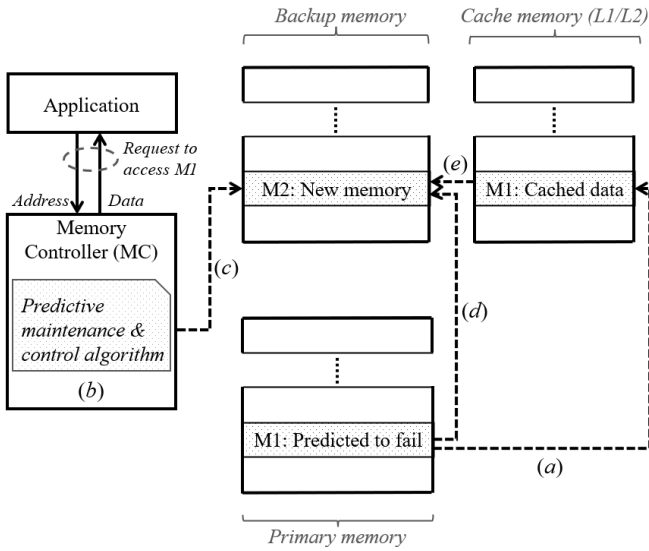


Fig 3. Predictive control algorithm proposal with Memory Controller (MC)

Compared to commonly used SoC architecture and existing prior arts, below are differences in terms of cost:

- For Step a), MC capability to inform cache controller to cache the memory location. This would add certain circuitry logic as well as latency of caching the memory location. This can happen in parallel to remaining steps to nullify the latency. In terms of timing, worst case it would be similar to a cache miss, but future accesses by application would fetch from cache improving the performance.
- For Step b), c) and d), no additional circuitry is needed as Backup memory is just another partition of the memory block which the MC can access.
- For Step b) and c), certain number of cycles would be needed by MC logic to identify the free memory location. Worst case time can be determined for the search logic. For Step d), fixed number of cycles would be needed for the logic to check if redirection is present for the requested memory access to MC. If redirection is present, then the fixed number of cycles is required to translate the requested memory location to the new memory location. The timing can be made deterministic considering the worst time taken for the above steps.

A general benefit of the algorithm is it would remain agnostic to the software application and its predictive nature, where memory remapping is permanent, thus, avoiding hard-fault situation. Existing prior art algorithms often deploy remapping techniques after the permanent faults are detected [8]. Similar prior art [9] proposes remapping techniques immediately as soon as memory detects a fault without predictive maintenance. In this paper, predictive maintenance scheme will deploy predictive control measures only if the risk is foreseen to happen. This provides more control and determinism which is critical from functional safety point of view. It will also avoid latency due to the control measures like remapping techniques applied immediately.

B. Proposal with cache controller

The second proposal shown in Fig 4 concentrates on the use of the Cache Controller (CC) with minimal involvement of MC associated to the faulty memory. Unlike in the case of the first proposal in Section III-A, the already-present cache memory itself (or its portion) will be used as potential back-up memory. When backup of memory predicted for failure is needed, it is possible to use a dedicated section (sufficient number of lines) of cache configured for back-up or re-use the cache location currently associated to the marked memory location. The CC can extend existing cache attribute or have a new cache attribute to distinguish between normal cache and back-up storage. The size of cache configured for potential backup should be selected carefully to reduce performance degradation over time with an active relocation. Also, care should be taken to ensure memory coherency, which may depend upon the cache level used as back-up. Namely, the cache needs to be in path between relocated memory and masters using it. This needs to be assured by the system designer. The cache memory can be tested by MBIST or similar technique at start-up or shutdown phase to ensure its health.

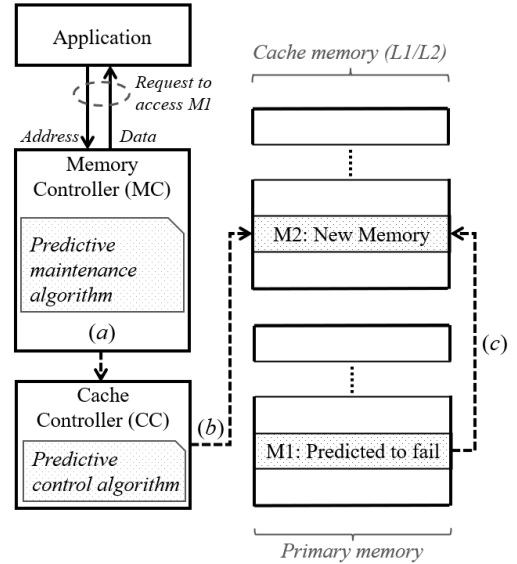


Fig 4. Predictive control algorithm proposal with Cache Controller (CC)

The algorithm follows these steps:

- The MC confirms whether the requested memory say M1 monitored data has exceeded the fault threshold. If so, it will inform the CC that the identified memory location is susceptible to failure and needs to be replaced.
- The CC identifies the M2 back-up memory location from its cache memory upstream. It is assumed, that back-up memory location M2 was configured for such potential use, is available (i.e., not already used as memory backup for another faulty location by checking the attribute), and that cache is in path between M1 and all bus-masters using it.
- The CC moves the data to the now-dedicated cache section and marks the existing cache location's

attribute as permanent for backup data storage. The CC informs MC of this change. From this point on, the controllers will always access the data from this cache location for any request of M1 memory location. In other words, the cache line(s) M2 will become permanently valid to shadow memory accesses for M1.

Compared to commonly used SoC architecture and existing prior arts, below are differences in terms of:

- For Step a), MC capability to inform CC the details of memory location and cache the data. This would add certain circuitry logic as well as additional latency to perform this action. In terms of timing, worst case it would be similar to a cache miss, but future accesses by application would fetch from cache improving the performance.
- For Step b) and c), no additional circuitry would be required as Backup memory is just another partition of the cache block which the CC can access. The cache attributes would be used to differentiate if actual cache is used or used as back-up memory.
- For Step b), certain number of cycles would be needed by CC logic to identify the free memory location. Worst case time can be determined for the search logic.
- For Step c), fixed number of cycles would be needed for the logic to check if redirection is present for the requested memory access to CC. If redirection is present, then the fixed number of cycles is required to translate the requested memory location to the new memory location. The timing can be made deterministic considering the worst time taken for the above steps.

Unlike in some of the prior art, where caches may be used for dynamic memory remapping or only remap within faulty cache itself, the use of M2 cache memory as a replacement for M1 is intended to be permanent [10], [11], [12]. This reduces available cache memory to the system over time but improves overall system availability even through device reset cycle. Other advantages of this proposal are its predictive nature, and that its implementation does not require significant changes compared to memory architecture of existing SoCs available in the market and only the CC needs to fetch content of M1 to M2 and make sure that M2 cache lines remain valid permanently (both extensions of common existing CC capabilities). Similar prior art [13] proposes predictive control measures but it performs remapping immediately when memory fault is detected, without predictive maintenance. In this paper, predictive maintenance scheme will deploy predictive control measures only if the risk is foreseen to happen. This provides more control which is critical from functional safety point of view.

CONCLUSION

Going forwards, with advent of more complex Functional Safety compliant embedded systems, current technology cannot scale up to prevent the issue of memory unavailability. Our paper proposes two predictive

maintenance and control architectures that tackle this issue and help to achieve availability of memory for critical safety systems and thus ensure fail-operationality.

Both the methods take advantage of the existing state-of-the-art and extend them to achieve the desired functionality. The first proposal describes using memory controller and a dedicated backup memory for a given memory region, when it becomes faulty. The second proposal describes using an existing cache controller and memory with minimal changes to achieve the same goal. The above methods execute agnostic of the running applications in the system. The two proposals which are compared against existing prior art has minimal circuitry changes and determined latency based on worst-case time calculations of the steps described.

REFERENCES

- [1] H. Wang, W. Zhang, D. Yang and Y. Xiang, "Deep-Learning-Enabled Predictive Maintenance in Industrial Internet of Things: Methods, Applications, and Challenges" in *IEEE Systems Journal*, vol. 17, no. 2, pp. 2602-2615, June 2023, doi: 10.1109/JSYST.2022.3193200.
- [2] S. Wang, H. Hu, H. Zheng and P. Gupta, "MEMRES: A Fast Memory System Reliability Simulator" in *IEEE Transactions on Reliability*, vol. 65, no. 4, pp. 1783-1797, Dec. 2016, doi: 10.1109/TR.2016.2608357.
- [3] P. J. Nair, D. A. Roberts, and M. K. Qureshi, "FaultSim: A fast, configurable memory-reliability simulator for conventional and 3Dstacked systems" *ACM Trans. Arch. Code Optim.*, vol. 12, no. 4, 2015, Art. no. 44.
- [4] J. Sarkar and F. -B. F. Sun, "Reliability characterization and modeling of solid-state drives" *2015 Annual Reliability and Maintainability Symposium (RAMS)*, Palm Harbor, FL, USA, 2015, pp. 1-6, doi: 10.1109/RAMS.2015.7105166.
- [5] V. Sridharan and D. Liberty, "A study of DRAM failures in the field" *SC '12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, Salt Lake City, UT, USA, 2012, pp. 1-11, doi: 10.1109/SC.2012.13.
- [6] B. Schroeder, et al. "DRAM errors in the wild: a large-scale field study", in *Proc. Measurement and modeling of computer systems (SIGMETRICS '09)*, pp. 193-204, June 2009.
- [7] Cadence PCB Solutions, "Types of Predictive Maintenance" <https://resources.pcb.cadence.com/blog/2023-types-of-predictive-maintenance> (accessed Nov. 15, 2023).
- [8] S. -K. Lu, S. -C. Yu, C. -L. Hsu, C. -T. Sun, M. Hashizume and H. Yotsuyanagi, "Fault-Aware Dependability Enhancement Techniques for Flash Memories," in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 3, pp. 634-645, March 2020, doi: 10.1109/TVLSI.2019.2957830
- [9] G. Mayuga, Y. Yamato, T. Yoneda, M. Inoue and Y. Sato, "An ECC-based memory architecture with online self-repair capabilities for reliability enhancement," *2015 20th IEEE European Test Symposium (ETS)*, Cluj-Napoca, Romania, 2015, pp. 1-6, doi: 10.1109/ETS.2015.7138734.
- [10] J. Li, L. Xiao, L. Li, H. Li, H. Liu and C. Wang, "A Write-Buffer Scheme to Protect Cache Memories Against Multiple-Bit Errors" in *IEEE Access*, vol. 10, pp. 89000-89010, 2022, doi: 10.1109/ACCESS.2022.3198989.
- [11] J. Wang et al., "Exploring Variation-Aware Fault-Tolerant Cache under Near-Threshold Computing" *2016 45th International Conference on Parallel Processing (ICPP)*, Philadelphia, PA, USA, 2016, pp. 149-158, doi: 10.1109/ICPP.2016.24.
- [12] S. Park and J. Kim, "On-Die Dynamic Remapping Cache: Strong and Independent Protection Against Intermittent Faults" in *IEEE Access*, vol. 10, pp. 78970-78982, 2022, doi: 10.1109/ACCESS.2022.3192879.
- [13] D. W. Kim and M. Erez, "RelaxFault Memory Repair," *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, Seoul, Korea (South), 2016, pp. 645-657, doi: 10.1109/ISCA.2016.62.

Integrating operator's cognitive profile for dynamic and human-centric adaptation of industrial processes

Eric Armengaud
Armengaud Innovate GmbH
Graz, Austria
eric@armengaud.at
ORCID: [0000-0003-4958-7562](https://orcid.org/0000-0003-4958-7562)

Yayoi Sakaki
Project Ipsilon, B.V.
Rotterdam, The Netherlands
sakakiyayoi@projectipsilon.com
ORCID: [0000-0003-4085-0483](https://orcid.org/0000-0003-4085-0483)

Rosen Dimov
emotion3D GmbH
Vienna, Austria
rosen.dimov@emotion3d.ai

Thomas Novak
UAS Upper Austria
Steyr, Austria
thomas.novak@fh-steyr.at
ORCID: [0009-0007-9329-5993](https://orcid.org/0009-0007-9329-5993)

Aldo Sorniotti
Politecnico di Torino
Torino, Italy
aldo.sorniotti@polito.it
ORCID: [0000-0002-4848-058X](https://orcid.org/0000-0002-4848-058X)

Yasumi Ito
University of Yamanashi
Koufu, Japan
yasumii@yamanashi.ac.jp
J-GLOBAL ID: 200901009235962344

Abstract—After several decades of automation (robotics, machine learning, AI) targeting to remove the “weaknesses” of the human, the re-integration of the human at the core of the creation process is seen as a key aspect to combine ingenuity and experience from the human together with the accuracy, speed and capability to manage large complex set of data from the robot / from the software. While solutions for human-machine interaction and for operator monitoring do exist, to the best of the author’s knowledge none of the solution is able to create a cognitive profile of the operator (capability of the operator to assess a complex situation and correctly react in a timely manner) and adapt its behavior accordingly. In this paper, we introduce the *Ipsilon Cognitive Personality*, enabling the computation of a cognitive profile of the operator assessing possible declining sensory perceptions, processing capabilities, cognitive dysfunctions associated with dementia-causing comorbidities. Further, we combine this approach with state-of-the-art operator monitoring systems to shift from attention monitoring toward prediction of risky operation. Finally, we discuss how this combined approach can be used in automotive domain to improve cooperative, connected and automated mobility.

Keywords—human system interactions, human monitoring and error evaluation, cognitive assessment, driver monitoring systems, autonomous driving

I. INTRODUCTION

The uptake of automation, embedded software and machine intelligence has been a game changer for the last 40 years in all industries. This has enabled all kind of processes to become faster, more accurate and more repeatable, finally increasing industrial competitiveness (e.g., industry 3.0 with the uptake of IT systems and robotics, industry 4.0 with autonomous systems and machine learning), and disrupting traditional applications (e.g., consumer electronics, automotive with the advances in autonomous driving functions). At the same time, the technology does not enable (yet) the integration of the human’s expertise and ingenuity in the machine-based automation to remove the human out of the loop – the human is still required at the core of the value creation [1]. A key challenge is the design and deployment of innovative and integrative human-system interactions, both in industrial processes and end-consumer products. Human-machine cooperation, human-computer interaction or human-robot collaboration are dedicated fields of research, see e.g., [2][3][4]. Beside this, the understanding of the operator awareness to apprehend the situation – and react appropriately – is a key aspect, already integrated in European regulation in

the automotive domain [5] and allocated to a dedicated field of research [6].

According to the literature main reasons for road accidents are distraction, fatigue and aggressive driving style [7], leading to the development of solutions to monitor different states categorized among 5 domains : drowsiness, mental workload, distraction, emotions, and under the influence of alcohol or drugs [8]. The target is to provide support along the three core activities during the driving process, which are situation awareness, timely decision-making and action delivery. The sensing technology is typically integrating one or a combination of the approaches listed below [7]: (a) Electroencephalogram (EEG) for brain activity and driver’s fatigue; (b) Electrocardiogram (ECG) to measure electrical activity of the heart and provide indication on driver’s stress or fatigue; (c) Electrooculography (EOG) to monitor eyes’ movement and provide indication on attention and fatigue; (d) Electromyography (EMG) to monitor muscle activity to provide indication on fatigue; (e) Electro-Dermal Activity (EDA) or galvanic skin response (GSR), respectively (f) skin temperature (ST) to provide indication on fatigue and stress. While some of the approaches can be efficiently onboarded as embedded systems to fit the needs from industrial environment (e.g., EOG, GSR), for other approaches (e.g., EEG) no embedded solutions are available and alternative approaches are required.

These approaches are typically focusing on the physiological indicators of the driver, and making the assumption of appropriate operator’s brain operation with respect to (a) reaction to stimuli, and (b) decision making. In other words, it is assumed that the operator is able to correctly apprehend any kind of situation, take the correct decision and react in a timely manner. While root causes such as operators’ fatigue and inattention are already well addressed, other aspects impacting the long-term cognitive abilities such as declining sensory perceptions, processing capability cognitive decline associated with dementia comorbidities, are yet taken into account. Regarding the automotive domain, different tools exist (e.g., [9]) to identify the point in time when the person is not able to drive anymore. While such tools are valuable from a road safety perspective, these approaches do not consider the fluctuation of cognitive responses depending on different aspects such as time, environment, mood of the driver. Further, this approach focuses mainly on the person, and does consider the support from the driving assistance systems to improve the overall reaction.

In the following, we introduce a solution for individual cognitive profiling (*Ipsilon Cognitive Personality*) and discuss how innovative driver monitoring systems (*camera-based in-cabin monitoring solution from Emotion3D*) can be enhanced by the pre-collected data of cognitive profiling. The target is to make estimation of the probability of appropriate and timely decision-making and attention level of the operator. This information can be then forwarded to the industrial process for dynamic adaptation. The following of the paper is organized as follow: In Section 2, the approach for individual cognitive profiling and enhancement of operator monitoring solution is presented. In Section 3, possible applications in the context of connected and collaborative automated mobility are addressed. Finally, Section 4 present an outlook of possible next steps for this research stream.

II. OPERATOR MONITORING AND COGNITIVE PROFILING

A. Cognitive Personality

Ipsilon Cognitive Personality has been developed based on its clinical version which is digital cognitive assessment. It presents high user acceptance by using gamified data collection with simulated piano playing (see Fig. 1), which is highly associated with executive functions, quasi-simultaneous decision-making from visuospatial task recognition. Music notes function as spatial-motor instructions, and users are to encode them to physical locations on the simulated keyboard in the app. It collects time-stamped finger tapping responses and their accuracy on location as primary data points. As a Cognitive Personality assessment, it uses the same data points as the clinical version but uses different analytic method to evaluate impulsivity and adaptation skills of the users besides the mentioned cognitive functionalities.

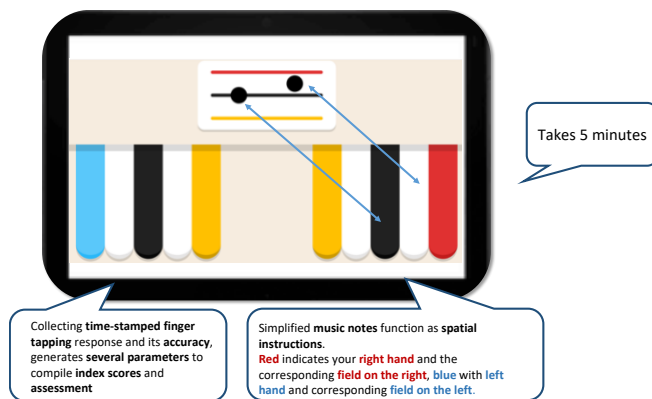


Fig. 1. The Ipsilon system: data collection by game playing

The target is to provide detailed feedback over attention, processing speed, impulsivity and learning capability of the operator to adapt the industrial process and the human-machine interface for customized interactions, leading to improve safety and inclusiveness of the operation. Especially, 11 profiles – categorized under 3 main pillars – can be identified: Pillar 1: *Low learning*, with the profile “LD” for learning dysfunction suspected, remedial and motivational training highly recommended; Pillar 2: *Moderate*, with the profiles “I1”: impulsive trait observed with consistent error rate; “E1”: error-dependent, repetitive learning type, slow improvement; “H”: consistent high error rate, potential lack of motivation and/or low learning capability; “S1”: slow learner, inconsistent response accuracy with longer response time logs; “S2”: slow learner, long learning curve expected, but

will improve; Pillar 3: *High performance*, with the profiles “I2”: impulsive but high performer, some variability in response errors observed; “E2”: error-dependent with repetitive learning type, fast improvement, “F”: fast and consistent learner and performer; “A” fast consistent learner with random inattention logs; and “C” cautious learning behavior, may take time but with high accuracy / quality work expected. The pre-collected data of cognitive profiling (gathered through regular operation of this check) will support on one side the operator to increase his/her awareness in his/her own skills, and on the other side will support the real-time driver monitoring system to better customize the vehicle reaction to the specific operator needs. Ongoing study is currently exploring the correlation between this approach and the widely accepted Trail Making Test [10], which provides information about visual search speed, scanning, speed of processing, mental flexibility, and executive functioning, and is sensitive for cognitive impairment.

B. Camera-based in-cabin monitoring solution

In-cabin monitoring solutions, as such provided by Emotion3D for the automotive domain, analyses human characteristics and action in real-time inside vehicles via cameras. This enables innovative safety, user experience and automation capabilities and addresses nowadays as well as future occupants’ needs. The core of Emotion3D’s system is its ability to integrate data from multiple sensors, including EEG, ECG, EOG, EMG, EDA, and skin temperature, each chosen for their proven effectiveness in monitoring specific physiological indicators of a driver’s state. For instance, EOG sensors are adept at tracking eye movement, offering precise data on a driver’s level of attention or fatigue, while ECG and EDA sensors provide invaluable insights into stress levels and emotional state. Recognizing the challenges of deploying certain sensors, such as EEG, within the automotive environment, Emotion3D employs a strategic approach to data integration. The advanced AI algorithms synthesize data from more easily onboarded sensors like EOG and EDA and enable the creation a comprehensive picture of the driver’s state without the need for invasive or impractical equipment. This integrated data approach allows Emotion3D to move beyond mere detection of physiological states and thus offer predictive insights and proactive safety interventions. By understanding the nuances of how various factors (to illustrate, time of day, environmental conditions, and the driver’s emotional state) interact and affect driving behavior, the system can anticipate potential safety risks before they escalate into serious safety incidents. Furthermore, Emotion3D’s technology enhances the synergy between driver and vehicle through its advanced analysis. By feeding real-time data into the vehicle’s assistance systems, Emotion3D enables a more adaptive and responsive driving experience. If the system detects a decline in the driver’s alertness or an increase in stress levels, it can trigger adaptive responses from the vehicle, such as alerting the driver, adjusting driving assistance features, or in critical situations, taking preventive actions to safeguard the occupants.

C. From mental workload indicator toward cognitive assessment

Physiological indicators exist to address mental workload, and include EDA (galvanic skin response), EEG or EOG (pupil diameter) [8]. At the same time, they are difficult to implement as embedded systems in an industrial context (e.g., EEG), respectively provide immediate information without

clear baseline (e.g., pupil diameter relying on EOG) which represent a challenge in terms of assessing the immediate risk related to mental load. The opportunity by combining pre-collected data of cognitive profiling with real-time driver monitoring systems is to refine the understanding of the driver status, finally providing more comprehensive assessment about the immediate risk associated to the current situation and the capability of the driver to appropriately react to the given situation. Hence, the outcomes of the cognitive profiling test add a degree of interpretation to the monitored data. The test can be administered up to about a month prior to the time when the real-time data is collected. In our clinical studies, the profiling data has been very consistent and does not vary up to almost a year unless there is any cognitively altering event (e.g. stroke, diabetic onset, contusion to the head). The cognitive profile provides information about visuospatial processing skill, which is the de-facto testing criterion to assess elder driver's cognitive fitness. Our value proposition is that the gold standard tests ("Trail Making Test" [10]) need to be administered by a qualified clinician (e.g. nurse, physician), but our solution functions as self-care digital cognitive assessments and does not require clinicians to be used as long as it is in the research setting at this time. We are in the process to file for CE Mark and other regulatory certifications to be able to operate as clinically qualified tool. Further, the integration of interior/exterior sensor fusion will enable to develop robust and responsive human-robot collaboration (HRC) environments. This could improve the adaptability of the HRC systems to the real-time cognitive state of the worker and external environmental factors. To reach this target, data from the cognitive personality assessment will be fused with data collected from the cameras. We exploit latest advances in Machine Learning (ML) in an innovative and novel way, following the multi-layer concept of neural networks for dividing complex problems into simpler sub-problems but using highly simplified architecture and low-complexity classifier representations (e.g. random forests). This delivers ML learning algorithms at a fraction of computational resources (estimated 5-10%), memory (less than 20 MB compared to 300+ MB) and effort while still generating high-quality outcomes. This leads to outstanding high accuracy and robustness even under challenging sensing conditions and allows the execution of accurate operator awareness recognition solutions on low-resource embedded CPU platforms.

An important enabler for industrialization of the solution will be the development of AI solutions in compliance to European regulation. Especially, the development of unbiased and ethical AI will be key to ensure that the cognitive personality trait analysis and operator monitoring systems are inclusive and ethically sound, finally ensuring optimal protection and support for every worker, regardless of their background. Ethics and user acceptance are key issues, too. Our studies have been done according to IRB ethics committees' approvals and under their strict protocols. Surveys conducted with the clinical studies suggested the high acceptability of our solution among users as non-stigmatizing, easy-to-access gamified solution for cognitive profiling/assessment alternative. The proposed solution is a digital / computerized cognitive assessment aid per FDA guidelines (similar with EU-based guidelines) being prepared to file for their certification process. The limits with such medical diagnostic tool to be implemented in the driving fitness assessment can be the early cognitive impairment detection of

elder drivers at large numbers, meaning that there may be a need for regulatory and systematic exploration on how to categorize those who are at borderline of mild cognitive impairment (precursor to dementia) and how to set the guidelines with them for their driving fitness and support.

III. EXPLORATING THE POTENTIAL: ROAD TRANSPORT

In 2021, road accidents were responsible for the unnecessary loss of 19,800 lives of EU citizens [11]. Analyses suggest that human error is an element in the causal chain of most road crashes [12], and a detailed study on accident causation factors on European streets [13] showed that 25% of the analysed accidents were caused by temporary person-related functions (influence of alcohol, drugs or medication, distraction and physiological stress). The introduction of advanced driver assistance systems and autonomous driving functions is expected to reduce the number of required human interventions, and ultimately the number of collisions [14]. Considering the fact that connected and fully automated driving (SAE level 5) will not be completely deployed before 2040 [14], complementary solutions are required to monitor and predict driver's fitness, to trigger appropriate reactions and finally minimize the number of hazardous situations. Automated driving (AD) is the most promising but also the most challenging area for innovation in the automotive industry in the coming decades. This is confirmed by a number of roadmaps and action plans issued by policy makers and government agencies [15] [16]. In Table 1, different use cases identified by the 5GAA automotive association [17] related to cooperative, connected and automated mobility (CCAM) are revisited. The idea of CCAM is to use as many information as possible to drive safely, efficiently and with a high degree of comfort and user acceptance. The term "connected" implies that data is exchanged among vehicles, but also between a vehicle and the road infrastructure. Referring to the use case in Table 1, data from sensors mounted at an intersection is transmitted to the vehicle via so-called collective perception messages (CPM) [18]. The data elements are a source of information for decision making in the CCAM functions in the vehicle. By introducing the cognitive personality trait analysis, the decision-making process receives additional parameters in the CCAM functions, illustrated in the following by the potential of the proposed approach.

The specificities of professional drivers (e.g., logistic operators) needs to be highlighted in this context. Hence, in contrary to normal drivers, the intensity of driving (8 hours a days) and thus the experience is significantly higher than standard drivers. At the same time, the regulations are more strict (e.g., maximum driving time without a break / per day) and comprehensive monitoring solutions are state of practice (and well accepted). The group of professional drivers is quite heterogenous. Looking at professional truck drivers, numbers show that 51% of all accidents on the road with HGV (heavy good vehicles) are caused by distraction. Additionally, 8% of all accidents result from fatigue and drowsiness. More than 80% of all accidents are caused by human errors even for professional continuously trained truck drivers [19]. Today's HGVs are equipped with a lot of assistance systems provided by OEMs and for route planning in logistics operation. In [20] a future solution called semi-automated platooning is discussed with the result that the traffic safety and energy efficiency can be increased substantially. To sum up, a long transition period can be expected until automated driving is widely rolled out. However, fewer truck drivers are available

TABLE I. TABLE TYPE STYLES

Road situation	Tailored driving function	Expected impact
City driving – Vulnerable Road User (VRU) detection	Automated identification of (VRUs) and warning	Higher awareness on the VRUs to trigger warning function in case of collision course while taking into account driver’s expected reaction time
Adaptive cruise control operation	Distance between ego and lead vehicle	Increase comfort and user acceptance by profiling vehicle behavior in a similar range as of the driver’s behavior
Vehicle approaching a turn	Pre-emptive braking control	More conservative behavior depending on the assessed cognitive impairment level of the driver
Emergency cornering scenario	Next-generation stability control system	More invasive interventions of the direct yaw moment controller to track the road curvature, depending on the driver’s condition

being under increasing pressure. Consequently, solutions that monitor the driving behaviour and consider the current cognitive profile of the driver are highly desirable.

An additional societal aspect to consider ageing societies with the uptake of advanced driving assistance systems and autonomous driving technology, new set of problem may arise with the risk of greater numbers of dementia onset due to the reduced, practice-able cognitive tasks, such as driving. The realistic impact of either lack of driving opportunities for the elders and/or the high level of cognitive assistance from the automated driving systems may impact the public health on cognitive wellness among older adults. The impact on public health needs to be put into consideration while addressing safer and more inclusive road transport in a 360° perspective.

IV. CONCLUSION AND OUTLOOK

The usage of the cognitive personality is introducing a new layer on operator monitoring system by intertwining psychology and neurology. It enables to create a relationship between the current situation and the likelihood for correct and timely reaction of the operator, respectively to predict the cognitive responses/ tendencies associated with the operator’s personality. This information shall be used for dynamic adaptation of the process (a) to keep the operator somewhat relaxed (not stressed), but also attentive (not sleepy/ drowsy), respectively (b) to trigger appropriate, customized countermeasures in case the cognitive profile of the operator is likely to lead to hazardous situation for the given situation. We believe that a more human-centric approach is required by combining the expertise and ingenuity of the human with the accuracy and analytic capability of the machine. We expect that such approaches will increase safety (less hazardous situations), increase user acceptance for automated functions (higher usage rate), and increase inclusiveness (being able to include higher range of population, either as worker for an industrial process, or as user of a product).

ACKNOWLEDGMENT



This project has received funding from the European Union’s Horizon Europe research and innovation programme under grant agreement No 101138110. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or CINEA. Neither the European Union nor the granting authority can be held responsible for them.

REFERENCES

- [1] European Commission, Directorate-General for Research and Innovation, Breque, M., De Nul, L., Petridis, A., Industry 5.0 – Towards a sustainable, human-centric and resilient European industry, Publications Office of the European Union, 2021, <https://data.europa.eu/doi/10.2777/308407>
- [2] C. Yang, Y. Zhu and Y. Chen, "A Review of Human–Machine Cooperation in the Robotics Domain," in IEEE Transactions on Human-Machine Systems, vol. 52, no. 1, pp. 12-25, Feb. 2022, doi: 10.1109/THMS.2021.3131684.
- [3] J. Gupta, M. Ahuja and N. Garg, "A Review on Human-Computer Interaction (HCI)," 2022 2nd International Conference on Innovative Sustainable Computational Technologies (CISCT), Dehradun, India, 2022, pp. 1-6, doi: 10.1109/CISCT55310.2022.10046656.
- [4] U. E. Ogenyi, J. Liu, C. Yang, Z. Ju and H. Liu, "Physical Human–Robot Collaboration: Robotic Systems, Learning Methods, Collaborative Strategies, Sensors, and Actuators," in IEEE Transactions on Cybernetics, vol. 51, no. 4, pp. 1888-1901, April 2021, doi: 10.1109/TCYB.2019.2947532.
- [5] Regulation (EU) 2019/2144 Of The European Parliament And Of The Council of 27 November 2019 concerning "on type-approval requirements for the general safety of motor vehicles, their trailers and systems, components and separate technical units intended therefor"
- [6] D. Yi et al., "Implicit Personalization in Driving Assistance: State-of-the-Art and Open Issues," in IEEE Transactions on Intelligent Vehicles, vol. 5, no. 3, pp. 397-413, Sept. 2020, doi: 10.1109/TIV.2019.2960935.
- [7] Khan MQ, Lee S. A Comprehensive Survey of Driving Monitoring and Assistance Systems. Sensors. 2019; 19(11):2574. <https://doi.org/10.3390/s19112574>
- [8] Halin A, Verly JG, Van Droogenbroeck M. Survey and Synthesis of State of the Art in Driver Monitoring. Sensors. 2021; 21(16):5558. <https://doi.org/10.3390/s21165558>
- [9] Centre for aging + Brain health innovation: "After the Dementia Diagnosis: When to Stop Driving: One-of-a-kind Canadian tool helps to make the hard decision", [Link](#)
- [10] R. M. Reitan, R. M. (1955). The relation of the trail making test to organic brain damage. Journal of Consulting Psychology
- [11] European Commission, Mobility and Transport, "2021 road safety statistics: what is behind the figures?", https://transport.ec.europa.eu/2021-road-safety-statistics-what-behind-figures_en
- [12] ERTRAC, Safe Road Transport Research Roadmap Towards Vision Zero: Following the Safe System Approach, Nov 2022, <https://www.ertrac.org/wp-content/uploads/2022/07/ERTRAC-Safety-Roadmap-2021.pdf>
- [13] Thomas, P., Morris, A., Talbot, R., & Fagerlind, H. (2013). "Identifying the causes of road crashes in Europe. Annals of advances in automotive medicine". Association for the Advancement of Automotive Medicine. Annual Scientific Conference, 57, 13–22.
- [14] ERTRAC, Connected Automated Driving Roadmap, Feb. 2022, available at <https://www.ertrac.org/wp-content/uploads/2022/07/ERTRAC-CCAM-Roadmap-V10.pdf>
- [15] Commission, E. (2016). Roadmap on Highly Automated Vehicles - Gear 2030 Discussion Paper.
- [16] Council, E. -E. (2017). Automated Driving Roadmap.
- [17] 5GAA Automotive Association – a visionally roadmap for advanced driving use cases, connectivix technologies and radio spectrum needs, Version 2.0, Nov 2022, available at <https://5gaa.org/content/uploads/2023/01/5gaa-white-paper-roadmap.pdf>
- [18] ETSI TS 103 324, Intelligent Transport System (ITS); Vehicular Communications; Basic Set of Applications; Collective Perception Service; Release 2. v2.1.1, 06/2023.
- [19] Statistik Austria. Verkehrsunfallstatistik 2014–2018. 2020
- [20] Energy-Efficient and Semi-Automated Truck Platooning. A. Schirrer, A. Gratzner, S. Torman, S. Jakubek, M. Neubauer, W. Schildorfer [Ed.]. Lecture Notes in Intelligent Transportation and Infrastructure. Springer. 2022.

Session Tu.3.A

ML/AI Certification

Tuesday 11th June

15:00

–

Auditorium

Perspectives on AI-ML Safety Assurance

Emmanuel Ledinot
emmanuel.ledinot@thalesgroup.com

Jean Gassino
jean.gassino@irs.fr

Amina Mekki-Mokhtar
amina.mekkimokhtar@ansys.com

Philippe Quere
philippe.quere@stellantis.com

Franck Serratrice
franck.serratrice@renault.com

Olivier Appere
appere@adacore.com

Philippe Baufreton
philippe.baufreton@safran.com

Hugues Bonnin
hugues.bonnin@continental.com

Joseph Machrouh
joseph.machrouh@thalesgroup.com

Damien Chabrol
damien.chabrol@kronosafe.com

Abstract— AI-ML suffers from a reliability glass-ceiling phenomenon (e.g. $\sim 10^{-3}$ error/inference), making it incompatible with safety-criticality. Several orders of magnitude are missing. We explain why, we point to the characteristics of ML that conflict with the assurance objectives assigned to safety-critical developments. Could encapsulation of ML constituents into fault-tolerant architectures, ML development assurance, and software/hardware development assurance, altogether mitigate the gap? We argue that in spite of impressive progress of ML State-of-The-Art, the answer is negative. Drawing from Topological Data Analysis (TDA) and set-based non-linear control, we propose to supplement ML point-based specification and verification with volume-based specification and verification to meet 10^{-5} err./inf. levels, as a minimum. We outline the rationale of a new research field we name (Ultra) Reliable Machine Learning, at the confluence of TDA, statistics on manifolds, and ML safety assurance. Some cross-domain safety regulation principles guide the underlying rationale. We illustrate the methodology on image classification.

Keywords— Machine Learning, ML reliability, Safety assurance, ML assurance, latent manifold, Topological Data Analysis, persistence homology, extensional coverage analysis.

I. INTRODUCTION

Data analysis and statistics have first developed to extract synthetic information from population data as *insights* on complex phenomena (descriptive statistics). Inferential statistics then focused on explanatory models of past observations, to get predictors on some limited aspects of complex phenomena. Never until recently, had statistical estimation to address safety-critical ‘control’. We use ‘control’ in the broad sense of OODA loops (Observation, Orientation, Decision, Action), where control of physics is involved and life, goods or environment are at risk.

Machine Learning, especially Deep Learning (DL), opened a new era: unprecedented performance in machine vision and problem solving in high dimension. However, chaotic behavior exemplified by adversarial examples plagued DL [39], and is still a matter of concern. Could DL-based components, developed with extreme rigor and encapsulated in fault-tolerant architectures, deliver services that meet the reliability requirements specific to safety-critical ‘control’? This type of requirements is new to Machine Learning and data science.

The co-authors of this paper are members of the Embedded France association’s working group dedicated to analysis of safety assurance standards in all safety-related industrial domains, to contribute their evolution [24]. We investigate the case of Machine Learning in this paper since ML-dependent safety-criticality is now on the agenda of aeronautics [2] and of automotive industry. Our focus is limited to ML *reliability*, ML *verification*, and to safety assurance of ML-dependent systems.

To our knowledge, current best accuracy scores on the easiest of image classification benchmarks (MNIST) are about $2 \cdot 10^{-3}$ error/inference [40]. From system safety perspective, this reliability level is poor: one error every seven lines containing 80 digits each. To make the gap more explicit, let us assume a 50Hz input stream of digits processed by an AI-ML-dependent safety-critical vision-based controller. It would make ~ 360 generalization errors per hour, when reliability target in the most critical case discussed in this paper would be one every billion of hours.

To address this gap, [1] screened the techniques amenable to improve ML reliability. They questioned feasibility of reaching the reliability levels required by highest DAL¹s and concluded negatively. After some scoping and terminological preliminaries, we summarize this survey of reliability augmentation methods. We propose a *conjectural* explanation why the reliability enhancement attempts uniformly failed (sections II, III, IV).

Then, we discuss why software assurance will have no impact on this reliability gap (section V), and why fault-tolerant architectures will solve only the easy cases (section VI). At this stage, we conclude that for true ML-dependent safety-criticality, there is no escape from improving ML reliability by several *orders of magnitude*.

From a geometric and topological perspective on approximant adjustment, we convey intuition on how great the challenge is. Thanks to recent advances in Topological Data Analysis (TDA), we propose a research path that would control ODD² modeling, data sampling, generalization domain definition, and approximant adjustment more tightly than standard data science does today. We review some recent papers that suggest relevance of such an attempt. We compare the rationale of safety-critical software verification, with our TDA-enabled (U)R-ML verification proposal (section VIII).

¹ Development Assurance Level

² Operation Design Domain, see Road vehicles — Safety of the Intended Functionality ISO 21448 standard.

Finally, we discuss whether ML-dependent safety-critical ‘control’ could reach the ultimate reliability level of 1, i.e. *correctness*. Software engineering and assurance managed to ensure extremely high levels of quality. We compare the two domains on specification and verification.

Contribution: we propose a diagnosis on the ML-reliability plateau. We propose orientations to overcome the reliability gap by supplementing current point-based approach of data science with a TDA-enabled *volume*-based approach.

Disclaimer: The views expressed in this paper are those of the authors as members of the Embedded France Working Group on safety assurance standards. They may not reflect the opinion of their affiliations.

II. SCOPING AI-ML-DEPENDENT SAFETY

A. Systems perimeter

We address ML-dependent safety-critical systems. Since our group is cross-domain, for the rest of the paper we use the following convention: DAL A is an abbreviation of all the corresponding assurance levels in the other industrial domains. DAL A stands for DAL A (aeronautic), ASIL D (automotive), SIL 4 (railway, process industry and many domains) and class 1 (nuclear).

An ML-component would be classified as safety-critical, synonymous of DAL A in this paper, if, and only if, it were a ‘Single Point of Catastrophic Failure’ (SPCF). In other words, some error, in adverse foreseeable conditions, could lead to a catastrophic accident. DAL A assignment is architectural: no mitigation mechanism in the system architecture to prevent some failure causality chain originating from the ML-component to evolve into a catastrophic accidental scenario. We abbreviate ‘SPCF-ML’ such situations.

Our prototypical SPCF-ML example in automotive is pedestrian detection systems coupled to automatic-braking systems. See [29] for state of the art on DL-dependent pedestrian detection performance: robustness and accuracy are still a major concern. In aeronautics, inhabited autonomous urban air mobility vehicle is the example we have in mind. More generally, we consider ML-dependent vehicle control, safety-critical health-care devices, and all kinds of safety-critical operational technologies (OTs).

B. ML perimeter

We consider off-line supervised learning in high to very high input-space dimension (e.g. 10^4 to 10^6 and beyond). We exclude continuous learning and recent ML developments like transformers and LLMs. Regarding the ML-safety survey [5], we address Robustness and Monitoring. Ethics and Alignment are out of scope of this paper.

C. Machine-vision perimeter

Open world semantic scene segmentation is the natural ML/DL computer vision long-term goal. However, we do not claim supplementing such complex ML developments with TDA at first. In this paper, we limit ourselves to development and assurance rationale of a proof of concept based on MNIST³. 10^{-5} err./inf. is our first milestone to fill the reliability

gap. We present it as an illustrative example of a generic methodology expected to be progressively scaled up to ML-processes as complex as 3D scene segmentation. After MNIST [35], the planned next step is LARD (Landing Approach Runway Detection) [25]. Only then, could one conclude on (U)R-ML practical viability. MNIST and LARD have in common existence of strong priors on the data generation process that enable structured data interpretation.

III. TERMINOLOGICAL PRELIMINARIES

We need to avoid misinterpretation on terms like ‘dimension’, ‘dimension reduction’, ‘latent’ and a few more.

A. Machine learning

- *Approximant*, any function $\mathbb{R}^n \rightarrow \mathbb{R}^p$, estimator of an underlying function specified by textual requirements and labeled datasets. We use ‘ML-model’, after adjustment, as synonymous of fitted approximant.
- *Inference*, and generalization, are used as synonymous: approximant activation on some input vector not seen during the training, calibration, and testing phases.
- *Ambient space*, also named embedding space: space where spread the vectors (or points) of the datasets. Depending on the context, we use ‘ambient space’ for input only (nD), output only (pD), or input-output ((n+p)D) space. For greyscale image classifiers, n is the number of pixels and p that of classes (e.g. MNIST: $n=28 \times 28=784$, $p=10$).
- *Latent space* or latent manifold, the regions of the ambient space where the dataset points concentrate, i.e. cluster. Latent space has its own dimension named *latent dimension*, or *intrinsic dimension*.
- *Dimension reduction*. Classical interpretation is identification of the input space features that prominently condition form of the output latent manifold (projection on a lower dimensional space keeping most of information like PCA⁴). We never use this meaning. We consider ambient to latent dimensionality collapse by shifting from external to internal view of the point cloud. When continuous natural processes generate data, dimensionality collapse occurs. Physical, operational, and control laws constrain input, state and output data to concentrate in low-dimensional regions that unfold, split, curl, merge etc. in ambient space. (Manifold Hypothesis (MH) on point clouds [11]).

B. Logics

- *Extensional*, qualifies extension as defined in ‘Extension Theory’ [6], i.e. vector encoding of magnitudes for geometric and algebraic calculation. In the sequel, we regard geometric and topological analysis of point clouds in vector spaces as synonymous with ‘extensional approach’
- *Intensional*, qualifies definition of sets or objects by *symbol* sequences (logical formulas, analytical

³ MNIST is a prominent entry point benchmark in image classification community. It consists of 70000 handwritten digits elaborated by NIST in the USA.

⁴ Principal Component Analysis

expressions, characteristic predicates etc.). First-principle models are intensional characterizations of process behaviors. Structural coverage in software testing is intensional. It is hooked to programs' source or binary code symbols. Ontologies of ODDs and analytic formulation of data-augmentation processes are on the intensional side as well.

IV. ML-RELIABILITY GLASS CEILING

A. Reliability augmentation techniques

In [1], a group of researchers investigated the means to improve ML reliability. Though ML made major progress on accuracy over the last two decades (1 to 2 orders of magnitude), $10^{-3}/\text{inf.}$ is still too poor from a safety engineering viewpoint. [1] reviews quantitative reliability results obtained by model diversification, by monitoring (ODD, robustness, I/O consistency), by robustness enhancement techniques (model stability and training stability), by selective classification, by conformal prediction, and by temporal redundancy on sequences.

Their main conclusion is the following: all the methods that tried to increase reliability by redundancy of independent models, i.e. models resorting to independent approximant spaces, independent datasets and independent optimization processes, succeeded only *marginally*. Reliability stayed stuck in the range of 10^{-2} / inference instead of the expected $10^{-4} = 10^{-2} * 10^{-2}$ or even $10^{-6} = 10^{-2} * 10^{-2} * 10^{-2}$. Moreover, these techniques improved reliability at expense of significant availability losses.

B. Common Cause Analysis

Strong correlation of inference errors between independently developed ML-models, i.e. pseudo-independence between redundancies, is an experimental fact evidenced by [1]. It is consistent with [39] where evidence is given that an adversarial example designed for model1 trained and tested on dataset1 still fools model2 specifically developed to be *independent* of model1 (datasets, approximant space, and optimization process). Similarly, [38] demonstrated a limited 13% reliability progress. It is negligible from safety engineering perspective given the reliability targets mentioned previously.

Since in this paper we are going to compare ML and software engineering in the safety-critical case, we recall that [37] evidenced independence failures with N-version programming in the 1980s.

What could be an explanation? Our working hypothesis that motivates our interest for TDA-augmented ML is that complexity of the latent manifold's shape could be the common mode that correlates error occurrences between the so-called "independent" redundancies⁵.



Fig. 1. Model adjustment to a point cloud (green shape adjusted to the red spots). The dashed ellipses delineate topologically complex regions hard to fit correctly.

State space complexity of non-linear dynamical systems (attractors, curvature, holes, cavities, etc.), compelled control engineers to start by splitting it into covering subspaces where dynamics regime has some homogeneity and regularity amenable to a local linear approach. Then, they aggregate these local controllers into a unique global controller by mode switching and scheduling logics, up to complete cover of the topologically complex reachable input/state/output space. ML we consider in this paper addresses the same type of continuous data manifolds. Standard data science addresses training datasets all at once, straight away at global scale.

Possibly, the ML model redundancies used in [1] failed to adjust reliably on the same topologically complex regions. Hard-to-fit regions of input space are *problem* dependent. In other words, they are ML-model *independent*, so they can correlate any pair of redundancies. Shape of training dataset is a potential *common cause* in ensemble learning.

C. Plateauing performance

When the approximant space is defined by the solutions to $(n - 1)$ polynomial equations over n variables, the ambient space is nD and the latent space is $1D$ algebraic curves. Given k points in nD Euclidian space, finding a polynomial curve that links the k points is still an open mathematical problem [9]. By 2022, a proof of existence was published on the Web. It is under peer-review. In case of confirmation, more than a century will have been necessary to solve the $(n$ -ambient, 1 -latent,) case for an intensively investigated class of functions.

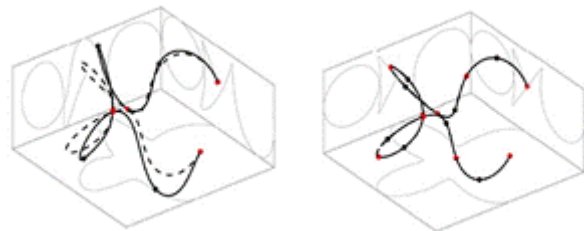


Fig. 2. 1D latent manifolds in 3D ambient space. Limiting generalization errors to very small number of occurrences requires controlling adjustment with extreme precision. Impact of "fitting" variability on the 3 projected curves (picture is courtesy of [8]) when "adjustment" varies slightly (difference between the dashed and non-dashed curves).

Admittedly, equation solving (i.e. 'exact adjustment') is of different nature than ML-model fitting. It is harder because of exactness of equation solving. However, precision-controlled fitting in high dimension is a very difficult problem as well, even if a "flexible" one⁶. We advocate that high reliability of generalization will necessitate sophisticated mathematical tools to control where and *why* generalization errors occur. Ability to explain why a generalization error occurred in order to fix it will be mandatory for DAL A ML. Any known error that potentially could be a single cause of catastrophic failure, should be eliminated to comply with ethics and regulation.

D. Zero-measure verification

Behavioral information given in point-cloud specifications is extremely poor with respect to:

- immensity of high dimensional ambient spaces,

⁵ In section X, another potential cause is considered on MNIST: labeling errors [41].

⁶ Because the inverse problem is ill-posed.

- shape complexity of input and input-output latent manifolds.

Meeting inference failure rates as low as 10^{-k} err./inf. $k \geq 5$ is highly demanding. Sample-oriented by nature, statistical functional estimation naturally relies on point-based verification. Extensional verification coverage by end of testing, i.e. the covered volume of behavior, by means of some N-point testing dataset is $N^*0=0$. Null cover. At the opposite, the nD volume of the latent input manifold over which the estimated function should generalize reliably is gigantic and nearly devoid of specification information. We illustrate the specification miss on the MNIST classification problem, and how TDA could help (section X). Worse, the generalization domain over which one should estimate probabilities of misclassification events is *undefined*. No integration, i.e. error counting, without specified integration domain, i.e. *defined* inference domain, and without *error-oracle* covering it exhaustively. Such error-oracle is named actionable specification in [30].

From safety engineering and assurance points of view, there is discrepancy between on one hand the absence of explicit input-domain definition, the gigantic space where specification misses, the limited control of adjustment, and on the other hand the extremely demanding reliability levels required to get certification approval on ML-dependent safety-critical systems.

V. FILLING THE GAP WITH SOFTWARE ASSURANCE

Could the reliability plateauing problem ($\sim 10^{-3}$ err./inf.) be mitigated by implementation of ML-models with extreme rigor, i.e. with DAL A assurance level? The reason would be, following some misconceptions about development assurance, that DAL A developments deliver high integrity software, and accordingly that high integrity software would ensure 10^{-k} failure/h reliability levels, for values of k ranging from 5 to 9, depending on industrial domains.

The goal of software assurance is to ensure fidelity of the transformation process that converts system functional specifications like ML-models (e.g. TensorFlow mathematical equations) into binary code instructions. Fidelity, also named implementation *correctness*, or *compliance* or *semantic invariance*, means ensuring *extensional* behavioral equivalence between some ML-model and its executable object code counterpart. On the intensional side, the transformation of symbol sequences is complex. Preservation of the defined behavior is at risk. Regarding reliability of inference, DAL A ensures high trust on *reliability invariance* from model to executable object code, i.e. “garbage in, garbage out”. It does not ensure reliability *augmentation* (e.g. up to 10^{-9} err./inf.) during the transformation process.

Explaining why there is no reliability augmentation provided by assured software is *not* discrediting the value of software assurance. Software assurance prevents introduction of flaws in the behavior-preserving symbolic transformation. One may find more information on the link between qualitative and quantitative aspects of development assurance in [10]. In particular, the domain-dependent relationships between reliability levels k and assurance levels (A, B, C, D) are *conventions* that associate qualitative leveling of rigor with expected reliability in case of residual faults. Assurance splits trustworthiness construction in two policy regimes (cf. section

VII). It needs some correspondence between the two for global consistency. This correspondence is not a *convertibility rule* between fault-prevention rigor levels, i.e. DALs, and reliability levels. However, return of experience over ~ 50 years demonstrated validity of these conventions.

VI. FILLING THE GAP WITH SYSTEM FAULT-TOLERANT ARCHITECTURES

We consider the case of catastrophic failure dependent on the performance premium *uniquely* delivered by Deep Learning. For pedestrian collision avoidance systems or autonomous air taxis, Deep Learning has by far outperformed the classical and certifiable algorithms of computer vision. If some classical underperforming algorithm is sufficient as safety monitor to keep controllability in fault detection-isolation-recovery phases, then the DL-dependent channel provides only performance bonus. Form safety architecture point of view there is no true criticality assigned to AI-ML.

We extensively discussed in the group whether software engineering and assurance managed over time to prove sufficient effectiveness so that SPCF software was introduced in safety-critical architectures. Answer was yes, for aeronautics, space, automotive and railway. Nuclear is the exception (DAL B at most). We have no representative of medical device industry in the group.

In aeronautics for instance, in flight control systems in particular, there are architectures, functions, and limited regions of the flight domain where a specification flaw or an implementation error may constitute a single point of catastrophic failure. DL-dependent vision-based control for air taxis or pedestrian collision avoidance will lead to true SPCF-ML constituents as surely as it was the case for software. In the “no-backup” situations that define DAL A, extreme reliability is required and even perfect reliability named *correctness*. This is the motivation of our research program proposal on TDA-enabled (U)R-ML.

VII. ELEMENTS OF ASSURANCE PRINCIPLES

We review the foundational aspects of development assurance that interact with ML characteristics in the safety-critical case. We start with the rationale that splits assurance in two policies: correctness and rareness. In the sequel, we use ‘quantitative’ objectives exclusively for probabilistic quantification of event occurrences. As an example, 100% DC coverage, though 100% is a quantity, is not a quantitative assurance objective, in our sense at least. It is a software testing termination criterion dependent on a numerical value that conditions intensional cover.

A. Correctness .vs. rareness policies

Historical perspective helps understanding the split between fault prevention/elimination on one side, and probabilistic quantification of feared failure events on the other side. The former is applied to software and hardware development. The latter is applied to physical failure modes and their cascading effects. We quote the following text from aeronautical regulation to prove that probabilistic quantitative arguments were not primal in trustworthiness demonstrations. Logical, argument-based demonstrations of safety, even when software was absent (i.e. electromechanical systems), preceded probability-based evidences.

Design and implementation correctness of fail-safe mechanisms in charge of passivating the single points of

catastrophic failures was the first and primary safety assurance objective in aeronautics. It was the origin of the fault prevention process-based assurance methods.

CS 25.1309 « Equipment, Systems, and Installations » AC251309-1B
“In the early years of aviation, airplane systems were evaluated to specific requirements: to the “single fault” criterion, or to the fail-safe design concept, which are explained below. As later-generation airplanes developed, their designers added more safety-critical functions, which generally resulted in an increase in the complexity of the systems designed to perform these functions. A safety-critical function was a function whose failure would result in a catastrophic accident. The potential hazards to the airplane and its occupants, in the event of failure of one or more functions provided by a system, had to be considered, as did the interaction between systems performing different functions. To assess the safety of a complex system—and the adequacy of system redundancy to meet the fail-safe criterion—the FAA began assigning statistical probabilities to system failures in AC 25.1309-1, dated September 7, 1982.”

Probabilistic assurance goals were introduced for the reasons explained in the verbatim, but the first accepted means of compliance were qualitative. Arguments of assurance cases were similar to that of qualitative physics applied to conservative approximations of failure propagation through system architectures. Orders of magnitudes were enough, and (causal) independence hypothesis between component and function redundancies were the primary concerns. Then, came computer-intensive probabilistic calculations and their acceptance as means of compliance (e.g. fault-tree analysis and Markov chain models).

Over a few decades, some unconscious cognitive bias spread in the safety engineering community. It consisted in reducing safety assurance goals to probabilistic ones, and probabilistic arguments to quantitative ones.

As software or hardware items, ML implemented models are deterministic artefacts. Nonetheless, as result of an engineering process they are realization of a random variable, valued by a mathematical function. The seeds of randomness are data sampling and stochastic features in adjustment algorithms. By extension, one could add as seeds of randomness, the model instability sources related to ill-posedness of the inverse problem, and addressed by the stability assurance objectives.

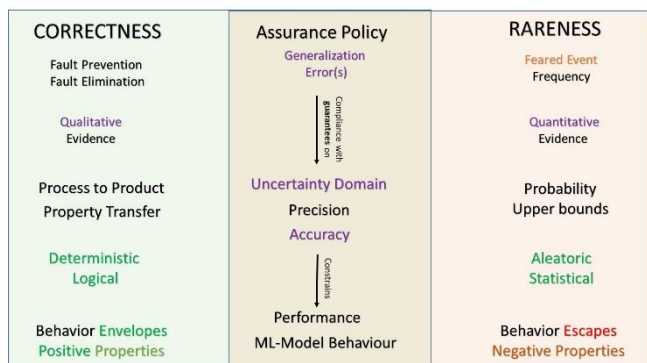


Fig. 3. Contrasting the two assurance policies. ML is amenable of both (overlay of green and amber). Preliminary to fig. 4 on status of SPCF generalisation errors.

Quantified sufficient rareness of ML-component failure modes would be the natural choice as assurance objective. We discuss this option in F. We tried to map the contrastive characteristics of the two assurance policies and their intricate relations with ML assurance in the following figure.

B. Actionable specifications

We support the analysis in [30] that singles out point-based specification as the prominent difficulty for ML safety engineering. We reuse the term “actionable”. We interpret it

as “amenable to computational evaluation” and consider it as equivalent to the ‘perfect oracle’ notion of [32]. Software (resp. hardware) testing of the implemented ML model, formal verification, and probabilistic quantification of error events, all need a computable oracle to decide whether model’s response on input vector deviates from the intended, as specified.

The specified may diverge from the intended if needs capture is not correct and complete. The specified is pivotal for the following assurance objectives:

1. *derivation* of implementation from specification,
2. *correctness* of implementation w.r.t. specification,
3. *quantification* of failure modes.

Computer-decidability (test oracles, failure-mode oracles) of the specified is necessary for both assurance regimes. There are ambiguity cases in image classification where even human-decidability is not ensured. Another source of oracle miss is lack of ground-truth, quite common in ML application to perception systems. Safety engineering and assurance are severely hampered by miss of deviation oracles. ML assurance should exclude SPCF-ML in such development conditions.

C. Implementation derived from specification

Mitigation of complexity-induced risks by decomposition of the specified, by piecewise refinement, and by progressive and traceable derivation of implementation constructs from specification traits, constitutes a cornerstone of assurance. It is a “divide & conquer” error-prevention strategy to cope with error-friendly complexity.

A second cornerstone of assurance is assessment of the small derivation steps by independent verifiers, possibly with variability and redundancy in verification methods. Traceability is the practical means to manage complexity along hierarchical decomposition paths. A by-product is diagnosability. In case of behavior deviation w.r.t. the specified, traceability-enabled backward dependence analysis enables precise localization of faults and errors. In turn, it enables fault elimination. Elimination of the known faults is characteristic of the correctness assurance regime. There would be no alternative to 100% accuracy in DAL A ML. Embedded known SPCF errors are ethically unacceptable.

ML violates the derivability and diagnosability assurance objectives of correctness policy. Approximant structure and parameter adjustment cannot be stepwise derived from training datasets. Consequently, when 100% accuracy is not reached, the root cause of fail-cases cannot be localized to enforce the error elimination policy. Correctness regime is intractable for ML, as of writing this paper.

D. No single point of failure

Regulation considers as unacceptable severe damage originating from a single specification, design, implementation, or operation error. Fault tolerant architectures are required. Since fault tolerance starts with fault detectors, on-line deviation oracles, in other words actionable specifications, are required. For ML, such actionable specifications are inaccessible on high-dimensional unstructured data like text, audio, and video signals.

E. Correctness policy

Software was regarded as a logical artefact that, in theory, could be developed without faults. By nature, it cannot spontaneously lose capabilities contrary to physical equipment. For these two reasons, standard committees applied fault prevention policy, i.e. correctness assurance to software. Safety standard committees regarded quantification of software reliability as ethically unacceptable for any safety-related development. In addition, it was deemed technically intractable in valid manner.

Like software, and contrary to physical equipment, ML model cannot spontaneously lose some capability as cause of a failure mode. They are deterministic, designed, time-invariant logical artefacts that make errors. Correctness regime should apply. However, miss of diagnosability prevents application of the “no-known-fault-left” policy.

F. Rareness policy

As seen previously, it could be an option for ML, considering the randomness sources in its elaboration process. However, it would be a paradigm shift to assimilate generalization errors to classical safety failure modes (i.e. random capacity losses). One would declare activation of preexisting flaws that are consequence of deliberate engineering choices, as equivalent to random physics-caused failures.

G. Perspectives on SPCF-ML assurance

The intent of the preceding review is to argue that there is no compelling choice of assurance policy for safety-critical machine learning. In addition, the application spectrum of ML is so large that a unique “one-size-fits-all” policy choice would be vain. Therefore, we reached consensus in our group on the following most flexible but principled rationale.

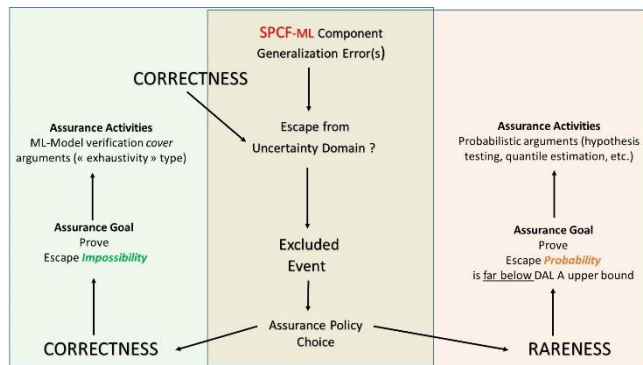


Fig. 4. Both options are sensible to some extent and missing means of compliance.

Since there is no compelling default option, our pragmatic stance is to leave the choice to the applicant, property-wise. For a given ML component, some failure modes could be assured by correctness means while others could be assured by probabilistic calculations. In our discussions, we even envisioned the case where a property could be partly demonstrated in correctness regime, and partly in rareness regime. Complexity of provably correct or ultra-reliable approximation in high dimension needs availability of any kind of well-founded verification technique.

VIII. COMPARING SW/1980s TO ML/2020s

Nearly half a century ago, software soared in embedded systems, while appearing brittle and raising concern about safety of software-intensive aircraft. In the early 80s, software-induced complexity ballooned as fast as grew the number of bugs per Kloc⁷. The foreseen “software crisis” for civil aviation lead to convention of assurance standard committees. First release of ED-12A/DO-178A was by 1982. About 40 years later, return on experience demonstrated that applying these assurance standards was effective.

ML and especially DL are following a similar trajectory: fast massive adoption by industry in spite of instable behavior (e.g. adversarial examples). Like for software, there are concerns about safety of ML-dependent aircraft or car. Automotive has been the leading industrial sector in the late 2010s. DL opened industrial viability of open- world computer vision. It made self-driving cars appear as a mid-term market opportunity. Consequently, development of ML-assurance standards started early, following ideas similar to that of proven-in-use software assurance standards. To what extent are these two histories comparable? Should we expect for ML assurance the success of software assurance?

A. Similarities

Foundations: a few decades before their respective booming industrial acceptance, both software and ML benefited from mathematical background: on computability and correctness for software (e.g. Turing, Floyd, Hoare); on statistical estimation, information and learnability for ML (e.g. Fisher, Shannon, Vapnik).

Engineering: in both cases these theoretical foundations had no immediate impact on tooling and industrial best practices.

High-dimensionality: software and machine learning share this characteristic. Curse of dimensionality to verify behavioral spaces is a common difficulty to meet the assurance requirements of the safety-critical. Safety-related embedded software has nowadays $D10^k$ input (resp. state, output) space dimensionality, with k possibly ranging from 2 to 7, and even beyond (e.g. ATM/ATC ground segment software). It is the same dimensionality order of magnitude as that of DL-based HD video streaming processes.

Extensional verification cover: it was a deep problem for software assurance. One needed a sufficiency criterion to stop IVVQ activities with DAL-dependent appropriate confidence. Structural coverage, amenable to DAL modulation, was the solution. Committees were aware that even with MC/DC coverage, *extensionally* speaking, behavioral space cover was near zero. It was the best ALARP⁸ cost/benefice trade-off at state of the art. Why then did software assurance succeed? Has near-zero extensional verification coverage the same significance for software as for machine learning?

B. Disimilarities

Point-based specification in high dimension and diagnostic inability seem to us the differentiating factors of

⁷ Kilo-lines of code.

⁸ As Low As Reasonable in Practice (risk)

ML w.r.t. software. Textual software specification are often example-based, i.e. scenario-based or use-case based. However, contrary to ML, all the examples are intended to be generalization seeds for human. Software developers generalize the examples when they formalize specifications and algorithms. Doing so, they implicitly create behavioral cells in their minds, named equivalence classes at testing stage. On the extensional side, these equivalence classes create volume-units of validity in the neighborhood of the 0-measure test cases. There is *implicit* augmentation of extensional coverage by principled code derivation and associated testing practice (i.e. requirement-based testing). Is there extensional coverage augmentation for ML, be it explicit or implicit?

IX. TDA-ENABLED (U)R-ML

We have justified why ML reliability must drastically improve to meet DAL A assurance objectives in the SPCF case. We have underlined a major difference between ML and software regarding verification cover: implicit volume-based cover for software, without any equivalent for ML. Foreseeable efficacy of ML assurance for the safety-related is likely to be far under the levels reached in the case of software.

We propose research orientations based on Computational Geometry (CG) and Topological Data Analysis (TDA) in higher dimensions [11], [28] to overcome these problems. It consists in supplementing classical statistical data science with awareness of topological complexity of datasets to support ML engineering activities like sampling, definition of In-Distribution oracles, diagnostic of inference errors, volume-based verification coverage analysis, empirical probability computation, etc.

In this section, we focus on sampling and explicit definition of the generalization domain (ID oracle). In the next and last section dedicated to the MNIST proof of concept, we adopt a broader view on use of topology.

A. Semantics of emptiness

High dimensional void is the ambient space around training and test point clouds. Emptiness around points may result either from principled choices, or from loopholes. Emptiness may be full of missing information that prevents from meeting correctness and/or reliability targets. We distinguish four types of voids:

1) Causal impossibility

Physics, scene or environment evolution laws, operational concepts or ODD constraints may prevent the generation of samples in definite regions of the input space. It leads to valid distant clusters or samples.

2) Sampling incompleteness

The sampling plan, compliant with the ODD and with the ML-model’s textual specification, may overlook some input space regions. Depending on local regularity and approximant characteristics, these sampling lacunas may or may not constitute potential risk of inference errors.

3) Designed parcimony

When variability of data is under control, sampling may be appropriately parsimonious. Energy saving, or footprint

constraints on embedded targets may also lead to local decimation of samples. In these cases, some extensive void regions are not risky.

For sampling coverage analysis, TDA should enable exploration of *dataset shape* to identify existence of unintended void regions (see fig. 5 and 10). It would consist in detection of non-interpreted large holes or cavities as potential sources of adjustment complexity and potential common modes for ensemble learning. This activity would be ML-model *independent* since it would only consider input spaces and ODDs as guide for data shape interpretation.

TDA offers a portfolio of algorithms to analyze point clouds in 2D, 3D, and in higher dimensions. We focus on persistence homology (PH) which plays a central role in TDA. It is used in ML for clustering, denoising, feature engineering (e.g. [12], [15]), and neural networks weight space or activation space analysis. We propose a new family of PH applications to machine learning whose overall goal is to overcome the reliability gap.

Roughly, PH computes a growing sequence of balls centered on each point of the dataset. For each ball radius of the sequential process named filtration, it computes the ball intersections and creates edges between the vertices that are centers of intersecting balls (see the four filtration steps of fig. 5). These edges constitute a nested mesh (simplicial complexes) that enables rigorous geometric and topological reasoning in higher dimension. They performs multi-scale modeling of point clouds. PH detects birth and death of kD-cycles, cavities and holes, as ball radius grows by discrete steps. It ends when the radius is so large that all balls intersect. Figure 5 illustrates some steps of 2D point cloud filtration.

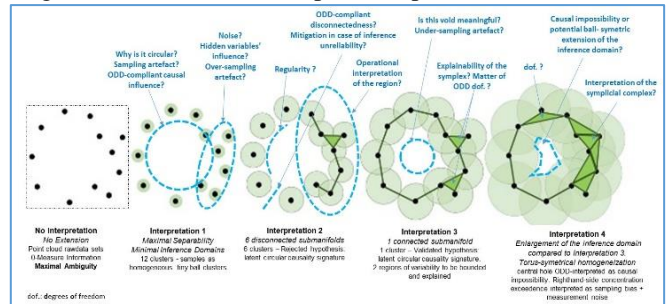


Fig. 5. Designing the inference domain, the “meaning” of the input part of training datasets. Four steps of persistence homology filtration are represented. In the upper part of the figure are examples of typical questions to interpret the filtration steps. At bottom we wrote examples of interpretation decisions that could lead to selection of a given filtration parameter.

We propose to use PH filtration as (U)HR-ML data engineering practice to design some ODD-compliant interpretation of the training and testing datasets. Output of this task would be the ID-OoD⁹ oracle of the approximant. For computational tractability, latent dimension must be far lower than ambient dimension.

B. Formal definition of inference domains

To our opinion, high reliability of approximants will require formal and executable definition of their domain (i.e. of their precondition from a formal method perspective). PH should offer means to define ID-OoD oracles in a way that does not depend on distributional assumptions or ML techniques [22].

⁹ In Distribution – Out-of-Distribution

C. Extensional verification coverage analysis

We envision PH-based construction of a latent space simplicial complex as a means to guide scrutiny of generalization reliability. Triangulated training input spaces could support tight verification coverage criteria, simplex after simplex, used as generalization cells and as candidate counterpart of equivalence classes in software engineering. We name *extensional coverage analysis* this *volume*-based verification activity. It would be the extensional counterpart of structural coverage analysis in software. Such latent-space oriented verification coverage ideas are being explored for instance in [23].

D. Contribution to ML safety assurance

We first review four applications that are independent of any ML technique. This is a distinctive advantage since assurance values independence methods in verification.

1) Model-independent applications

1. **Explicit generalization domains:** using data augmentation, tuned filtration parameters, and PH simplices, design of a simplicial complex of operationally explainable generalization cells. The aim is an ID-oracle.
2. **Designed separability:** using persistence diagrams, homology groups, and homotopy classes as topological alerts of potential hard to fit regions for classifiers (cf. illustration on MNIST).
3. **Extensional verification coverage analysis:** using PH-complexes as *covers* of generalization domains, with multi-scale resolution.



Fig. 6 Filling the ML reliability gap by enhanced verification coverage techniques. Extensional verification would ensure non-zero measure coverage, explicitly contrary to software where extension of equivalence classes remains implicit.

4. **Novelty detection:** non-stationarity tests in ML-Ops processes. TDA and Information Geometry could be used jointly to monitor datasets' shape trajectories and thoroughly diagnose risks of adjustment obsolescence.

2) Model-dependent

Research on how PH enables shape analysis of neural network activation spaces is undergoing. It has interesting potential for safety assurance as it could become in (U)R-ML engineering the extensional counterpart of structural coverage analysis and dead code elimination in safety critical software engineering. As we are close to the end

of this paper we don't elaborate any further but potential for assurance is great.

X. PROOF OF CONCEPT ON MNIST

Last section is an outline of a proof of concept we are developing to support our discussions. It is also intended to support future (U)R-ML data-science challenges. The figures in this section do not result from TDA computation results, yet. They aim at presenting some (U)R-ML goals and activities, and at conveying intuition on a method whose engineering is still to develop. Preliminary results on digits {6, 0, 9} are documented in [33], to be made public after completion on the ten digits.

A. Related work and discussion

[31] is a systematic literature review devoted to certification of Machine Learning. Comparison with software is developed. There is no mention of the N-model non-independence problem. Topological data analysis is not mentioned either. [30] is another review of the main certification challenges for safety-critical ML. TDA is addressed and advocated as a promising approach. [28] is a survey of TDA applications to AI-ML, with focus on bio-molecular engineering. In image classification, all uses of PH reported in this survey are at image level, for dimensionality reduction, denoising, feature extraction, etc. In this PoC, we use PH at dataset level, to analyze the shape of the training and testing image databases.

In [26], PH applied to MNIST is reported. It enabled reducing 784D to 28D at iso-accuracy (96.3%). On our side, we want to augment accuracy (drastically), not to save computation time and energy without accuracy penalty. In [27], a table reviewing the performance scores of top10 MNIST classifiers is given. It provides evidence that reliability is plateauing at $(1 - 3.10^{-3})$ on MNIST. We identified significant labelling errors in MNIST ($\sim 10^{-3}$ as well). It is a serious issue for (U)R-ML [41]. An ultra-reliable e-MNIST 10^7 -sample dataset is needed (see fig. 8 for ambiguity cases of digits with letters).

In [34], persistence homology filtration of the testing dataset, and abstract interpretation of the neural network are combined. Goal is verification coverage analysis and global robustness verification. They adjust the filtration parameter to the ball radius used by the abstract interpreter. This work is the closest to ours in these last two sections. They use simplices for robustness cover only. We propose to use them also for explicit inference domains definition, and for *functional* property verification.

B. Rationale of the Proof of Concept

Our group is qualified to discuss safety assurance rationales. In the MNIST PoC, we adopt a safety assessment standpoint. As SPCF-ML is our focus, we consider the assurance objectives and activities of a team whose methods and tools should be independent of that in action by system and AI-ML development teams. TDA on raw datasets ensures independence w.r.t. statistical estimation. No data transformation to qualify, no statistical estimation to provide confidence levels on statistical estimation. TDA-enabled (U)R-ML is a sample-dependent method 10. In this PoC, beyond independence w.r.t. statistical estimation, we also

¹⁰ With confidence region of the ID triangulated manifold model

have independence w.r.t to ML-models. We concentrate on complexity of the problem to solve.

Topological Data Analysis on MNIST is applied to functional hazard analysis. Verifying stability of approximant behavior w.r.t dataset variability and optimization variability are non-functional risks. These assurance objectives address the engineering risks inherent to ill-posed inverse problems AI-ML is part of. They are of fundamental importance for ML life-cycle (e.g. MLOps), but they do not address correctness or rareness on functional failure modes.

Our PoC explores TDA support for verification of verification¹¹: sampling coverage analysis and cross-validation coverage analysis.

C. Functional Hazard Analysis

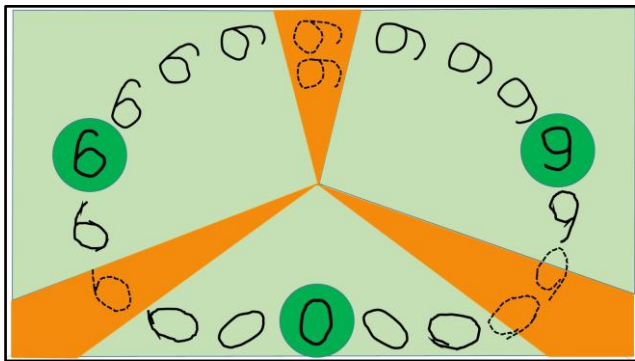


Fig. 7. Didactic evocation of {6,0,9} homotopy equivalence, and of {rotation, translation, homotety} symmetries. They create input-space hazardous regions (amber) subject to unreliable class separation by *any* ML-model. See fig. 10 for MNIST images belonging to hazardous regions (amber annulus).

LARD [25] is our planned next step in case of success on MNIST. We motivate our assurance activities by some fictitious aircraft landing narrative: we assume that some digits are painted on runways, and that their accurate recognition conditions safety-critical¹² operations.

1) Ambiguity

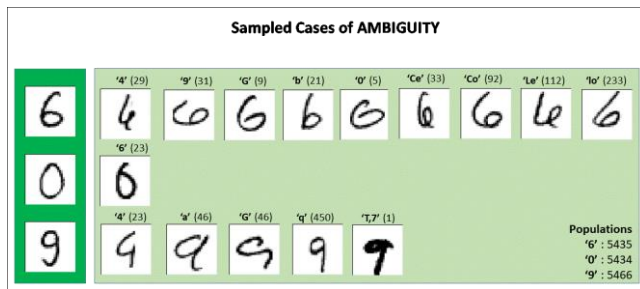


Fig 8. The ambiguity cases (and their cardinality) in MNIST\{6,0,9}

ODD of SPCF-ML MNIST classifiers should specify the image domain boundaries where even humans cannot decide. It should also identify where context-sensitive image interpretation occurs. For interpretation of distorted digits, presence of alphabetic characters matters. (see ‘a’, ‘le’, ‘co’, etc. in fig. 8).

We follow and extend the perturbation taxonomy of [6]. We assume ‘0’ has a distinctive operational role, and false

positives on ‘0’ are classified catastrophic by safety assessment, in the “no back-up” case. We assume false positives on ‘6’ and ‘9’ are hazardous. Detected false negatives on the three digits have no safety effect. Undetected false negatives are classified minor for safety, major for airport performance.

2) Perturbations

Light green background of the digits means that a single distortion (order 1) is regarded as common. Ultra-high reliability inference domains of {6, 0, 9}-classifiers should contain order 1 perturbed digits.

Reference Pattern	Handwriter's strokes			Pen & Paper		Sensing Process	
	Slant Backward	Slant Forward	Opened	Thickened	Swell	Noise	Clipping
6	6	6	6	6	6	6	6
0	0	0	0	0	0	0	0
9	9	9	9	9	9	9	9

Fig. 9 Seven examples of “pure” perturbations (unitary, order 1), grouped by causal sources that are partially independent. Most of them combine freely up to high orders (e.g. a clipped+noisy+thickened+slanted digit is regarded as perturbed at order 4).

D. Safety objectives

We must ensure impossibility or extreme remoteness of False Positives on ‘0’ (‘0’-FPs). Choice of the assurance policy proposed to Authority is a critical issue. In both cases, 0-FP failure mode is an *excluded event* since we are in the SPCF case. If rareness assurance policy is chosen, probability of this failure mode should be demonstrated at one or two orders of magnitude below 10⁻⁹/h. TDA in this PoC will be explored to support both policies.

E. Correctness policy

Current intermediary goal is a provably correct {6, 0, 9}-classifier on a restricted part of the inference domain (fig 11.left). Stratified persistence homology will be used to develop simplicial complex modeling of the inference domain over digits ‘0’, ‘6’ and ‘9’ distorted by *unitary* perturbations. Unitary perturbations are the counterpart of component failure modes in classical safety (e.g. fault tree analysis). Their independence is an issue under rareness assurance policy.

A progressive *data-integration* process, counter-part of progressive *code-integration* process in software assurance, is enforced. PH is applied after every data integration step, to interpret growth of topological complexity, to locally augment data and to tune a subset of filtration parameters as multi-scale inference domain design decisions. Intuition of the data augmentation process is conveyed in figure 10. Separability on the ambiguity regions will be designed by simplicial engineering. Order 1 involves 3x7 local boundary designs (cf. fig 9), and 16 separability designs¹³. The resulting simplicial complex’ actionable boundary will play the role of model-independent safety net.

¹¹ The assurance activities that verify that AI-ML verification activities are properly done.

¹² Admittedly, likelihood of runway hand-painted digits is extremely remote.

¹³ “Separation” is somehow a misnomer. Most of the light green sub-clusters of fig. 11 share intersections. See the tessellation of sub-clusters as evocative of designed separation, or designed entanglement (e.g. fig1 like).

If not geometrically and combinatorically too complex, Order 2 will also be addressed under correctness policy (i.e. geometric models of decision boundaries and proofs by simplicial set inclusions or null intersections). Order 3 and beyond will be addressed only under rareness policy.

PH is necessary, but not sufficient for the envisioned (U)R-ML engineering. Implicit augmentation of the complex to address local and global symmetries is one of the needed additional ingredients.

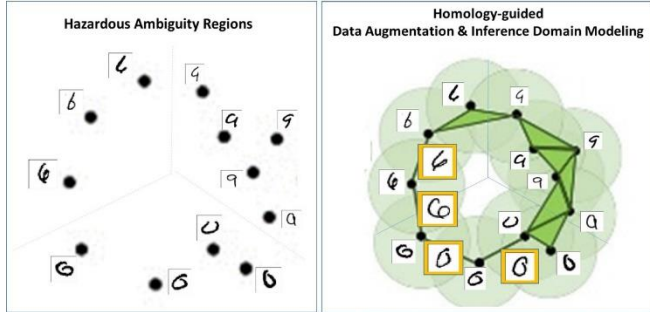


Fig. 10 Conceptual didactic figure derived from fig. 5. Left: a group of ambiguous distorted images. Right: for the selected radius of balls (filtration parameter – scale unit measure), PH seems to indicate the 12 images could be on a risky cycle like that of fig.7. Data augmentation (sampling or generation) is needed along the four newly created 1D simplices to confirm their relevance as new extensions of the inference domain..

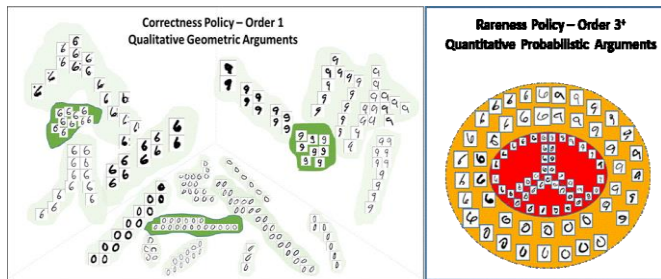


Fig. 11 Notional didactic figure suggesting the data-integration process. PH is applied after every sub-cluster data increment, to assess the increment of topological complexity.

F. Rareness policy

As order of perturbation increases, latent dimensionality and entanglement of shapes grow. Complexity of the ambiguity regions computed by PH become intractable for correctness policy. Conservative over-approximation of amber regions and probabilistic integration over triangulated manifolds will be the explored path. Its potential acceptance by certification bodies will be discussed in the group. Fault Tree Analysis should not be accepted as means of compliance to quantify failure modes in this context.

G. Current status and future work

MNIST restricted to $\{0, 6, 9\}$ was sub-labeled to isolate 54 unitary perturbations (see 21/54 in fig. 9) and 16 ambiguity cases (fig. 8), as part of independent safety assessment in (U)R-ML (Functional Hazard Analysis activity). PH was computed within 1 hour (~16500 images), and within 5 hours on whole MNIST (60000 images) on standard computing platform¹⁴. Interpretation of the persistence diagrams and design of the green and light-green ID¹⁵ boundary oracles are in progress.

XI. CONCLUSION

Starting from a reliability issue related to error correlation between AI-ML-model redundancies, we proposed a geometrical and topological explanation, not confirmed yet. We discussed the role of software development assurance and that of fault tolerant architectures to circumvent the problem. We argued that for ML components to be accepted as single points of catastrophic failure, like safety-critical software engineering and assurance managed to do, additional efforts and drastic progress on reliability are required.

We discussed the assurance regimes applicable to generalization errors in the most radical case. We promoted a flexible approach and gave its underlying rationale. We proposed TDA as a candidate means of compliance to supplement statistical estimation theoretical guarantees. We limited ourselves to a safety assessment and ML-model independent perspective. We illustrated the envisioned methodology on a fictitious airborne SPCF MNIST classifier.

ML state of the art is progressing impressively fast. However, fundamental problems remain unsolved. We made explicit our top3 showstoppers: actionable specification, diagnostic inability, 0-measure specification and verification cover. We are confident that mathematics, algorithms and tooling maturation will fill the gap, as it was the case for software. We gave a first glimpse on TDA as a promising asset to substantiate this optimism. It will take time, as it was the case for software. New engineering has to emerge and mature, leaving many opportunities for applicants and Authorities to resist race to market expedencies.

REFERENCES

- [1] Lucian Alecu, Hugues Bonnin, Thomas Fel, Laurent Gardes, Sébastien Gerchinovitz, Ludovic Ponsolle, Franck Mamalet, Éric Jenn, Vincent Mussot, Cyril Cappi, & al. “Can we reconcile safety objectives with machine learning performances?”. ERTS2022, Jun2022, Toulouse, France.
- [2] “EASA Concept Paper : First usable guidance for level 1 & 2 machine learning applications” March 2024. Issue 02.
- [3] Morayo Adedjouma, Christophe Alix, Loic Cantat, Eric Jenn, Juliette Mattioli, et al.. “Engineering Dependable AI Systems”. 17th Annual System of Systems Engineering Conference (SOSE), IEEE, Jun 2022, Rochester, United States.
- [4] Michael M. Bronstein, Joan Bruna, Taco Cohen, Petar Velickovic. “Geometric Deep Learning Grids, Groups, Graphs, Geodesics and Gauges. arXiv:2104.13478v2 [cs.LG] May 2021.
- [5] Dan Hendriecks, Nicholas Carlini, John Schulman, Jacob Steinhardt. “Unsolved Problems in ML Safety”. arXiv:2109.13916v5 [cs.LG] 16 Jun 2022.
- [6] Hermann Grassmann, “Extension Theory” 1862. History of mathematics Vol. 19. American Mathematical Society 2000.
- [7] Jian Liang, Frederick Park, and Hongkai Zhao. “Robust and Efficient Implicit Surface Reconstruction for Point Clouds Based on Convexified Image Segmentation”. University of California, Irvine March 21st, 2011.
- [8] Carlotta Giannelli, Lorenzo Sacco, Alessandra Sestini. “A local C2 Hermite interpolation scheme with PH quintic splines for 3D data streams”. arXiv:2108.12948v1 [math.NA] 30 Aug 2021.
- [9] Clémentine Laurens, “Un vieux problème de courbes enfin bouclé”. Pour la Science N° 545, Mars 2023.
- [10] Jean-Paul Blanquart, Philippe Baufreton, Jean-Louis Boulanger, Jean-Louis Camus, Cyrille Comar, Hervé Delseny, Jean Gassino, Emmanuel Ledinot, Philippe Quéré, Bertrand Ricque. “Software safety assessment and probabilities”. DSN 2016 Toulouse June 28th-July 1st.

¹⁴ No use of accelerators at this stage.

¹⁵ ID: In Distribution and Inference Domain.

- [11] Jean-Daniel Boissonnat, Frédéric Chazal, Mariette Yvinec “Geometric and Topological Inference” Cambridge Texts in Applied Mathematics 2018.
- [12] Aditi S. Krishnapriyan¹, Joseph Montoya, Maciej Haranczyk, Jens Hummelshøj, Dmitriy Morozov “Machine learning with persistent homology and chemical word embeddings improves prediction accuracy and interpretability in metal-organic frameworks” Nature Scientific Reports 11:8888 2021.
- [13] Frédéric Barbaresco, Frank Nielsen Editors, “Geometric Structures of Statistical Physics, Information Geometry, and Learning. SPIGL’20, Les Houches, France, July 27-31.
- [14] Marc Mézard “Désordre et frustration ... et au-delà” in Systèmes complexes, autour de Giorgio Parisi. Institut de France 11 octobre 2022 (unpublished communication).
- [15] Mark Lexter D. De Lara, “Persistent homology classification algorithm” PeerJ Computer Science January 10, 2023.
- [16] Simon Martin, Pierre Yves Lagrave, “On the benefits of SO(3)-Equivariant Neural Networks for Spherical Image Processing. 2022. Hal-03763121.
- [17] Herbert A. Simon “The Architecture of Complexity: Hierarchical Systems” in The Sciences of the Artificial, MIT Press 1969.
- [18] Martin J. Wainwright “High dimensional statistics – A Non-Asymptotic Viewpoint”. Cambridge Series in Statistical and Probabilistic Mathematics 2019.
- [19] Kanti V. Mardia, Peter E. Jupp “Directional Statistics” Wiley Series in Probability and Statistics 1999.
- [20] Karim Benmeziane, Patrick Fabiani, Stéphane Herbin, Jérôme Lacaille, Emmanuel Ledinot “Trusting Machine Learning Applications in Aeronautics” IEEE Aerospace Conference, Yellowstone, March 4-11 2023.
- [21] Stéphane Mallat, “Cours 3 : Malédiction de la grande dimension,” in L’apprentissage face à la malédiction de la grande dimension, Collège de France, 2018.
- [22] Mohammadreza Salehi, Hossein Mizaei, Dan Hendrycs, Yixuan Li, Mohammad Hossein Rohban, Mohammad Sabokrou “A Unified Survey on Anomaly, Novelty, Open-Set and Out-of-Distribution Detection: Solutions and Future Challenges” arXiv:2110.14051v1 26 oct. 2021.
- [23] Taejoon Byun, Sanjai Rayadurgam “Manifold for Machine Learning Assurance” arXiv:2002.03147v1 8 Feb. 2020.
- [24] Embedded France [Groupe de travail - NSL Normes pour la Sécurité de fonctionnement Logiciel et système - Embedded France \(embedded-france.org\)](#)
- [25] LARD - Landing Approach Runway Detection – Dataset for Vision Based Landing - Mélanie Ducoffe, Maxime Carrere, Léo Féliers, Adrien Gauffriau, Vincent Mussot, Claire Pagetti, Thierry Sammour. HAL Id: hal-04056760
- [26] Adélie Garin, Guillaume Tauzin “A Topological ‘Reading’ Lesson : Classification of MNIST using TDA”. aXiv 1910.08345v2 Oct 2019.
- [27] Amarnath R, Vinay Kumar V, “Pruning Distorted Images in MNIST Handwritten Digits”. arXiv:2307.14343 May 2023.
- [28] Chi Seng Pun, Kelin Xia, Si Xian Lee. “Persistent-Homology-based Machine Learning and its Applications: A Survey”. arXiv:1811.00252v1. Nov. 2018.
- [29] Mary L. Cummings, Ben Bauchwitz. “Unreliable Pedestrian Detection and Driver Alerting in Intelligent Vehicles” IEEE Transactions on Intelligent Vehicles · January 2024.
- [30] Alwyn Goodloe. “Assuring Safety-Critical Machine Learning-Enabled Systems: Challenges and Promise. Computer September 2023.
- [31] Tambon, F., Laberge, G., An, L. et al. « How to certify machine learning based safety-critical systems? A systematic literature review”. Autom Softw Eng 29, 38 (2022). <https://doi.org/10.1007/s10515-022-00337-x>
- [32] Bev Littlewood, “The use of Bernoulli and Poisson Processes for the evaluation of the reliability of critical software-based systems” Annex to IEC 61508 rev. 10, 2016.
- [33] Andrey Bychkov, Emmanuel Ledinot. “(U)R-ML experiments on MNIST”. Technical Report, Thales Research & Technology, 2024.
- [34] Faouzi Adjed, Mallek Mziou-Sallami, Frédéric Pelliccia, Mehdi Rezzoug, Lucas Schott, Christophe Bohn, Yesmina Jaafra. “Coupling algebraic topology theory, formal methods and safety requirements toward a new coverage metric for artificial intelligence models”. Neural Computing and Applications, 2022, 34 (19), pp.17129-17144.
- [35] Patrick Grother, Kayee Hanaoka, “NIST Special Database 19 Handprinted Forms and Characters 2nd Edition”. August 2016.
- [36] Daniel C. Castro, Jeremy Tan, Bernahrd Kainz, Ender Konukoglu, Ben Glocker “Morpho-MNIST: Quantitative Assessment and Diagnostics for Representation Learning” Journal of Machine Learning Research 20 (2019) 1-29.
- [37] JC Knight, NG Leveson, “An experimental evaluation of the assumption of independence in multiversion programming” IEEE Transactions on software engineering, 96-109.
- [38] Júlio Mendonça¹, Fumio Machida, Marcus Völp, “Enhancing the Reliability of Perception Systems using N-version Programming and Rejuvenation”, 53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W), 2023.
- [39] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, Rob Fergus, “Intriguing properties of neural networks”. arXiv:1312.6199v4 2014.
- [40] Jabde, M., Patil, C., Mali, S., Vibhute, A. “Comparative Study of Machine Learning and Deep Learning Classifiers on Handwritten Numeral Recognition”, In: Thampi, S.M., Mukhopadhyay, J., Paprzycki, M., Li, KC. (eds) International Symposium on Intelligent Informatics. ISI 2022. Smart Innovation, Systems and Technologies, vol 333. Springer, Singapore.
- [41] Curtis G. Northcutt, Lu Jiang, Isaac L. Chuang. “Confident Learning: Estimating Uncertainty in Dataset Labels” arXiv:1911.00068v6 [stat.ML] 22 Aug 2022.

A study of an ACAS-Xu exact implementation using ED-324/ARP6983

Christophe Gabreau*, Marie-Charlotte Teulières†, Eric Jenn‡, Augustin Lemesle¶, Dumitru Potop Butucaru||, Floris Thiant§, Lucas Fischer**, Mariem Turki‡

* Airbus, † Airbus Protect, ‡ IRT Saint-Exupéry, § IRT System X, ¶ CEA, || INRIA, ** Datakalab

Abstract: *This paper studies the exact implementation of the ACAS-Xu ML models (designed using Machine Learning technique) on several hardware platforms while ensuring some properties: ML model full semantics description, memory footprint optimisation, integer representation, formal verifiability. Certification aspects are also addressed using the EUROCAE/SAE joint group WG-114/G-34 current draft of the future standard ED-324/ARP6983 for embedding ML technology in aeronautical systems.*

Keywords: *Machine learning, Quantization, Formal verification, Implementation, ED-324/ARP6983, Certification.*

Disclaimer: This paper is based on preliminary results of the EUROCAE WG-114/SAE G-34 working group and only reflect authors' view.

I. INTRODUCTION

A. Context

In the airborne context, a safety-critical system cannot be certified as long as it is not demonstrated that this system safely performs its intended function under all foreseeable operating and environmental conditions. This demonstration only holds when the intent, along with the safety objectives, are satisfied in the target environment. When it comes to embed systems based on Machine Learning (ML) technology, the use of formal methods at design level seems very promising to support this demonstration and therefore, alleviate massive and costly testing activities on the selected HW platform. In addition, formal verification covers some of the learning assurance objectives from the novel ML standard ED-324/ARP6983 (ongoing work from the joint EUROCAE/SAE working groups WG-114/G-34).

This paper extends the previous works already performed on the ACAS-Xu case study in [3],[7] and [8]. While [3] and [7] were elaborating the design aspect of the development, [8] was focusing on the implementation part by proposing a method to implement a certifiable system with respect to ED-324/ARP6983 objectives. This paper proposes to study the capability to apply this method on several target platforms and to support the exact replication of formally proven ML

models. This work has been performed in the frame of the Confiance.ai project¹ and the DeepGreen project².

B. Contributions

The contributions are the methods to implement a certifiable ACAS-Xu system using surrogate ML models. These methods are supported by the following activities:

- Design the models, i.e. quantify, formally verify the quantified models and provide a specification of the quantified models' semantics
- Implement the quantified models, i.e. study the capability of an exact replication by respecting the specification of the designed models on several targets
- Contribute to certification, i.e. provide elements of a certification argumentation to demonstrate the conformity with the current recommendations of the WG-114/G-34 joint working group.

The abstract is structured as follows: section II describes the related work in the field of implementation of embedded ML; section III gives a brief description of the use case used to illustrate the approach and introduces the overall implementation strategy; Sections IV, V, and VI respectively address the design, implementation and certification aspects of the development.

II. RELATED WORK

There is an important survey [16] that structures and analyzes challenges, techniques, and methods for developing AI-based safety-critical systems. In particular, it addresses the development and the integration of a ML-based function as part of a safety critical system hosted on hardware platforms. References to development are mainly based on Autonomous Driving (AD) software frameworks using existing automotive standard guidance. To our knowledge, there is no such work in the airborne context, about the implementation of a ML model on hardware targets following the guidance of the future aeronautical standard ED-324/ARP6983. In an airborne context, there are some previous works dealing with the implementation of a DNN model on a recent CPU processor [1], studies about the prevention of the propagation of hardware errors that can lead to catastrophic failures in DNN

¹Web site: confiance.ai

²Web site: deepgreen.ai

accelerator systems [14], or some guaranties of robustness against hardware soft errors corrupting the target memory during implementation [9]. There are also work on other domains, for instance robotics, where implementation is also a challenge when it comes to embed DNN model for computer vision and system control purposes in different target hardware such as CPUs, GPUs or Intel’s neuromorphic chip [4].

III. CASE STUDY AND IMPLEMENTATION STRATEGY

A. The ACAS-XU case study

The ACAS-XU case study is described in details in [3]. Basically, the ACAS-XU system contains a specific function which computes the correct maneuver in order to prevent collision between UAVs for a set of input data (geometrical configuration of the ownship and the intruder UAVs). The implementation of the function is based on standardized lookup tables which provide the best advisory maneuver according to the geometry of the system given in Figure 1. The parameters are:

- ρ (ft), the distance from ownship to intruder
- θ (rad), the angle to intruder relative to ownship heading
- ψ (rad), the heading angle of intruder relative to ownship heading direction
- v_{own} (ft/s), the speed of ownship
- v_{int} (ft/s), the speed of intruder
- τ (s), the time until loss of vertical separation.

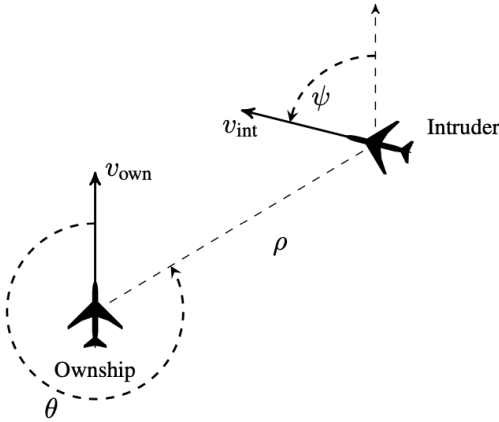


Fig. 1. ACAS Xu geometry [11]

The ACAS-XU function is implemented using ML. The ML part of the system, or ML Constituent (MLC) as per WG-114/G-34, is composed of ML model(s) and associated pre/post data processing that can be deployed on one or several items. The use of the term “item” complies with the definition given in the existing airborne system development guidance ED-79B/ARP4754B [19].

B. Implementation strategy

We decided to reuse the safe hybrid architecture that was used to develop the ACAS-Xu ML-based function[3], which is

composed of the *NN-based controller* (basically the MLC contains the models and the pre/post processing code), the *safety net* and a *check module* which, in real time, conditions the execution path to provide the correct advisory (cf. Figure 2).

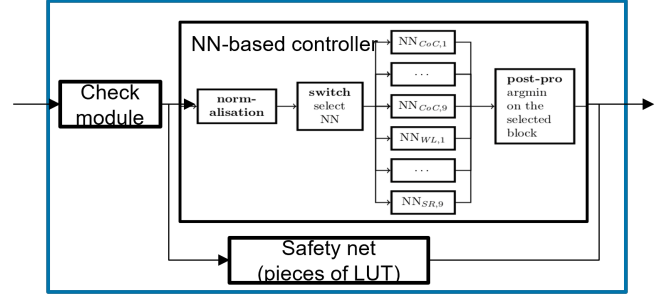


Fig. 2. Hybrid Architecture Overview [3]

We applied the method proposed in [8] to define an implementation strategy with the following main phases:

- **Specification:** We specified some requirements at MLC level (performance, functional, safety, operational), and we reused the similarity property defined in [3] (“*decisions of the model are identical to the ones obtained from the tables*”). In addition, we defined some specific requirements to cover the pre/post processing of the data. This phase is not developed further.
- **MLC Design/Quantization:** The full MLC logical architecture is decomposed into 45 models (and 45 Operational Design Domains-ODDs) as per the original study [3]. We simplified the problem by only considering avoidance maneuvers in the horizontal plan $\tau=1$. The 5 considered models (with their own ODD) are quantified in order to meet the implementation constraints, indeed integer representation is preferred for memory footprint and inference latency reduction purposes. Once quantified, the models are exported using the ONNX format. These models become the reference for both the formal verification and the implementation.
- **Model Design/ODD partitioning:** Each of the 5 ODDs is partitioned into two domains: one domain in which each quantified model is *proved* to hold the similarity property and one domain in which it does not. In the latter case, a safety net is used instead (cf Figure 2). These proofs are realized by using formal verification methods. At the end of the formal verification activity, we know for sure where the quantified models correctly perform.
- **Model Design/Description:** As per ED-324/ARP6983, the outcomes of the design phase is the ML Model Description (MLMD). This MLMD should express the full semantics of the quantified model to enable the capability of exact replication during implementation. This paper describes the semantics of the quantified models and stresses the gap between the current existing format and what is really needed in the avionic field.

- **Implementation:** This phase starts with the architecture of the ML Model, i.e. its decomposition into multiple ML Model Item Descriptions (MLMIDs), each MLMID corresponding to one item of the physical architecture. Then MLMIDs are transformed into implementation models for the different targets. In this paper, we consider the *exact* replication of the ML models, i.e. the inference ML model correctly and completely implements the specification of the ML model semantics expressed in the MLMID. Specific techniques (MLIR/iree tools suite) are used to control the transformations from MLMID to executable code in order to demonstrate the full preservation of the quantified model semantics and support the exact replication.
- **Exact replication study:** According to the obtained replication level (related to the used target), the performance of the implemented models are verified against the performance of the designed models.
- **Certification argumentation:** We built an argumentation which demonstrates the objectives defined in the EURO-CAE/SAE WG-114/G-34 ongoing standardization work (ED-324/ARP6983). This argumentation contributes to guarantee that the implementation preserves the model properties and do not introduce any unacceptable unintended behaviours. The argumentation is developed using the GSN assurance case notation.

IV. DESIGN

A. Quantization

Quantization plays a pivotal role in the optimization of machine learning models, with a specific focus on the Data-free method [15], [6]. This method aims to preserve the mathematical function of Deep Neural Networks (DNNs), making it particularly relevant in the context of the ACAS-Xu case study. Here, we consider a subset of five models at $\tau=1$. The notion of data free quantization involved no use of the data to compute a metric to optimize. In [6], Residual error has been introduced, and represents the error between the original weights, and the quantized weights. Let $W - Q^{-1}(W^q)$, with $R^1 = W^q$ be the residual error, and R^k be the K^{th} residual expansion term such that :

$$R^{(K)} = Q \left(W - \sum_{k=1}^{(K-1)} Q^{-1}(R^{(k)}) \right) \quad (1)$$

The maximal error between the original weight and the quantized weight decreases exponentially with the expansion order. The quantization process ensures a high preservation of the accuracy.

Table I summarises the results obtained by the quantization process, compared to the original models (with $\tau = 1$). An overall prediction accuracy of under 2% compared to the full precision model has been attained. The accuracy on next state prediction remained consistently higher than 95%, highlighting the effectiveness of the applied quantization techniques.

The quantization process achieves precise activation using diverse calibration methods. Surprisingly, despite numerous attempts, the introduced bias correction by [15] did not contribute significantly to this task.

Previous advisory	R2 Score		Argmin truth accuracy	
	Original	Quantized	Original	Quantized
model_CoC	99,82	99,1	96,05	95,35
model_WL	99,48	97,9	96,5	95,25
model_SL	99,52	98,02	96,03	95,19
model_WR	99,51	97,57	96,88	95,04
model_SR	99,57	97,98	96,15	95,15

TABLE I

COMPARISON OF METRICS BETWEEN ORIGINAL MODEL, AND QUANTIZED MODELS, WITH $\tau=1$ AND DEPENDING ON THE PREVIOUS ADVISORY.

B. ODD partitioning

The partitioning of the ODD is realized by verifying where the similarity property holds. This leads to the partitioning of the ODD into 2 classes of input space: where the NN-based controller performs correctly and where it does not (in this case the safety net is used as per Figure 2). The property is then verified using formal methods.

Following the same approach as [3], we decompose the input space of the neural network in p-dimensional boxes. As we selected a subset of the models with $\tau = 1$, we consider in this section only five models for the formal verification, each depending on the previous advisory and taking inputs in a five dimensional space with $(\rho, \omega, \psi, v_{own}, v_{int})$. The decomposition of the input space is thus a set of 7 356 800 5-dimensional boxes for each model, for a total of 36 784 000 boxes.

For each of these boxes, we aim to determine that the decision of the neural network are included in the decisions of the look up table (LUT) as formalised in [3]:

$$\forall l \subseteq \mathbb{R}^5, \text{decisions NN}(l) \subseteq \text{decisions LUT}(l)$$

For the formal verification, PyRAT [12] is used. PyRAT is based on abstract interpretation [2] and relies on abstract domains such as Zonotopes [10] to soundly overapproximate all possible outputs of a neural network. PyRAT computation and domains are sound w.r.t. floating point arithmetics, correctly rounding the different variables towards minus and plus infinity. Due to the number of verifications to perform and the constraints of the quantification, we chose to use only the Zonotope domain for the verification in a correct but incomplete way. Indeed, as the ReLU activation functions of the network have been replaced by quantisation operations, classical branch and bounds approaches by case disjunction on ReLU cannot be performed or would need to be adapted.

The formal verification is done by PyRAT on an implementation of the operators of the neural network reproducing exactly the ONNX reference implementation. On the ACAS-Xu case study, while the models are partially quantified (all the weights and bias of the models are converted to 8-bits unsigned integer), some rescaling operations with floating points are

still occurring in the QGemm operators that constitute the network. Additionally, a first conversion from float32 to uint8 is applied by a QuantizeLinear operator as the input of the network are float32 numbers. Following this, the inputs of a QGemm operator are first cast to 32-bits integer on which the the classical Gemm is applied (with weights and biases also in int32). The results of which is cast to float32 and rescaled before being cast back into 8-bits unsigned integer. Thus, while the integer operations preclude any floating point inaccuracy and rounding considerations, the rescaling operations (multiplication by scalar and addition with a bias in float32) may lead to floating point errors.

As we aim to have a verification correct w.r.t. the floating point implementation, *i.e.*, taking into account all possible floating point errors, the errors introduced by the rescaling in float32 (and more specifically in the addition) have been integrated in PyRAT in the Zonotope abstract domain. In addition to this, a specific abstract transformer has been defined for the cast function from float32 to uint8. Indeed, this function is not linear and must thus be overapproximated by a linear function to be used with an abstract domain like the Zonotopes. For an input x , we define the abstract transformer of the cast function as:

$$\text{cast}^\#(x) = x + 0.5 * \epsilon_{noise}$$

with $\epsilon_{noise} \in [-1, 1]$ the approximation introduced by the operation. This is illustrated in Figure 3.

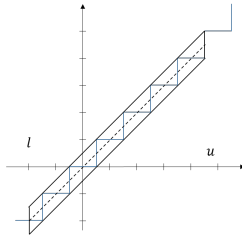


Fig. 3. Abstraction of the cast function from float32 to uint8 for the Zonotope abstract domain in PyRAT over an input interval $[l, u]$. In blue the function itself and in black its abstraction.

Table II summarises the results of the verification in function of the previous advisory. The total analysis time on the 36 millions of boxes with PyRAT was of approximately **40 hours**. Safe boxes means the similarity property holds, for unsafe boxes a counter example to the similarity property was found and unknown means we cannot conclude. As it is usually difficult to find counter examples with simple abstract interpretation based approaches, we rely on counter examples search using adversarial attacks inspired functions. We see that a large proportion of unknown results still remains as we used a fast but incomplete method. Using more precise domains or a complete approach may improve these results. However, considering the running time of complete methods and the number of points to verify (even if only 30% of the points are remaining) the computational cost of a complete verification remains prohibitive.

Previous advisory	Safe boxes	Unsafe boxes	Unknown
CoC	73%	0%	27%
WL	56%	0%	44%
SL	73%	3%	23%
WR	66%	4%	30%
SR	74%	4%	22%

TABLE II
PERCENTAGE OF SAFE, UNSAFE AND UNKNOWN BOXES IN THE ODD PARTITION AS VERIFIED BY PYRAT.

In order to further improve these results, the p-box partitioning might be delegated directly to PyRAT, instead of hard-coding it. This could allow a more dynamic p-box splitting, leveraging the fact that the similarity property might have been proven on already larger parts the input space than our current p-box. Thus allowing a finer partitioning only when needed with heuristics already existing in PyRAT [5]. On the other hand, for this verification the Zonotope domain was only adapted to the integer values with overapproximation. It might benefit from more in-depth changes, allowing to handle the cast operations with more precision.

C. Expression of the ML Model Description (MLMD)

As per ED-324/AR6983 guidance, the MLMD is the input of both the model verification and the MLC implementation process. It is key for the exact replication approach because it provides the capability to describe the complete semantics of the designed ML model, and enables to demonstrate that the implementation process does not alter this semantics and therefore, that the ML model properties which have been verified at design level, still hold.

In the context of ACAS-Xu case study, the MLMD is based on the existing ONNX format. This format is not sufficient to meet the "complete semantic description" requirement of the MLMD. Several elements are missing and are currently being addressed in a working group (connected to the ONNX SIG Operator). These elements are:

a) *Operators full specification*: The objective is to propose a complete specification of operators, unambiguous, no subject to interpretation or approximation. The specification should use a set of well-defined operators/primitives. The ACAS-XU model is a model composed of 9 fully connected layers represented by QGEMM as the model has been quantized. A complete specification of the QGEMM operator would be as follows:

- **Definition of the operator** : QGEMM is the quantized form of the GEMM (GEneral Matrix Multiply) operator
- **Description of inputs and attributes** : it includes roles description, data type, default value, constraints, description of the impact on the semantic of the operator. Example on Q_a input :
Description : Q_A is the first input tensor involved in the multiplication.
Data Type : tensor (integer)
Default : No default, input required
Constraints : If $\text{trans}Q_A \neq 0$, Q_A should be transposed.

Transposition should be performed before computation of the operation. If $transQ_A = 0$, the shape of Q_A should be (M,K). If $transQ_A \neq 0$, the shape of Q_A should be (K, M).

- **Specification**

Textual :

$$Y_q[i, k] = \frac{S_A \times S_B}{S_Y} \left(\sum_{j=1}^N Q_A[i, j] \times Q_B[j, k] - Z_B \times \sum_{j=1}^N Q_A[i, j] - Z_A \times \sum_{j=1}^N Q_B[j, k] + N \times Z_A \times Z_B + Q_C[i, k] \right) + Z_Y \quad (2)$$

Q_A , Q_B , and Q_C are the quantized input tensors of QGEMM, and Q_Y is quantized output. Z_A , Z_B , and Z_Y (resp. S_A , S_B , and S_Y), the zero points (resp. scaling factors) of the quantization³.

- **Reference implementation** Based on the C++ implementation proposed by [ONNX Runtime](#).

To establish a correspondence between the equation and the QGEMM documentation in ONNX Runtime, we can equate Q_A , Q_B , and Q_C to matrices A, B, and C, respectively. Correspondingly, Z_A , Z_B , and Z_Y can be associated with the parameters `a_zero_point`, `b_zero_point`, and `y_zero_point`, while S_a , S_b , and S_y align with the scales `a_scale`, `b_scale`, and `y_scale`, respectively.

b) *Control flow*: The control flow introduces the scheduling, the decomposition of the MLMD into several items with its interactions. Items could be hardware or software. In the case of the ACAS XU, three items are considered as shown in figure 4 on the implementation on TDA4VM Jacinto SoC: one hardware item, the Jacinto SoC, and two software items CPU and DSP (or GPU in the case of NVIDIA Xavier TX2). Several tools are being investigated within the MLMD community to support the expression of the control flow.

c) *Syntax*: The syntax should be textual and human readable in order to be open to manual code, and auditable. ONNX is based on protocol buffers, natively binary. The ONNX format is not human readable, this point is also addressed in the MLMD community

V. IMPLEMENTATION

A. Physical design

As per ED-324/ARP6983 guidance, this phase aims at defining the appropriate architecture of the MLC supporting the deployment of the ML Models on possibly one or several resources of the selected target. This implementation activity deals with the both components of the design outcome:

- MLMD part (ML model): The appropriate architecture is designed with respect to the resources available on the

³All attributes, inputs, outputs are supposed to be well-defined as described with Q_A

selected hardware platform. This activity will identify all the items which are necessary for the deployment of the model(s) onto the target. The MLMD is decomposed into ML Model Item Descriptions (MLMIDs). There is one MLMID per item.

- Non-MLMD part: the specific software parts needed for the pre/post data processing and the selection of the proper model execution (*Check Module* in Figure 2) according to operational inputs.

The ACAS-Xu model has been deployed on several targets:

- a TDA4VM Jacinto SoC fitted with 2 Cortex A72, 4 Cortex R5F, 1 C7X DSP and 2 C66X DSPs
- a NVIDIA Xavier TX2 fitted with 8 NVIDIA Carmel cores and a Volta GPU with 512 CUDA cores and 64 Tensor cores,
- a Xilinx Kria KV260 fitted with a [K26 SoM](#) embedding a ZYNQ UltraScale+ with 4 Cortex-A53 cores, 2 Cortex R5F cores, and a 256K logic cells FPGA fabric.

In all three cases, the ML model is implemented partially on CPU cores and an accelerator (respectively: a C7X DSP, a GPU, a DPU IP). The ML model (MLMD) is decomposed into 2 SW items (MLMIDs) hosted on one HW item (the SOC target):

- SW item 1: deployed on the CPUs (pre/post processing code wrt Figure 2)
- SW item 2: deployed on the accelerator (models)

For instance and with reference to Figure 4, in the Jacinto (HW item) implementation, each model is implemented using two kernels, one doing the quantization and the other doing the QGEMM operation. Kernels deployed on the C7X DSP accelerator using TI's [TIOVX](#) workflow that implements the [OpenVX](#) standard.

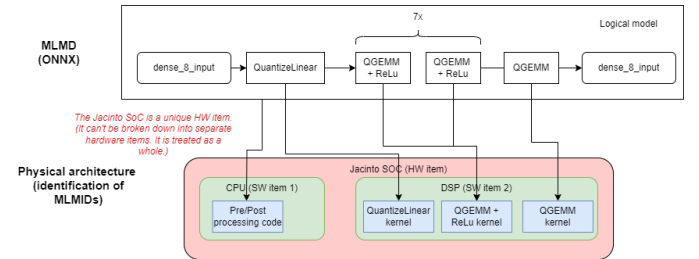


Fig. 4. Physical architecture on Jacinto target

B. Items development

In this phase, each item is implemented (coding, compiling, linking, loading and integrating) for the platform resource to which it is deployed. For instance the Jacinto implementation is described in Figure 5.

C. Exact replication study

a) *Verification strategy*.: Two verification methods can be applied:

- 1) Option 1: demonstrate that the series of transformations performed by the tool chains actually preserve the model semantics

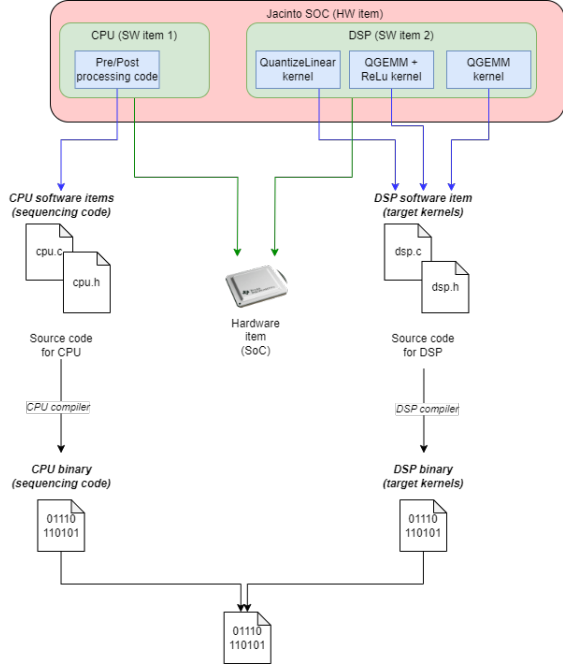


Fig. 5. SW items implementation on Jacinto target

- 2) Option 2: verify formally or by testing that the series of transformations preserves the model semantics.

When using complex COTS implementation tool chains, the first option is generally not an option, because the series of transformations is too complex to be analyzed and demonstrated to be correct (see paragraph about the VITIS AI and TIDL implementations later in this section), or simply because the details of the implementation are not available. This is to some extent similar to the case of the compilation of source code: except in some very specific cases (e.g. CompCert[13]), the complex series of transformation performed by a compiler cannot be demonstrated to preserve the semantics of the input code. In that case, the solution consists to verify the implementation by testing. For testing to be applicable, a correctness criterion must be defined and an oracle providing the expected value is needed. A correctness criterion can be (i) the identity of the final output of the network (the advisory to be applied), (ii) the identity of the last layer of the network (costs provided by the LUT), or (iii) the identity of all activations of the network. If testing is exhaustive, then all criterion are equal so the simplest (i.e., (i)) is sufficient: indeed, if the outputs of the implementation under test and the output of the reference implementation are identical for any input, there is no need to check what is going on in the intermediate layers. However, if testing cannot be exhaustive, which is generally the case, more confidence can be achieved by ensuring that intermediate results, e.g., activations, are actually identical.

This means that those activation values must be computed, which also means that the execution infrastructure (software and hardware) is also part of the oracle.

b) *Application to the ACAS-Xu use case:* As shown on Figure 6, the ACAS-xu model is a 9-layer fully connected network using essentially QGEMM operators, a quantified version of the classical GEMM (Generalized Matrix Multiplication) operator.

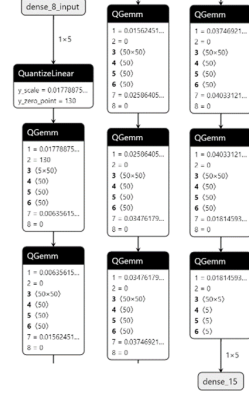


Fig. 6. ACAS-xu neural network (represented using Netron)

The ONNX runtime provides several optimized implementations of the QGEMM operator for different hardware targets. In our context, we implemented the QGEMM operator in the most straightforward way with respect to the mathematical expression of the operation shown on Eq. 2.

It is worth noting that the QGEMM documentation provided by ONNX gives a minimalist description of the semantics of the operator: “Quantized Gemm”. In particular, no part of the specification mentions anything about the accuracy of the operator. In our context, the ONNX-runtime implementation was considered as the reference implementation, i.e., the one defining the actual and exact semantics of the operator.

Our first implementations on the TDA4VM Jacinto SoC through **TIOVX** and the NVIDIA Xavier GPU using **cutlass** and **CUDA** were based on a direct implementation of Eq. 2.

Excerpts of the cutlass⁴ and Cuda implementations of Eq. 2 are given hereafter.

```
void qgemm(...) {
[...]
if (num_layer == 0) {
sz_in = SIZE_MODEL_IN;
quantize(r1, s1, z1, q1, size_layer_in); }
else if (num_layer == NB_LAYER-1) {
sz_out = SIZE_MODEL_OUT; }
// q1 = q1 x q2 + q3
result = cutlass_gemm(1, sz_out, sz_in, 1,
q1, q2, 1, q3, q1);
[...]
if (result != cudaSuccess) {...}
// q1 = q1 - z1 x ones * q2
result = cutlass_gemm(1, sz_out, sz_in, -z1,
ones, q2, 1, q1, q1);
[...]
// s2 = s1 x s2 / s3
scaleOp(s1, s2, s3, s2, sz_out);
// r1 = q1 x s2
elementwise_matmul(q1, s2, r1, sz_out);
if (num_layer != NB_LAYER-1) {
relu(r1, q1, sz_out); }
```



```

1 func.func @qgemm(
2   %a:tensor<?x?xui8>, %a_scale:tensor<f32>,
3   %a_zp:tensor<ui8>, %b:tensor<?x?xi8>,
4   %b_scale:tensor<f32>, %y_scale:tensor<f32>,
5   %y_zp:tensor<ui8>->(tensor<?x?xf32>) {
6   %b1 = mhlo.convert %b
7         : (tensor<?x?xi8> -> tensor<?x?xi32>
8   %1 = mhlo.convert %a
9         : (tensor<?x?xui8> -> tensor<?x?xi32>
10  %qlq2 = mhlo.dot %1, %b1 : (tensor<?x?xi32>,
11        tensor<?x?xi32>) -> tensor<?x?xi32>
12  %2 = shape.shape_of %a : ...
13  %3 = arith.constant 0 : index
14  %4 = arith.constant 1 : index
15  %5 = tensor.extract %2[%3] : tensor<2xindex>
16  %6 = tensor.extract %2[%4] : tensor<2xindex>
17  %7 = mhlo.convert %a_zp
18        : (tensor<ui8>) -> tensor<i32>
19  %o1 = tensor.splat %7[%5,%6]:tensor<?x?xi32>
20  %zp1 = mhlo.dot %o1, %b1 : ...
21  %8 = arith.subi %qlq2, %zp1 : ...
22  %acc = mhlo.convert %8
23        : tensor<?x?xi32> -> tensor<?x?xf32>
24  %9 = arith.mulf %a_scale,%b_scale : ...
25  %M = arith.divf %9,%y_scale : tensor<f32>
26  %10 = func.call @scalar_tensor(%M,%acc):...
27  func.return %10 : tensor<?x?xf32>
28 }

```

Fig. 7. High-level MLIR representation of the qgemm function

optimized GPU, CPU, TPU or even FPGA-based implementations. This process typically passes through general linear algebra representations (dialect `linalg`), affine loop nests (dialect `affine`), or vectorized code (dialect `vector`) where domain-specific optimizations are applied before reaching low-level dialects such as `llvmlir` or `amdgpu`. When necessary, the compilation process will partition the code between multiple devices (e.g. GPU kernels vs. host CPU control code).

MLIR allows the representation of MLMD specifications, of the MLMID-level information, and ultimately the implementation code. The format is textual, allowing inspection after each lowering or optimization phase, making it possible to trace the transformation of high-level operators such as `Conv2D` into, for instance, loop nests that have been fused with other loops, tiled, vectorized and parallelized on a multi-core.

We provide in Fig. 7 an MLIR encoding of the `qgemm` function.⁷ Functionally equivalent Python code is provided for reference in Fig. 8. To facilitate understanding, the MLIR code closely follows the Python reference, all Python variables having counterparts in the MLIR code. For conciseness, we assumed that scalar-tensor product is implemented under the form of external function `@scalar_tensor`, called from function `@qgemm`. The MLIR implementation is naturally more verbose, as the language is designed as a compiler IR. In particular, the type of each variable and operation is explicitly defined (in gray in Fig. 7).

Each MLIR operation (in blue in Fig. 7) belongs to a domain-specific dialect, represented as a prefix to the operation name. In addition to function `@qgemm` (and the scalar-tensor

⁷Under simplifying assumptions considered previously in the paper: initialization with zeroes, zero point for the second argument equal to 0, float output...

```

import numpy as np
def qgemm(a,a_scale,a_zp,b,b_scale,
         y_scale,y_zp):
    b1 = b.astype(np.int32)
    qlq2=a.astype(np.int32)@b1
    o1 = np.full(a.shape,a_zp,dtype=int32)
    zp1 = o1@b1
    acc = (qlq2-zp1).astype(np.float32)
    M = a_scale*b_scale/y_scale
    return M*acc

```

Fig. 8. Python functional reference for the MLIR code in Fig. 7

product functions it calls), the high-level MLIR specification of the ACAS-Xu neural network (of Fig. 6) also includes a `@relu` function, and the top-level model function that calls `@qgemm` and `@relu` multiple times.

Note that the semantics of all operations is fully defined, which requires the specification of all data types. In function `@qgemm` this requires the specification of all data conversions, e.g. that of 8-bit unsigned integers to 32-bit signed integers (in lines 8-9). Notice how `@qgemm` allows mixing high-level data processing operations of the `mhlo` dialect, simple arithmetic operations applied pointwise on tensors of the same size (of dialect `arith`), general tensor manipulations of the dialects `tensor` and `shape`, and classical function constructs of the `func` dialect (function definition, call, return).

Importing into MLIR is possible from a multitude of formats including Jax, Tensorflow, or Pytorch, or ONNX. However, `qgemm` involves complex type conversions produced by the automatic quantization process described in Sec. IV-A, meant to reduce execution time and parameter size without penalizing ML performance. Thus, the implementation of a single Dense layer involves `int8`, `uint8`, `int32`, and `float32` tensors. To make sure these conversions are preserved, we have manually produced the code of Fig. 7.

The output of the quantization process has the important property that floating point operations (which are not associative) are not used for the performance-critical matrix multiplications, thus allowing their optimization using the full power of parallelization algorithms (e.g. tiling), without affecting the output. Thus, exact replication can be attained without restricting optimization.

The dialect mechanism is designed for extensibility. For instance, in previous work [17] we have extended MLIR with a dialect providing the dataflow control primitives of `Simulink` and `Lustre/SCADE`, thus allowing the specification of stateful cyclic controllers (allowing the representation of stateful neural networks or reinforcement learning algorithms) and then their direct compilation into high-performance cyclic controllers.

But for the context of this paper, the extensibility of MLIR is best demonstrated by the experimentation platform we used, which is `iree`—an ML run-time and compiler producing code tailored for the run-time, to run on various targets : CPU (single- or multi-core), GPUs, and other accelerators. The `iree` compiler extends "vanilla" MLIR with new dialects allowing the description of a hardware abstraction, of execution man-

agement, of partitioning, allocation and scheduling. It also streamlines compilation for the run-time. On the resulting implementations, we apply fine-grain performance tracing using the [Tracy](#) profiling tool, to gain insight into performance bottlenecks.

Note that, unlike other ML compilation infrastructures, MLIR allows adopting a white-box approach to the design of the compilation pipeline, which can be completely customized. Along with the ability to trace source code transformation between compilation passes, this allows exposing critical transformations such as XLA-level linear algebra optimizations, tiling, buffer allocation, identification of computational kernels (to be executed on GPUs), parallelization, etc., which in turn allows (1) making performance trade-offs by activating/parameterizing compilation passes depending on the application and target architecture and (2) incrementally establishing the equivalence between the source code and the implementation.

VI. CERTIFICATION

Regarding the certification aspects, we used the concept of assurance case developed in [7] and followed the method and the notation developed in [8] to elaborate the argumentation. Indeed, we built an assurance case based on the development assurance principles of the last draft version of the future ML standard ED-324/ARP6983 available at the time we wrote the paper. Doing so, we anticipate that the airborne authorities (e.g. EASA) will recognize this standard as an acceptable means of compliance to the regulation requirements.

The standard ED-324/ARP6983 is an end-to-end standard that provides guidance to develop and certify an airborne system whose at least one function is ML-based. The figure 9 proposes the first level of the assurance case encompassing all the processes that have been standardized. The ED-324/ARP6983 contains all the usual processes included in a development assurance standard (cf CONFIDENCE module): planning, development, validation, verification, configuration management, process assurance and certification liaison. However the assurance case only focuses on the development, validation and verification processes. The whole assurance case has been developed but there is not enough space in the paper to display all the flow diagrams, therefore we will detail only some aspects textually. The underneath subsections textually develop the assurance case from the the goals defined in figure 9.

A. ED-324/ARP6983 - [GOAL] The MLC requirements are a satisfactory refinement of the allocated system requirements

[SOLUTION]: The ACAS-Xu function is fully specified by the standardized lookup tables developed in the EUROCAE WG 75.1/RTCA SC-147 MOPS for ACAS-Xu. Therefore, there is no need for any other MLC requirements except some operational requirements (timing, similarity property, target definition). It includes the partitioning of the input space into 5 ODDs to accommodate the 5 previous advisories and the definition of the similarity property.

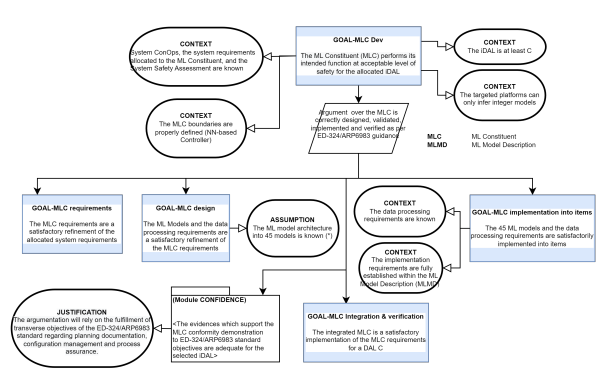


Fig. 9. Upper-level assurance case

B. ED-324/ARP6983 - [GOAL] The ML Models and the data processing requirements are a satisfactory refinement of the MLC requirements

[STRATEGY]: Argument over the datasets and the design models satisfy the ED-324/ARP6983 objectives. For any reference to initial development, refer to [3].

- 1) [SUBGOAL] *The ML datasets comply and are traceable to the MLC requirements* - [SOLUTION]: The 5 datasets are extracted from the standardized lookup tables developed in the EUROCAE WG 75.1/RTCA SC-147 MOPS for ACAS-Xu.
- 2) [SUBGOAL] *The 5 ML models comply with the MLC requirements, generalize out-of-sample, are stable and robust within their ODD* - [SOLUTION]: The models are quantized from the existing float models due to implementation constraints (targets platforms are integer-based). The compliance to MLC requirements is met by formally verifying that the similarity property holds. Wherever the quantized models (defined through their MLMD) do not hold the similarity property, the related part of the ODD is captured as a derived requirement and passed up to the system engineering level for safety net mitigation.
- 3) [SUBGOAL] *The ML data processing complies with the MLC requirements and ML Model architecture, and is traceable from the ML data processing description* - [SOLUTION]: The ML data processing code is developed from the MLC logical architecture (5 ML models in parallel) and the MLC requirements.
- 4) [SUBGOAL] *The 5 MLMDs are traceable to the 5 quantized ML models* - [SOLUTION]: Each MLMD is defined using the future ONNX extended format that expresses the full semantics of the model.

C. ED-324/ARP6983 - [GOAL] The 5 ML models and the data processing requirements are satisfactorily implemented into items

[STRATEGY]: Argument over the MLC is deployed onto traditional SW/HW item(s)

- 1) [SUBGOAL] *The MLC architecture into MLMDs is developed wrt the ED 324/ARP6983 objectives* - [SO-

LUTION]: Each of the 5 models is deployed onto 2 SW items (CPU+Accelerator). U(MLMIDs) = MLMD is easily demonstrated using the description of data/control flow between items.

- 2) [SUBGOAL] *The implementation of MLMIDs, the ML Constituent architecture requirements and the refined ML Constituent implementation requirements complies with applicable item level standard* - [SOLUTION]: The 5 models are manually coded from the specification of their semantics (extended ONNX format) using target platform libraries. The ED-12C/DO-178C guidance is used to demonstrate that the items development does not introduce any unacceptable error for DAL C software.

D. ED-324/ARP6983 - [GOAL] The integrated MLC is a satisfactory implementation of the MLC requirements for a DAL C

[STRATEGY]: Argument over the MLC integration and verification activities satisfy the ED-324/ARP6983 guidance

- 1) [SUBGOAL] *ML and Traditional Item Implementations are produced and loaded onto the target platform for ML Constituent integration and verification* - [SOLUTION]: The 5 ML models and pre/post processing code are integrated on the selected platforms (TI Jacinto, nVIDIA Xavier).
- 2) [SUBGOAL] *ML Training and Target Environment differences are identified and assessed for their impact on stability and generalization* - [SOLUTION]: The exact replication is supported by the verification that the predictions are binary identical (checked on the whole input space).
- 3) [SUBGOAL] *ML Constituent performance is verified* - [SOLUTION]: Verification credit can be sought for formal verification activity performed during design phase.
- 4) [SUBGOAL] *Integrated ML Constituent complies and is robust with MLC reqs* - [SOLUTION]: Operational requirements verification is in progress.

VII. CONCLUSIONS

In this paper, we managed to optimize pre-designed ML models in float format (from [3]) into models that we can embed in integer-based targets. The ONNX format of these models have been formally verified and exactly implemented onto 2 target platforms (TI-Jacinto and nVIDIA Xavier). At last we have demonstrated the compliance with the main development assurance objectives of ML future avionic ML standard ED-324/ARP693, making the ACAS-Xu system implementation certifiable.

It shall be noted that the exact replication is possible only when the ML model semantics and dynamics are completely and unambiguously specifiable. This makes the achievement of such a MLMD format an enabler of such technique.

If manual development is possible for this kind of non-complex model development, there is a need to automate the demonstration of the exact replication of the reference implementation in order to scale up the method to complex

models. The MLIR/iree tools suite sounds a very promising technique for this purpose.

As a generalization of the ACAS-Xu case study, the certification of such ML surrogate modelling technique seems doable using the ED-324/ARP6983 guidance. Indeed, the use of such technique to approximate a function that can be specified with physical equations (or any other Oracle like ACAS-Xu lookup tables), makes the demonstration of conformity to ED-324/ARP6983 objectives very attainable.

Going further in the context of the use of surrogate modelling technique, the effort to comply with ARP6983/ED-324 objectives is significantly reduced for a DAL C development. With the same level of effort, it may be contemplated to comply with upper DALs.

VIII. ACKNOWLEDGMENT

This work has been supported by the French government under the "France 2030" program, as part of the SystemX Technological Research Institute, and as part of the DeepGreen project with grant ANR-23-DEGR-0001

REFERENCES

- [1] S. CHICHIN, D. PORTES, M. BRUNDLER, and V. JEGU. Capability to embed deep neural networks: Study on cpu processor in avionics context. In *ERTS 2020*, 2020.
- [2] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 238–252, Los Angeles, California, 1977. ACM Press, New York, NY.
- [3] M. Damour, F. de Grancey, C. Gabreau, A. Gauffriau, and J.-B. Ginestet. Towards certification of a reduced footprint acas-xu system: a hybrid ml-based solution. In *Computer Safety, Reliability, and Security 40th International Conference SAFECOMP 2021*, pages pp.34–48, 2021, 978–3–030–83903–1. <https://hal.archives-ouvertes.fr/hal-03355299v2>, 2021.
- [4] T. DeWolf, P. Jaworski, and C. Eliasmith. Neurorobotics nengo and low-power ai hardware for robust, embedded neurorobotics. *Frontiers in Neurorobotics* (2020).
- [5] S. Durand, A. Lemesle, Z. Chihani, C. Urban, and F. Terrier. ReCIPH: Relational Coefficients for Input Partitioning Heuristic. 1st Workshop on Formal Verification of Machine Learning (WFVML 2022), July 2022.
- [6] M. C. Edouard Yvinec, Arnaud Dapogny and K. Bailly. Rex: Data-free residual quantization error expansion. arXiv preprint arXiv:2203.14645, 2022.
- [7] C. Gabreau, A. Gauffriau, F. de Grancey, J.-B. Ginestet, and C. Pagetti. Toward the certification of safety-related systems using ml techniques: the acas-xu experience. In *ERTS 2022*, page Session Th.4.C Assurance & Certification, 2022.
- [8] C. Gabreau, A. Gauffriau, M.-C. Teulières, D. Marandas, C. Pagetti, and C. Maxim). Implementation using ed-xxx/arp6983: the acas xu experience. In *CTIC 2023*, 2023.
- [9] F. Geissler, S. Qutub, and S. Roychowdhury. Towards a safety case for hardware fault tolerance in convolutional neural networks using activation range supervision.
- [10] E. Goubault and S. Putot. A zonotopic framework for functional abstractions, 2009.
- [11] 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.
- [12] A. Lemesle. PyRAT Analyzer website. <https://pyrat-analyzer.com/>. Accessed: March 18th, 2023.
- [13] X. Leroy, S. Blazy, D. Kästner, B. Schommer, M. Pister, and C. Ferdinand. Compert – a formally verified optimizing compiler. In *ERTS 2016: Embedded Real Time Software and Systems*. SEE, 2016.
- [14] G. Li, S. K. S. Hari, M. Sullivan, T. Tsai, K. Pattabiraman, J. Emer, and S. W. Keckler. Understanding error propagation in deep learning neural network (dnn) accelerators and applications.

- [15] e. a. Markus Nagel, Mart van Baalen. Data-free quantization through weight equalization and bias correction. In *ICCV, pp. 1325–1334, 2019*, page 1325–1334, 2019.
- [16] J. Perez-Cerrolaza and al. Artificial intelligence for safety-critical systems in industrial and transportation domains: A survey. *ACM Computing Surveys* <https://dl.acm.org/doi/10.1145/3626314>, 2023.
- [17] H. Pompougnac, U. Beaugnon, A. Cohen, and D. P. Butucaru. Weaving synchronous reactions into the fabric of ssa-form compilers. *ACM Trans. Archit. Code Optim.*, 19(2), mar 2022.
- [18] F. Rastello and F. Bouchez Tichadou (eds.). *SSA-based Compiler Design*. Springer, 2022.
- [19] SAE/EUROCAE. Aerospace Recommended Practices ARP4754B/ED-79B- development of civil aircraft and systems, 2023.

On the Feasibility of EASA Learning Assurance Objectives for Machine Learning Components

Florence de Grancey, Thalès Avionics
Sébastien Gerchinovitz, IRT Saint Exupéry and IMT
Lucian Alecu, Continental
Hugues Bonnin, Continental
Joseba Dalmau, IRT Saint Exupéry
Kevin Delmas, ONERA
Franck Mamalet, IRT Saint Exupéry

Florence.de-Grancey@fr.thalesgroup.com
Sebastien.Gerchinovitz@irt-saintexupery.com
Lucian.Alecu@continental-corporation.com
Hugues.Bonnin@continental-corporation.com
Joseba.Dalmau@irt-saintexupery.com
Kevin.Delmas@onera.fr
Franck.Mamalet@irt-saintexupery.com

Abstract— Despite the significant success of using Machine Learning (ML) in numerous industrial applications, how to integrate these technologies in safety-critical contexts poses many challenging questions. Several industrial and academic research groups, as well as various standardization committees are actively working to provide (partial) answers to these questions. In this document, we focus on one such initiative led by the EASA, which proposes a series of guidelines and requirements to develop ML-based systems for critical applications in the aviation domain. In this paper we investigate whether these requirements can be satisfied when using ML to solve a relatively simple regression task, that of building a neural network surrogate of the International Geomagnetic Reference Field (IGRF) model. Though we acknowledge all the structuring efforts towards the ambitious certification goal, our analysis pinpoints several important issues with some of these guidelines, such as ambiguous definitions, prohibitive computational costs, or currently very limited theoretical guarantees. Our analysis compels us to remain cautious about the various general recommendations proposed for designing trustworthy ML components for safety-critical systems. These conclusions call for the academic and industrial communities concerned by "Trustworthy AI" to strengthen their collaboration and pursue the research efforts necessary to address the existing challenges and establish sound methodologies for building safe ML-based applications.

Keywords— machine learning, safety, guidelines, certification, trustworthiness.

I. CONTEXT

In recent years, we have witnessed a multitude of ongoing initiatives to establish recommendations, guidelines and norms on how to develop and certify trustworthy Machine Learning (ML) solutions for safety-critical systems in the context of several application domains. One such initiative in the aviation domain is led by the European Union Aviation Safety Agency (EASA). In early 2023 the EASA released an open version of the "EASA concept paper: first usable guidance for level 1&2 machine learning applications", updated in March 2024 [1]. The document proposes a series of guidelines aimed at increasing the trustworthiness of ML components intended for aviation-related safety-critical applications. The authors formulate several objectives which, in their view, must be met to certify such technologies.

II. CONCEPT PAPER OVERVIEW

Ensuring that a data-driven software component is trustworthy raises numerous challenges. The EASA concept paper attempts to provide a holistic design methodology for ML-based systems in the aviation domain. In this section we briefly describe the

structure of the concept paper and point out the requirements we choose to analyze.

The EASA concept paper is structured around four main blocks: *AI Trustworthiness analysis*, *AI assurance*, *Human factors for AI* and *AI safety risk mitigation*. The safety assessment lies at the heart of the first block. It is within this phase that a system is assigned its main objectives in terms of safety, in particular, the assessment of the impact of a system failure on its environment (and notably on human lives), i.e. the dangerousness of the failures. The other blocks complete this assessment from different angles. *AI assurance* reinforces the level of trust in the AI system itself: on the one hand via "learning assurances" that "cover the paradigm shift from programming to learning", on the other hand via "development explainability", which seeks to open the "black box" that is machine learning. The remaining two blocks participate in safety "from the outside" of the system: the *Human factors for AI* cover the aspects of the relationship of the system with its user/operator, while the *AI safety risk mitigation* covers the residual risks identified by the *AI Trustworthiness analysis*.

We focus on AI/ML component safety only, because our field of research focuses on the ML models themselves. We seek to evaluate both the intrinsic risks of ML components, as well as the means of mitigation of these risks, which are also directly associated to the models. In this context, our analysis will focus on the objectives of the blocks *AI Trustworthiness analysis* (SA) and *AI assurance* (LM) only.

LM objectives can be divided into two categories: the objectives pertaining to the transparency and consistency of the engineering process, and the objectives related to the exploitation of quantitative and mathematical elements of the AI/ML models.

We do not address the objectives related to the engineering process, because they are classical and relatively indisputable. These objectives mainly request that each of the engineering activities must be clearly defined, traced and verified. In this set of objectives, the causal relationship between the measures taken and the safety risk is obvious, since it ensures that there is no discrepancy (or that it is as minimal as possible) between the discourse and the reality of engineering. Indeed, a lack of transparency and consistency in the engineering process undermines the whole safety demonstration.

The LM objectives related to the quantitative and mathematical elements of the ML model (the ones we focus on in this paper) are the following:

- LM-04: Quantifiable generalisation bounds

- LM-07: Bias-variance trade-off
- LM-08: Bias-variance requirement
- LM-09: Performance result
- LM-11: Stability analysis of the learning algorithm
- LM-12: Stability of the trained model
- LM-13: Model robustness
- LM-14: Verification of the anticipated generalisation bounds

Let us highlight that the EASA concept paper establishes a strong link between some objectives of the SA and of the LM. This link is implemented in the objectives relating to performance (LM 09), generalization (LM 04) and safety assessments (SA 01). Indeed, the anticipated Mean of Compliance of SA 01 objective indicates that "as part of the safety assessment process, AI/ML item failure modes are expected to be identified. Performance metrics should provide a conservative estimation of the probability of occurrence of the AI/ML item failures modes". The LM 09 and IMP 09 objectives are then referenced in the same paragraph as participating in this estimation, in connection with the LM04 generalization objective, which allows pronouncing on the failure rate in operation. Therefore, even if we do not analyze SA objectives directly, we discuss in Section V the link between LM objectives and safety.

The goal of the present paper is to evaluate the feasibility of the above LM objectives. However, doing so for a new operational use case is notably hard, mainly due to the cost of data acquisition. Therefore, we choose to focus on the magnetic declination estimation use case, a surrogate modelling problem (cf. the technical details in the next section). We have chosen this particular use-case for three main reasons:

- it is suitable for integration into an airborne system,
- a ML-based approach appear promising as compared to more traditional approaches,
- both data and algorithms are readily available.

For most common ML tasks, the ground truth values are either unknown, or observed via a noisy measurement process. The case of surrogate modelling is simpler, since it aims at approximating existing complex functions with ML models, and the ground truth values are therefore known. As such, the LM objectives are easier to evaluate for our surrogate modelling use case than for other ML tasks. We thus anticipate that the challenges identified in this work about the application of the LM objectives will also hold for other (more complex) use cases.

For the magnetic declination estimation use case, we can derive system/ML requirements using the requirements on magnetic heading provided in [2], which are performance oriented. Ensuring these requirements is considered as sufficient to demonstrate trustworthiness. Consequently, while the proposed use case may not strictly fall under the EASA guidelines for critical airborne systems, it still presents a realistic, well-defined, and thoroughly documented system. Moreover, the study performed on this use case is mostly generalizable to other surrogate software items used in critical embedded systems. In the upcoming sections, along with the analysis of our surrogate modelling use case, we also discuss the generalization of our findings to other types of ML tasks.

III. USE CASE AND APPROACH

In this section, we describe the magnetic declination estimation use case in detail, as well as our analysis approach, including the experiment setup.

Use case. We consider the following use case: build a neural network surrogate model of the International Geomagnetic Reference Field (IGRF) produced by IAGA. The IGRF describes the Earth's main magnetic field, by modelling the geomagnetic potential as a finite series of spherical harmonics. The latest generation, IGRF-13 [3], involves Schmidt semi-normalized associated Legendre functions of degree up to $n=13$, and provides the values of all spherical harmonics Gauss coefficients (which vary over time) at various 5-year-spaced epochs. IGRF-13 enables users to compute the magnetic field components in three dimensions, the magnetic inclination and the magnetic declination at each location on and above the Earth's surface, from 1900 to the present.

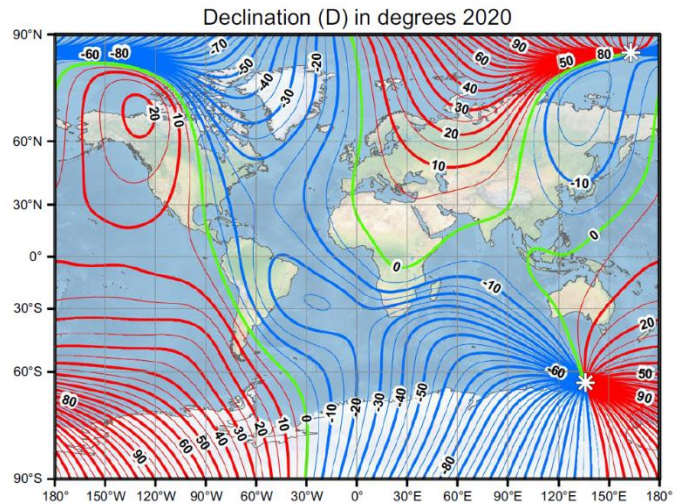


Figure 1: Declination map at the WGS84 ellipsoid surface for epoch 2020 (source: Alken et al. [1], Fig 1).

In the sequel we focus on the magnetic declination (also known as magnetic variation), which is the angle between the true North and the magnetic North. The magnetic declination depends on the latitude, longitude, altitude, and time; see Figure 1 for an illustration. The magnetic declination model is currently embedded in large commercial aircrafts to compute the aircraft's magnetic heading in real time. The magnetic heading data is crucial during the landing process, as airport diagrams still describe runways by their magnetic heading. However, for some aeronautics systems, embedding limitations prevent the use of the complete IGRF model and call for using a computationally more tractable model. To that end, and for illustrative purposes, we approximate the magnetic declination computation done by the IGRF-13 with a shallow neural network surrogate (see the experiment setup details below).

We stress that our goal is not to produce the most efficient or accurate surrogate model, but rather to propose a simple real-world use case on which the LM objectives can be instantiated. However, to keep things realistic, we consider the performance requirements on magnetic heading provided in [2].

Latitude range	Acceptable accuracy (95%)
50°S - 50°N	2°
50°N - 73°N	3°
60°S - 50°S	3°
73°N - 79°N	5°
79°N - 82°N	8°

Table 1: Acceptable accuracy values for magnetic heading [2]

Approach. We analyse the LM objectives of the EASA concept paper listed in Section II. For each of the objectives, we study the following (related) aspects:

- *Clarity*: is the objective clearly formulated or might it be prone to ambiguous interpretations?
- *Applicability*: does the objective apply to the considered use case?
- *Feasibility*: can the objective be achieved within reasonable costs and in a timely manner? Can it be formally and/or empirically assessed?

For each of the objectives, we start our analysis by instantiating generic definitions and principles to our use case. Then, we seek and apply well-established methods and results in the scientific literature to fulfill these requirements. Our experiment setup is described in the paragraph below. We also identify the hypotheses that must be met to ensure the validity of these approaches. Finally, we assess their practical feasibility and computational complexity.

Experiment setup. We use the python tool PyIGRF as the ground truth reference [4]. We restrict our study to the latitude range 60°S-82°N for which performance requirements are available (see Table 1). We consider all locations within that range, at an altitude of 100 meters and for the year 2005 for simplicity. This defines the Operational Design Domain (ODD).

We build three independent datasets that will prove useful in the next sections. The letters θ and ϕ denote the latitude and longitude.

1. Training set: it consists of 750K points $x_i = (\theta_i, \phi_i)$ drawn independently at random, uniformly within the latitude range 60°S-82°N and longitude range 180°W-180°E. This dataset is used for model training, that is, to build the neural network surrogate.
2. Calibration set: it consists of 10K points (θ_i, ϕ_i) drawn independently at random, uniformly within the latitude range 60°S-82°N and longitude range 180°W-180°E. This dataset is used in order to obtain estimates about the trained model, for the objectives pertaining to generalisation (see Section IV.F).
3. Test set: it consists of 250K points (θ_i, ϕ_i) drawn independently at random, uniformly within the latitude range 60°S-82°N and longitude range 180°W-180°E. This dataset is used only for test purposes.

Our surrogate model is a neural network having the following architecture: a fully connected ReLU neural network with 3 hidden layers, and 20 neurons per layer. We provide four scalar inputs to the neural network: $\cos(\theta)$, $\sin(\theta)$, $\cos(\phi)$, $\sin(\phi)$. The network has one scalar output modelling the magnetic declination. The outputs are normalized to [0,1]. We train the network to fit the magnetic declination (obtained with PyIGRF) on the training set, using the square loss with the SGD optimizer, a learning rate of 0.005, batches of size 32 and 15 epochs. We thus obtain a *trained model*.

Notation. We denote by f the true IGRF-13 model. We denote by S the training set and by \hat{f}_S the trained surrogate model built as explained in the paragraph above and trained used the training set S .

IV. RESULTS

In this section, we instantiate all the aforementioned LM objectives to this specific use case, and we evaluate them in terms of clarity, applicability, and feasibility.

A. Analysis of objective LM-09: Performance on test set

The first objective is about the performance of the trained model.

Objective LM-09: The applicant should perform an evaluation of the performance of the trained model based on the test data set and document the result of the model verification.

To achieve this objective we evaluate appropriate metrics over a “representative” test set. There is no particular issue concerning the *clarity*, *applicability* and *feasibility* of this verification step. For the IGRF surrogate model, representativity is simple since we can build the test data set as desired. Moreover, the performance metric is defined as the 95% quantile of all absolute errors on the test set, where an *absolute error* (also termed *accuracy* thereafter) is the absolute difference between the true and predicted magnetic declination values. Results are displayed in Table 2.

Latitude range	95% accuracy on test set
50°S - 50°N	1.51°
50°N - 73°N	1.97°
60°S - 50°S	3.14°
73°N - 79°N	3.47°
79°N - 82°N	5.00°

Table 2: Accuracy of the trained model when evaluated on the test set. We report the 95% empirical quantiles of the absolute errors (accuracies) on each latitude range.

Note that the trained model seems accurate enough in that the 95% accuracies on the test set almost meet the performance requirements of Table 1.

Even if the LM-09 objective is feasible for this surrogate use case, the choice of the adequate performance metrics may be a complex activity for the applicant. Notably more so if the applicant is dealing with computer vision or natural language processing models, where common metrics have a less clear-cut interpretation.

B. Analysis of objectives LM-07 and LM-08: Bias-Variance

The next objectives are about the Bias-Variance trade-off.

Objective LM-07-SL: The applicant should account for the bias-variance trade-off in the model family selection and should provide evidence of the reproducibility of the model training process.

Objective LM-08: The applicant should ensure that the estimated bias and variance of the selected model meet the associated learning process management requirements.

These two objectives seem, at first glance, both justifiable and achievable. Informally speaking, achieving a low bias and low variance corresponds to learning a sufficiently expressive model that does not depend too much on the training set. For many ML models, achieving low bias and low variance simultaneously should constitute a good indication of a well-performing predictive model. Despite these first intuitions, our analysis

shows that, even for the surrogate model case, the satisfaction of these objectives is not straightforward.

From a theoretical point of view, we can often consider, at least intuitively for regression tasks, the mean least square error decomposition into bias and variance. For a given example x , this decomposition expresses the expected squared error as the sum of a bias term (squared), a variance term, and a noise term:

$$\mathbb{E}_{S,y} [(\hat{f}_S(x) - y)^2] = \left(\mathbb{E}_S[\hat{f}_S(x)] - f(x) \right)^2 + \mathbb{E}_S [(\hat{f}_S(x) - \mathbb{E}_S[\hat{f}_S(x)])^2] + \mathbb{E}_y [(y - f(x))^2]$$

In the above equation, $\mathbb{E}_{S,y}$ means that we consider averages over all training sets S of a given size and all possible labels y for a fixed input x . The notation \mathbb{E}_S and \mathbb{E}_y are understood similarly. We can identify:

- the bias: $B(x) = \mathbb{E}_S[\hat{f}_S(x)] - f(x)$
- the variance: $V(x) = \mathbb{E}_S [(\hat{f}_S(x) - \mathbb{E}_S[\hat{f}_S(x)])^2]$
- the variance of the noise: $\sigma^2(x) = \mathbb{E}_y [(y - f(x))^2]$

In our surrogate use case, the variance of the noise equals zero. We perform a **rough estimation** of the bias and variance terms with a bootstrap method [5]. It consists in performing M experiments where a new data set $S(i)$ is drawn by sampling with replacement inside S . For the bias term, we estimate $\mathbb{E}_S[\hat{f}(x)]$ with $\frac{1}{M} \sum \hat{f}_{S(i)}(x)$ for each x , subtract the known value of $f(x)$, and average the squared result over all values of x in the test set. We proceed similarly for the variance. Results are shown in Figure 2, for a reduced training dataset of $n=25K$ points and $M=200$ bootstrap experiments. These estimates are repeated for several values of model complexity corresponding to the number of neurons per layer.

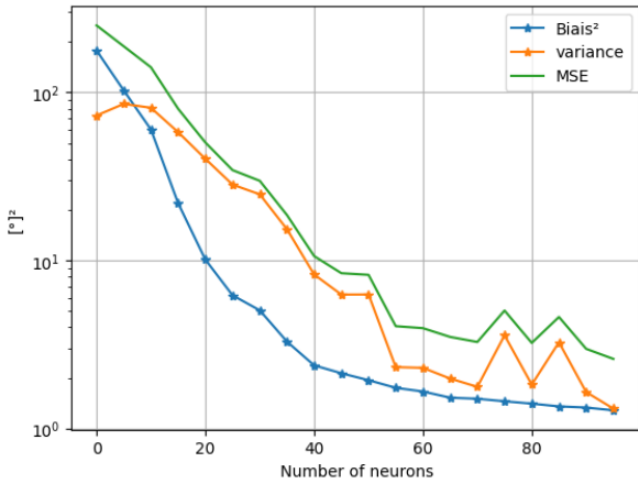


Figure 2: Rough estimation of MSE decomposition terms, for a variable number of neurons per layer.

Note that both the bias and variance terms are (roughly) decreasing for layer widths larger than 5. These rough observations are reminiscent of the double descent phenomenon in deep learning. This could lead the applicant to choose the largest network among those evaluated, while smaller networks (with about 40 neurons per layer in our case) might already be sufficiently accurate. This might raise embedding challenges. We thus argue that the bias-variance estimation may not always be the best tool to select an ML model architecture.

¹ Indeed, for an overparametrized neural network that can easily overfit the training data, the in-sample error can be zero, while the bias and the variance can be positive.

These considerations and experiments allow us to provide the following answers with respect to the criteria enumerated above:

- *clarity*: bias and variance are mathematical notions that are easily misinterpreted. Since the suggested informal definitions in the concept paper (see Anticipated MOC LM-08) are ambiguous and possibly different from the traditional notions¹, we used instead the formal definitions above, which are in line with those of the CoDANN report [6].
- *applicability*: these definitions should be specialized to the learning task at hand and the performance metrics used. Applying them to the absolute error metric (which would be more consistent with our use case) instead of the squared error metric is not straightforward. Applications to classification use cases would raise similar difficulties. A unified framework for bias-variance decomposition is proposed in [7, 8], but this decomposition is complex (the performances may not decompose as a sum of bias and variance terms) and not feasible in general.
- *feasibility*: While estimating the bias is possible in this surrogate model context, this is not the case for general ML problems, where the true value $f(x)$ is typically unknown. Furthermore, even in our setting, estimating the bias-variance tradeoff is computationally prohibitive as it requires training an important number of models (number of settings of complexity parameter, times number of bootstrap experiments).

Our analysis shows that attempting to satisfy this seemingly intuitive criterion for the trustworthiness of ML models can raise significant technical and methodological challenges. This calls for further academic research efforts. It would also be useful to investigate the quantitative link between an optimal bias-variance tradeoff and the resulting ML performances for several task-specific metrics.

C. Analysis of objective LM-11: Learning algorithm stability

The next selected objective is about stability of the learning algorithm.

Objective LM-11: The applicant should provide an analysis on the stability of the learning algorithms.

This objective aims at assessing the reproducibility of the learning process. As no anticipated means of compliance is provided in the concept paper, we choose to rely on the definition provided in [9]: Assume A is a symmetric learning algorithm², which given a training set $S = \{z_i = (x_i, y_i), i = 1, \dots, n\}$, outputs a function \hat{f}_S (a model) mapping x to y . For any i and any new sample $z' = (x', y')$, consider the modified training set $S_i = (S \setminus \{z_i\}) \cup \{z'\}$ obtained by replacing z_i with z' in S . The algorithm A is called β -stable if, for any training set S , any i , and any new sample z' , the losses of the models \hat{f}_S and \hat{f}_{S_i} on any sample $z = (x, y)$ differ by at most β . More formally, the algorithm A is called β -stable if

$$\forall S, \forall i, \forall z', \forall z, \quad |\text{loss}(\hat{f}_S, z) - \text{loss}(\hat{f}_{S_i}, z)| \leq \beta.$$

² To be rigorous, this symmetry assumption does not hold in our case (we use batch stochastic gradient descent). Though this assumption is useful for the theoretical guarantees proved in [9], the rest of the definition still makes sense without it.

This definition helps to define a process to assess learning stability:

- create new training datasets S_i by modifying one sample of the training dataset S and train a replacement model \hat{f}_{S_i} ;
- compute, for any sample z of the test dataset, the absolute value of the loss difference between the trained model \hat{f}_S and the replacement model \hat{f}_{S_i} ;
- find the maximal absolute difference, which should be lower than a given threshold β .

In our surrogate use case, we follow this process to empirically estimate a lower bound of β through Monte Carlo experiments on M modified datasets and their corresponding trained ML models. Due to the extensive computational cost, we only use a reduced initial training dataset $n=25K$, and $M=200$ modified training sets. For each experiment, we evaluate the maximum absolute difference over the test set. We also experiment with two different design choices to evaluate their influence on the estimated lower bounds: the first one uses the same weights initialization for all trainings, the second one uses independent random weights initialization for each experiment.³

Figure 3 presents the results obtained for each training iteration. We observe that the estimated β parameter is very high in the random weight initialization case. Even with a fixed weight initialization, the variation of the loss can be high which is difficult to interpret (optimization problem, parameters choice, complexity of the ground truth function to approximate,...).

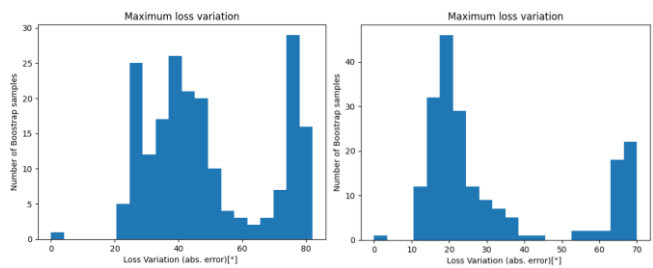


Figure 3 Maximum loss variation across the M training sets – (left) with random weights initialization – (right) with fixed weight initialization. Maximum value represents a lower bound on the β threshold of learning algorithm stability.

These considerations and experiments allow us to provide the following answers with respect to the criteria enumerated above:

- *clarity*: even if the formal definition given above seems understandable, it raises the challenge of the specification of the appropriate β parameter. The interpretation of this parameter represents the worst case over all training datasets S , all their modifications S_i , and all ODD points z .
- *applicability*: despite this favorable use case (surrogate modeling), we cannot find a sound choice of the threshold β . We must notice that usual ML learning processes rely on randomness (e.g., dataset shuffling, random ML model weights initialization) and on the choice of an optimizer. These elements all constitute a source of variability that can lead to the un-stability of performances in several points in the input space. Consequently, the design choices and the knowledge of the training framework highly affect the *applicability* of the learning process stability assessment.
- *feasibility*, considering that few or no formal methods are found in the literature even for our use case, we rely on a

Monte Carlo estimation of the parameter β . Such an evaluation is computationally expensive (for instance we had to work with a limited training set in our case), and would be even more challenging for large models and datasets. Moreover, performing this objective requires the evaluation of each source of variability in the learning process, which may be not be feasible with a black-box training framework. Furthermore, for general ML problems, a major difficulty may also come from the impossibility of generating new dataset samples.

To conclude, even for this surrogate use case, the interpretation of the evaluated β parameter is not clear, since the variation of the loss can depend on sources other than the dataset, such as the optimization process. Thus, we argue that the choice of the β parameter during model design is almost impossible, as it must bound all possible loss differences across the choice of the changed example.

D. Analysis of objective LM-12: Trained model stability

The next objectives are about the stability of the trained model \hat{f}_S .

Objective LM-12: The applicant should perform and document the verification of the stability of the trained model, covering the whole AI/ML constituent ODD.

The Anticipated MOC LM-12-1 gives only an informal definition as the evaluation of “perturbations in the operational phase due to fluctuations in the data input (e.g. noise on sensors) and having a possible effect on the trained model output”. We can rely on the formal definition given in [6]: given two thresholds δ and ϵ , stability is assessed by evaluating if :

$$\forall x, x' \in ODD, \quad \|x - x'\| \leq \delta \Rightarrow |\hat{f}_S(x) - \hat{f}_S(x')| \leq \epsilon.$$

The values δ and ϵ are supposed to be given in the ML component requirements, but the choice of δ and ϵ raises several challenges, as we point out both below and in Section V.B.

Clarity: At a first glance, this definition seems understandable and easy to achieve. However, even for a surrogate task it may not be adequate: if the ground truth function presents high local variations (large Lipschitz constant) in some parts of the ODD, a good surrogate ML model \hat{f}_S will also vary greatly. In particular, the surrogate model will only be able to fulfill the above condition for either very high values of ϵ , or for very small values of δ . Of course, such a choice of δ and ϵ is too conservative in regions of the ODD where the Lipschitz constant of the ground truth function is small, and does not at all guarantee that the surrogate model will be stable in such regions.

Applicability: For the IGRF use case, performance objectives are given in Table 1; we can therefore specify an acceptable ϵ threshold based on these performance requirements. For the δ threshold describing position errors we suggest to use the maximal lateral position error of 20 Nm (Nautic mile) given in [10]. In order to evaluate the stability condition above, we compute a two dimensional perturbation within this maximal

³ This again goes slightly outside of the scope of [9], which only considers deterministic algorithms. This experiment can however be useful to assess learning stability in a wide sense.

radius. Note that, for more complex use cases involving data in the form of text or image, the notion of “perturbation” is not so well-defined as in our use-case, and choosing the right notion of “perturbation” is already a challenge necessitating knowledge on the operational noise level and the Lipschitz constant of the targeted function (i.e., the local variation of f). A recent example of an expert definition of the maximal safe perturbation in subranges of the ODD can be found in [11].

Feasibility: On the IGRF use case, both the input space and the neural network have a small size. Therefore it is possible to employ complete formal methods (such as the SMT-based method described in [12]), to verify the stability property over all the input space⁴. For more complex problems, an estimation of ϵ can be empirically evaluated; either by sampling in the neighborhood⁵ of the test set samples, by using adversarial attack methods [13, 14] (aiming to maximize the error in the neighborhood), or by incomplete formal methods (such as abstract interpretation [15]). None of these methods provides guarantees for all x and x' , much less so when the ODD is high dimensional. For such more complex problems, the following probabilistic formulation of the property would be more convenient:

$$P_x(\forall x' \in B(X, \delta), |\hat{f}_S(X) - \hat{f}_S(x')| \leq \epsilon) \geq 1 - \alpha,$$

where X is a random point in the ODD (drawn from some distribution), and the ball $B(X, \delta)$ is the set of all $x' \in ODD$ such that $\|x' - X\| \leq \delta$. The above probabilistic property would mean that for most points x in the ODD (representing a fraction at least $1 - \alpha$ of the ODD), the surrogate model would not vary too much in a close neighborhood of x .

For the IGRF use case, we have experimented Monte-Carlo sampling estimation. We also perform an incomplete formal method (with Alpha-Crown [15]) on one thousand samples of the test set. This method is designed to compute upper u_p and lower l_p bounds containing model outputs (e.g. $l_p \leq \hat{f}_S(x') \leq u_p$), when the input is contained inside an l_p -ball around x : $B_p = \{x' \mid \|x' - x\|_p \leq \delta\}$ ⁶. We perform the computation with $p = 2$ for 10K samples and plot the absolute difference between these bounds and $\hat{f}_S(x)$.

Results are shown in Figure 4. Interestingly, on this use case, both methods (Monte-Carlo and Alpha-Crown) present coherent results and show that:

- the ML model is not stable close the north pole,
- the estimated model stability is consistent with the performance requirements.

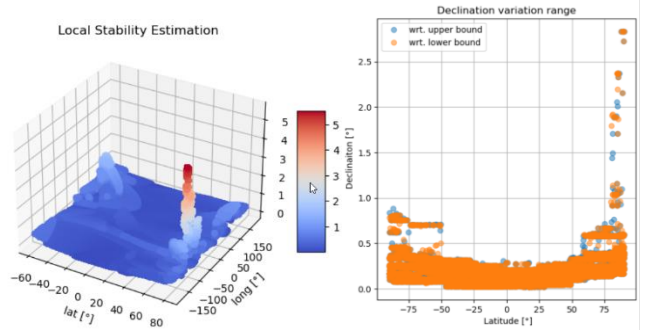


Figure 4 Trained model stability estimation: (left) Estimated local variations by Monte-Carlo sampling (right) Alpha-Crown upper and lower bound estimation for a 20Nm position perturbation [Note that only a subsample of test was processed with this method].

In conclusion, stability estimation is a pertinent tool to evaluate model vulnerabilities, when the variations of the ground truth function are known.

These results prove the feasibility of the objective in the low dimensional ML model that we study. However, the *clarity* and the *applicability* of the objective, such as the definition of the appropriate perturbation (e.g., δ) and model impact (e.g., loss function and ϵ), greatly depend on the use case and require both ML and operational expertise. Furthermore, since currently formal methods do not scale to large deep learning models, stability is mainly estimated by Monte Carlo or Adversarial methods, providing only a statistical lower bound estimation. Consequently, we cannot assume that this objective is *feasible* for such tasks.

E. Analysis of objective LM-13: Model robustness

The next objective is defined by:

Objective LM-13: The applicant should perform and document the verification of the robustness of the trained model in adverse conditions.

The concept paper suggests evaluating the model’s robustness against three types of examples:

- *Edge or corner cases* that can arise when considering data within the ODD but with one (resp. at least two) input variable(s) that is(are) close to the extremal values of the ODD;
- *Out of distribution (OoD)* examples that correspond to input data that are not covered by the training set distribution;
- *Adversarial* examples that may affect the AI/ML constituent expected behavior.

Edge, Corner and OoD. Concerning *clarity*, the definition of the edge, corner, and OoD examples is quite clear for our use case, but this definition is challenging for high-dimensional data.

As for *applicability and feasibility*: In the context of the surrogate use case, as we master data generation, it is quite easy to generate such test samples. For example, OoD samples will coincide with out of ODD samples (since the training dataset is drawn uniformly within the ODD). We can generate samples in the latitude range [82°N, 83°N] and evaluate the ML model

⁴ This was not done in this study due to lack of time and resources.

⁵ A particular attention should be paid to the condition “inside the ODD”.

⁶ For the sake of simplicity, the study was done on an ℓ^2 -ball of the neural network input space. Since our neural network is in fact only provided with vectors of the form $(\cos(\theta), \sin(\theta), \cos(\phi), \sin(\phi))$ as inputs, this over-approximation leads to conservative stability estimates.

performances. Table 3 presents some results for 1000 samples, revealing, as expected, a performance degradation outside the ODD.

Latitude range	95% accuracy on OoD set
82°N - 83°N	11.60°
61°S - 60°S	8.81°

Table 3 OoD performances evaluation

We must highlight that for real-world use-cases, collecting corner, edge, and OoD points may be a challenge in itself. Detection of OoD samples is also a challenge for safety in order to monitor the usage of the ML model, but this is part of other objectives. *Feasibility* may not be reachable for some real-world cases, since OoD, edge and corner cases are not easily defined for high-dimensional data.

Adversarial robustness. Adversarial robustness is generally defined for classification tasks, with few works in the literature addressing regression. For regression tasks, a definition is provided in [16], which refers to the "worst perturbation" \hat{u} defined, for a given sample (x, y) , as:

$$\hat{u} = \operatorname{argmax}_{u: \|u\| \leq \delta} |\hat{f}(x+u) - y|$$

Estimating the worst perturbation \hat{u} seems *applicable and feasible* for our surrogate use case. However it must be noted that the "worst perturbation" for a given sample (x, y) will depend on the operational noise level and the Lipschitz constant of the targeted function (i.e., the local variation of f). Besides, the proposed definition prevents us from using many of the existing formal methods, such as [15].

To enhance *feasibility*, we propose to use the following tractable definition: Given a tolerated variation ϵ , find the largest perturbation norm δ (also called robustness radius) on x :

$$\max\{\delta \geq 0: \forall \|u\| \leq \delta, |\hat{f}(x+u) - \hat{f}(x)| \leq \epsilon\}.$$

With this definition, the previous results on trained model stability obtained with [15] or by Monte-Carlo sampling (Section IV.D) are applicable. This local largest perturbation should be compared to the knowledge of the target function to provide interesting features on ML model robustness.

F. Analysis of objectives LM-04 and LM-14: Generalisation bounds

Two LM objectives focus on the generalisation bounds of the trained model \hat{f}_S :

Objective LM-04: The applicant should provide quantifiable generalisation bounds.

Objective LM-14: The applicant should verify the anticipated generalisation bounds using the test data set.

Informally speaking, the generalisation ability of a trained model is about how well it performs on unseen operational data. This is formalized in the statistics and ML theory literatures (see, e.g., in [17] or [18]) through the statistical notion of *risk*.

The *risk* of a trained model \hat{f}_S , denoted by $R(\hat{f}_S)$, is the (theoretical) average error over all possible operational points, weighted by an appropriate distribution. In our case, since the ground truth is given by the output $f(x)$ of the IGRF-13 model, and since we consider uniformly distributed latitude θ and

longitude ϕ within the ranges 60°S-82°N and 180°W-180°E, the risk reads:

$$R(\hat{f}_S) = \int_{-60}^{82} \int_{-180}^{180} |\hat{f}_S(x_{\theta,\phi}) - f(x_{\theta,\phi})| \frac{d\theta d\phi}{142360},$$

where $x_{\theta,\phi}$ denotes the Earth location at latitude θ and longitude ϕ (at an altitude of 100 meters). Importantly, the risk $R(\hat{f}_S)$ depends on the training set S ; it is a random variable.

A *generalisation bound* is a probabilistic bound on the risk $R(\hat{f}_S)$, typically expressed as a sum of an observed average error (called the empirical risk) and some statistical margin. Depending on whether the empirical risk is measured on the training set S or on some new calibration dataset S' , different mathematical tools are used. To the best of our knowledge, there are at least three families of methods to obtain generalisation guarantees.

A first family of bounds, which we could call *training-based generalisation bounds*, use the training set S to estimate the risk $R(\hat{f}_S)$ with the empirical risk given in our case by

$$R_S(\hat{f}_S) = \frac{1}{n_S} \sum_{x \in S} |\hat{f}_S(x) - f(x)|,$$

where n_S is the number of training examples. A rigorous statistical margin is then computed, i.e. a guaranteed upper bound on the *generalisation gap* $G = R(\hat{f}_S) - R_S(\hat{f}_S)$ that holds with high probability over the draw of the training set S . Various such bounds exist. They typically depend on the number n_S of training examples, on some (light) properties of the data distribution, and (to account for possible overfitting) on the model family complexity (e.g., the number of layers or parameters of the neural network, the type of activation function, etc). Unfortunately, such bounds are typically too large to be practical. A historical example in regression is given by the pseudo-dimension bounds (a generalization of VC-bounds to regression problems; see Theorem 11.8 in [18]). It is well known that these bounds are conservative (and thus typically pessimistic), as noted in the concept paper. Indeed these bounds control the generalisation gap $R(g) - R_S(g)$ of all models g under consideration (e.g., when varying all parameters of a given architecture) instead of the trained model \hat{f}_S only, and hold for virtually any data distribution.

Next we focus on *post-processing methods* that seem more promising in the near future. Such methods require a *calibration set* S' , which is a new dataset drawn independently from the training set S , and on which the trained model \hat{f}_S is either evaluated or modified (see below). Post-processing approaches typically yield better bounds than training-based methods, as they offer guarantees on the trained model only, instead of the whole model family.

Post-processing evaluation of \hat{f}_S . In this post-processing setting, the empirical risk is computed on the calibration set S' :

$$R_{S'}(\hat{f}_S) = \frac{1}{n_{S'}} \sum_{x \in S'} |\hat{f}_S(x) - f(x)|$$

where $n_{S'}$ is the size of S' . Then, the risk $R(\hat{f}_S)$ is upper bounded by $R_{S'}(\hat{f}_S)$ plus some guaranteed statistical margin. Various such generalisation bounds exist [17, 18] (using so-called concentration inequalities [19]). For example, when both outputs $\hat{f}_S(x)$ and $f(x)$ are bounded in $[0,1]$, Hoeffding's inequality yields $P_{S'}\left(R(\hat{f}_S) \leq R_{S'}(\hat{f}_S) + \sqrt{\frac{\ln(1/\delta)}{2n_{S'}}}\right) \geq 1 - \delta$, which means that the generalization bound

$$R(\hat{f}_S) \leq R_{S'}(\hat{f}_S) + \sqrt{\frac{\ln(1/\delta)}{2n_{S'}}$$

is valid for at least a fraction $1 - \delta$ of all possible calibration sets S' (while only one of them is observed in practice). Note that the bound is valid for any training set S .

Another example is given by Bernstein's inequality. The following version also holds when $\hat{f}_S(x), f(x) \in [0,1]$, for a fraction at least $1 - \delta$ of all possible calibration sets S' :

$$R(\hat{f}_S) \leq R_{S'}(\hat{f}_S) + \sqrt{\frac{2R_{S'}(\hat{f}_S)\ln\left(\frac{1}{\delta}\right)}{n_{S'}}} + \frac{2\ln\left(\frac{1}{\delta}\right)}{n_{S'}}$$

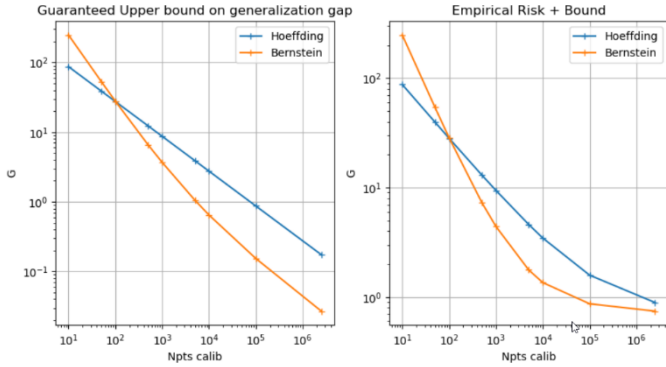


Figure 5: Hoeffding and Bernstein generalisation bounds in the IGRF use case.

In Figure 5 we plot the above two generalisation bounds in our IGRF use case (to that end, $\hat{f}_S(x)$ and $f(x)$ are first normalised to $[0,1]$, but the bounds are then converted back into degrees). On the left, we display the two guaranteed statistical margins

$\sqrt{\ln\left(\frac{1}{\delta}\right)/(2n_{S'})}$ and $\sqrt{2R_{S'}(\hat{f}_S)\ln\left(\frac{1}{\delta}\right)/n_{S'}} + 2\ln\left(\frac{1}{\delta}\right)/n_{S'}$ as functions of $n_{S'}$. On the right plot, we display the resulting generalisation bounds, given by the sum of the empirical risk $R_{S'}(\hat{f}_S)$ and these guaranteed statistical margins. They are statistically guaranteed upper bounds on the average magnetic declination error, for a latitude θ and longitude ϕ that are uniformly distributed within the ranges $60^\circ\text{S}-82^\circ\text{N}$ and $180^\circ\text{W}-180^\circ\text{E}$. A classical observation (from statistics theory) is that Bernstein's inequality entails a better bound, at least for sufficiently many calibration examples.

Post-processing modification of \hat{f}_S : risk-controlling prediction sets. An alternative family of post-processing methods consists in adding a predictive uncertainty quantification feature on top of the trained model. After modification, the predictor outputs a set of values (called a *prediction set*, typically an interval) instead of a single value, but with the guarantee of containing the ground truth with high probability. Conformal prediction methods [20] have gained renewed attention due to their simplicity and genericity. Next we focus on one algorithmic variant known as *risk controlling prediction sets (RCPS)* [21]. Just as before, this approach is only applicable once the model \hat{f}_S has been trained and requires an additional independent calibration dataset S' .

Several RCPS instances exist. For pedagogical purposes we describe a simple version below, which consists in replacing predictions $\hat{f}_S(x)$ with a prediction set $C_\lambda(x) = [\hat{f}_S(x) - \lambda; \hat{f}_S(x) + \lambda]$. To that end, the user first defines a risk level α , and then computes a margin value $\hat{\lambda}$ by solving some optimization problem specified in [21]; roughly speaking, $\hat{\lambda}$ is

chosen so that the empirical risk on the calibration set S' plus some statistical margin (given by, e.g., Hoeffding's or Bernstein's inequalities) falls below α .

This process comes with a probabilistic guarantee, which in our surrogate use case reads:

$$P_{S'}(P_X[f(X) \in C_{\hat{\lambda}}(X)] \geq 1 - \alpha) \geq 1 - \delta$$

This means that for a fraction $1 - \delta$ of all possible calibration sets S' , the prediction sets $C_{\hat{\lambda}}(x) = [\hat{f}_S(x) - \hat{\lambda}; \hat{f}_S(x) + \hat{\lambda}]$ contain the ground truth $f(x)$ for most inputs x (at least a fraction $1 - \alpha$ of all inputs).

We apply RCPS to the IGRF use case, with $\alpha = \delta = 0.05$ and a calibration dataset of 10K samples for each latitude range. Results are shown in Table 4. The computed margin $\hat{\lambda}$ (3rd column) is consistent with the performance requirements (2nd column). Note that the 4th column somehow corresponds to the objective LM-14, but that our conclusions (3rd column) are slightly more conservative, as statistical wisdom suggests.

Latitude range	Performance requirements (95%) (°)	Lambda (°)	Observed 95% quantile (°)
-50°, 50°	2	1.7	1.51
50°, 73°	3	2.1	1.966
-60°, -50°	3	3.5	3.140
73°, 79°	5	3.9	3.472
79°, 82°	8	5.9	5.004

Table 4: Application of the RCPS method to the IGRF use case

In conclusion, back to our three criteria:







- *clarity*: we found that the LM-04 and LM-14 objectives are clear enough, though several interpretations are possible (cf., e.g., our two post-processing approaches).
- *applicability*: these objectives are applicable. Note that we had to assume some distribution on the latitude θ and longitude ϕ . For another distribution, the results in Table 4 would likely be different.
- *feasibility*: the objectives can be reached with post-processing methods, within reasonable computational costs, and with theoretical guarantees. The latter however crucially rely on the fact that the examples in the calibration set are independent and drawn from the right distribution (the uniform distribution in our case).

We stress that instances where the above guarantees are breached may be concentrated within specific segments of the ODD, which could significantly impact the integration of such metrics in safety assessments.

G. Overview of LM objectives analysis

The detailed analyses of the previous sections led to several conclusions regarding the clarity, applicability, and feasibility of the LM objectives under study. Table 5 below provides a synthetic overview of our results, with a focus on feasibility. The conclusions drawn pertain to the magnetic declination estimation use case. Though not showed in the paper, we also analyzed the LM objectives on other toy use cases, for regression (a univariate nonparametric regression problem with Gaussian noise) and for classification (the classical two moons dataset). We obtained similar conclusions, though these (non-surrogate) ML tasks raise additional challenges.

Table 5. Overview of the analyzed objectives.

Objective	DAL	Feasibility?
Performance on test set (LM-09)	D, C	 Empirically: Yes
Bias-Variance analysis (LM-07, LM-08)	C	 Empirically: only rough estimation of bias and variance. Computationally prohibitive. Bias estimation is mostly specific to the surrogate setting (known $f(x)$). Formally: No
Learning algorithm stability (LM-11)	C	 Empirically: Only rough approximation Formally: No, very limited existing Theory
Trained model stability (LM-12)	D, C	 Empirically: Yes with average metrics Formally: Yes, formal methods for specific model architectures (not scalable to higher dimensional problems).
Model robustness (LM-13)	D, C	 Empirically: Yes but clearer definitions and metrics needed
Generalization bounds (LM-04, LM-14)	C	 Formally: No for training-based bounds: computable but not actionable. Yes for post-processing bounds. Warning: these bounds require statistical properties on the datasets.

V. DISCUSSION

In Section IV, we only address the technical challenges raised by the LM objectives, namely: the clarity of the objectives in terms of their mathematical definitions, the applicability to the IGRF use case, and the feasibility (computational cost, choice of some parameter values, theoretical guarantees, assumptions on the data, etc). However, we do not address the link between system safety and the LM objectives. This connection is established within the concept paper for LM-04 and LM-09 (as recalled in Section II), but in the future it would be useful to re-assess and refine this link. The contributions of the other LM objectives to system safety also need to be thoroughly investigated.

In Section V.A, we recall the paradigm shift from programming to learning, since it has key consequences on safety assurances. In Section V.B, we raise several questions concerning the contribution of safety assurance to system safety.

A. The paradigm shift

The concept paper proposes to address "the paradigm shift from programming to learning" with "learning assurances". We remind that the goal of the assurances is to obtain as many guarantees as possible that the contribution to safety of residual errors during operation will be acceptable. We highlight below how the two main engineering approaches described here (human programming-based and machine learning-based) address this goal radically differently. We conclude that they are significantly different information processing (i.e., transformation) approaches. In this perspective, we depict here the following fundamental differences of this "shift":

1. **Actor of the transformation.** To minimize the errors made by humans, engineers rely on well-established principles and practices, supported by strong evidence gathered throughout extensive experience. On the other hand, to minimize the errors done by the machine, the applicant can only rely on a deep understanding of the learning process.

⁷ Even in the case of surrogate models, for which detailed specifications of the function may be available, the compression task performed by the model cannot be fully specified. Note that

2. **Complexity of the problems to be solved.** We make the rather obvious assumption that ML techniques are to be used whenever no efficient alternative solution exists (i.e., one which can be completely specified and coded by humans)⁷. Consequently, no individual (or group of individuals) can analyze and verify exhaustively whether the computations of the ML-based software are correct or not. In most cases (in particular, when formal methods do not apply), engineers can only perform an empirical analysis of the ML model on some finite set of test examples, as if it were a black box. The human-written code based on complete software specifications can, on the other hand, be fully verified by other humans.
3. **Intrinsic nature of the transformation process.** In the case of human programming, software requirements are transformed into code via a succession of abstractions and decompositions, from the highest and widest level, to the lowest and thinnest one. This transformation allows several intermediate verifications, by either tests or analysis, and is end-to-end understandable and traceable. In the case of ML software, the transformation (i.e. the learning phase) is mostly done by an optimization algorithm, which computes the parameters of the ML model, by minimizing a loss function to automatically capture statistical patterns in the training data. These are two fundamentally different ways of processing the information.
4. **Coverage of the input data.** ML is mostly used to solve highly dimensional problems, which are impossible to describe / specify completely. Therefore, ensuring an exhaustive coverage of the input data space through massive testing is prohibitive for ML software (in absence of strong hypotheses regarding the data or the model). On the other hand, extensive coverage tests of human-written software are far more feasible essentially with the help of "equivalence classes" methods. The concept of "equivalence classes" frequently used in classical software test practices does not apply to ML software, due to the incomplete nature of the specifications of the problem being solved.

B. Safety-related challenges

We now briefly discuss important safety-related challenges that arise from the aforementioned paradigm shift. In safety, the main goal is to identify foreseeable failures and to obtain as many guarantees as possible that the impact and likelihood of failures will be acceptable in operation. It is thus important to question the link between the satisfaction of the LM objectives and this safety principle. Though these objectives appear to be very intuitive at first sight, we anticipate that seemingly small technical details in their instantiation might influence safety conclusions in a non-negligible way. Let us mention several examples, which appear at different levels.

1. **When interpreting an LM objective in terms of a mathematical definition.** For example, for the IGRF use case, in Section IV.F we provide two generalisation guarantees, but only one of them seems directly related to safety or, more precisely, to the performance requirements

is a reason to forbid compression options in the compilers in the safety critical software.

given in [2]⁸. Indeed a small risk $R(\hat{f}_S)$ only means that the average absolute error over the Earth's surface (for a specific distribution) is small, which does not directly translate into whether the 95% performance requirements of Table 1 are met⁹. On the other hand, the RCPS method yields results that can be directly compared to these requirements; see Table 4.

2. **When applying a mathematical definition that depends on parameters, metrics, assumptions, etc.** Since the link between the LM objectives satisfaction and system safety is not clarified, the applicant can have trouble in motivating the choice of some parameter values (such as the β , δ or ϵ parameters in Sections IV.C and IV.D), performance metrics (the loss function involved in the risk definition), acceptable performance values, and data assumptions.

For the IGRF use case the absolute error seems to be the most natural metric choice, but this is use-case specific. The choice of parameter values, and how they contribute to system safety, also seems very delicate. For example, as discussed in Section IV.D, a very stable trained model might feature a poor accuracy. Therefore, while a too small δ for a given value of ϵ could be detrimental to safety (since the ML model could be sensitive to adversarial attacks), a too large δ may lead to inaccurate predictions and could be detrimental to safety too.

Note from the previous paragraph that maximizing both robustness and accuracy is virtually impossible. This phenomenon contrasts with traditional assurance rules on software development. Indeed, traditional assurance rules can be cumulated to reduce the residual risk *i.e.*, the effects of a given rule will not cancel out the effects of another rule. The experiments conducted in this paper reveal that, when interpreted with our mathematical definitions with some parameter values, some LM objectives could be competitive. The classical cumulative property no longer holds for this specific phase of the ML development process. In other words, the Rearson metaphor of Swiss cheese slices does not apply anymore. In practice, whether all objectives can be satisfied simultaneously or not will depend on parameter values as well as other choices (e.g., performance metrics), which should thus be properly linked to system safety.

Overall, important efforts are needed to establish the links between the LM objectives satisfaction and system safety. This will enable to refine such objectives (in terms of mathematical definitions, parameter values, performance metrics, acceptable performance values, data assumptions, etc), or possibly to define new LM objectives.

VI. CONCLUSIONS

In this paper, we analyze several of the objectives proposed in [1]. We would first like to acknowledge all the structuring efforts towards the challenging goal of certifying safety-critical systems with AI components. However, our study shows that, even on a seemingly simple use-case, these objectives raise a

⁸ As noted in Section II, though the IGRF use case may not strictly fall under the EASA guidelines for critical airborne systems, it presents a realistic, well-defined, and thoroughly documented system. We use it as an illustrative example here.

⁹ If the risk $R(\hat{f}_S)$ were redefined for each latitude range of Table 1 (instead of a global average), Markov's inequality

series of technical and methodological challenges; see Section IV.G for a synthetic overview. While intuitive and arguably helpful to gain confidence in ML-based systems, some of these objectives turn out to be ambiguous or unfeasible from a practical standpoint in the analyzed context. Satisfying these objectives for non-surrogate ML tasks, or quantifying their eventual (degree of) satisfaction to the reduction of safety-related risks may posit additional hard challenges.

In light of these findings, we consider that:

- **Further academic research** must be conducted to develop methods that guarantee trustworthiness of an ML constituent. The scientific literature contains few appropriate methods that allow for the straightforward and efficient verification of the above objectives.
- Despite the relevance of the guidelines towards the certification goal, the scope and formulation of **several requirements should be refined and clarified**. This clarification is key to address complex use-cases.

VII. REFERENCES

- [1] EASA, "EASA concept paper: first usable guidance for level 1&2 machine learning applications," 2024.
- [2] EASA, "Easy Access Rules for Large Aeroplanes (CS-25)," revision January 2023.
- [3] P. Alken, E. Thébault, C. D. Beggan and al, "International Geomagnetic Reference Field: the thirteenth generation," *Earth Planets Space*, vol. 73, no. 49, 2021.
- [4] "pyIGRF: IGRF-13 Model by Python," [Online]. Available: <https://pypi.org/project/pyIGRF/>.
- [5] B. Efron, "Bootstrap Methods: Another Look at the Jackknife," *Annals of Statistics*, vol. 7, pp. p. 1-26, 1979.
- [6] EASA and D. AG, "Concepts of Design Assurance for Neural Networks (CoDANN)," 2020.
- [7] P. Domingos, "A unified bias-variance decomposition and its applications," in *Proceedings of the Seventeenth International Conference on Machine Learning (ICML)*, Stanford, CA, USA, 2000.
- [8] G. Valentini and T. G. Dietterich, "Bias-Variance Analysis of Support Vector Machines for the Development of SVM-Based Ensemble Methods," *Journal of Machine Learning Research*, vol. 5, pp. 725-775, 2004.
- [9] O. Bousquet and A. Elisseeff, "Algorithmic Stability and Generalization Performance," in *Advances in Neural Information Processing Systems*, 2000.
- [10] ICAO, Performance Based Navigation, 3rd edition, 2008.
- [11] M. Ducoffe, G. Povéda, A. Galametz, R. Boumazouza, M.-C. Martin, J. Baris, D. Daverschot and E. O'Higgins, Surrogate Neural Networks Local Stability for Aircraft Predictive Maintenance, 2024.

would imply high probability results similar in spirit to Table 4, but this implication would be crude. The RCPS method addresses probabilistic bounds directly, in a mathematically tighter way.

- [12] G. Katz, C. W. Barrett, D. L. Dill, K. Julian and M. J. Kochenderfer, "Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks," *CAV*, 2017.
- [13] J. Goodfellow, J. Shlens and C. Szegedy, "Explaining and harnessing adversarial examples," 2015.
- [14] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," *ieee symposium on security and privacy (sp)*, 2017.
- [15] K. Xu, H. Zhang, S. Wang, Y. Wang, S. Jana, X. Lin and C.-J. Hsieh, "Fast and Complete: Enabling Complete Neural Network Verification with Rapid and Massively Parallel Incomplete Verifiers," in *ICLR*, 2021.
- [16] K. Gupta, B. Pesquet-Popescu, F. Kaakai, J.-C. Pesquet and F. D. Malliaros, "An Adversarial Attacker for Neural Networks in Regression Problems," *IJCAI Workshop on Artificial Intelligence Safety (AI Safety)*, 2021.
- [17] S. Shalev-Shwartz and S. Ben-David, *Understanding Machine Learning: From Theory to Algorithms*, Cambridge University Press, 2014.
- [18] M. Mohri, A. Rostamizadeh and A. Talwalkar, *Foundations of Machine Learning*, 2nd ed., MIT Press, 2018.
- [19] S. Boucheron, G. Lugosi and P. Massart, *Concentration inequalities: a nonasymptotic theory of independence*, Oxford University Press, 2013.
- [20] V. Vovk, A. Gammernan and G. Shafer, *Algorithmic Learning in a Random World*, 2nd ed., Springer-Verlag, 2022.
- [21] S. Bates, A. Angelopoulos, L. Lei, J. Malik and M. Jordan, "Distribution-Free, Risk-Controlling Prediction Sets," *Journal of the ACM*, vol. 68, no. 6, 2021.
- [22] M. Ducoffe, S. Gerchinovitz and J. Sen Gupta, "A high-probability safety guarantee for shifted neural network surrogates," in *SafeAI 2020*, 2020.

Session Tu.3.B

Realtime System 2

Tuesday 11th June

15:00

–

Lauragais Room

Performance and confidence in feasibility analysis of real-time multi-core distributed systems

Etienne Hamelin*, Alexandre Berne*, Myrhal Boudiaf*, Paul Dubrulle†

**Université Paris-Saclay, CEA, List, F-91120, Palaiseau, France (email: firstname.lastname@cea.fr)*

†*Alkalee, France (paul.dubrulle@alkalee.fr)*

Abstract—With the trend towards software-defined vehicles, the scale and complexity of automotive software application is increasing rapidly, so that classical timing analysis methods become hardly practical. This paper proposes a new method, where a system model, formalized in an abstract multi-rate dataflow model of computation, is refined into a precedence-constrained scheduling problem. We characterize, and extend where needed, several schedulability analysis techniques to tackle this problem, and we demonstrate its use in the exploration of partitioning choices.

Index Terms—real-time, multicore, distributed, response time analysis, schedulability, dataflow

I. INTRODUCTION

The automotive industry is facing many challenges in its transformation towards the Software Defined Vehicle (SDV). The number of software components or services is vigorously expanding, as is the complexity of their interaction, and the associated integration issues, especially the real-time aspects. SDV applications represent a new step in complexity, involving thousands of mixed-criticality, both hard and soft real-time tasks, deployed on heterogeneous computing architectures made of microcontrollers and multicore processors, distributed across multiplexed networks, where configurations are moreover subject to over-the-air (OTA) updates during vehicle road life. Due to this whole new range of complexity, classical approaches of software timing analysis face significant challenge.

For these applications, many system-level properties, including timing feasibility (schedulability, reaction latency) cannot be left to be verified in the late integration testing phase. To avoid costly redesign cycles, we advocate a model-based approach to manage software timing properties, from service design to implementation and integration, enabling early in the process to predict application performance and timing feasibility.

In this paper, we present an approach, and a model-based analysis toolkit, based on the real-time multi-rate dataflow language PolyGraph, that allows to determine the feasibility of real-time constraints early in the design cycle.

First, in Section II we position our research with respect to previous work, then we present the context of modern SDV applications and a simple illustrative use-case in Section III. In

Section IV-A we present the timing annotations added on the PolyGraph language. Our main contribution is a model of execution that relates the abstract behavior of the dataflow with a real-time, precedence-constrained, task scheduling model. This scheduling model is then analyzed via both simulation and response-time analysis in Section IV-E and IV-G. Finally, in Section IV-H we compare the outcomes of this schedulability study with an actual execution on an embedded platform.

II. RELATED WORKS

In particular, the scientific community has developed a large knowledge base on models of computation, on the deployment of corresponding applications onto embedded computing platforms, and various approaches which enable formal reasoning about safety-related properties.

The embedded and cyberphysical systems community is progressively adopting technologies from the Internet world. For instance, the adoption of embedded Service-Oriented Architecture (SOA) frameworks like the Robot Operating System (ROS) allows engineers to easily develop, deploy and maintain complex processing chains made of many software services, by abstracting behavior from the actual details of the distributed execution platform, in particular the complex effects of dynamic scheduling. However, ROS builds a layer of cooperative scheduling of callbacks within each executor, on top of a typical priority-driven scheduling of processes. This complex scheme makes timing analysis extremely complex [1]. In comparison, some other SOA frameworks (e.g. ZMQ) provide message-oriented abstractions independent of any task or scheduling model.

Since the seminal works by Lee & Messerschmitt [3] on Synchronous Dataflow graphs (SDFG), many variants of the dataflow graph paradigm have been proposed. They explore various trade-offs between expressiveness of the model on one hand (related to the ability to accurately model a diversity of practical industrial systems), and the potential for formal analysis of useful properties based on a system model on the other hand [4]. Properties can be formally analyzed such as causal determinism, consistency, liveness, static schedulability, memory-boundedness, which are of particular interest in safety-relevant applications.

The SDFG paradigm does not explicitly model passing of physical time, therefore does not lend itself to timing analysis. Similarly, the synchronous language paradigm focuses

This research was partially supported by project DeepSEA, Grant agreement ID: 955606

on logical instants [5], and requires all reactions to happen between consecutive instants – which limits the potential integration of longer-running tasks. In contrast, in the Time-Triggered (TT) paradigm [6] and Logical-Execution-Time (LET) [7], time is the principal means chosen to ensure deterministic communication among software nodes. In these models, specifying a physical time instant for all task release (TT) or communication events (LET) leads to deterministic communications, but at the price of adding new tasks or new constraints to the online scheduler [8].

As a consequence, the models of computation listed above do not scale well to support modern, multi-rate, SOA-distributed SDV applications. In our study, we select the PolyGraph language introduced in [9], [17]. It inherits the determinism and causal actor semantics of dataflow networks, but adds a formal model of time, similar to TT however it constraints only a specific subset of tasks. This model lends itself particularly well to data-driven applications where multi-rate behavior is dictated by the diversity of sensors or actuators typically used in modern automotive, such as Advanced Driver Assistance Systems (ADAS). However the mapping of those applications onto schedulable task sets is yet to be defined.

Several implementations derived from dataflow graphs use static scheduling strategies [3] or static time-triggered implementations [15], as often used in aeronautic environments that require strong time- and space-partitioning, it seems comparatively few approaches have been proposed to deploy dataflow networks using fixed priority [16].

Many real-time software systems rely on a fixed-priority preemptive scheduling policy. While often dominated by other policies (e.g. deadline-driven), it has the advantage of being simpler to implement reliably, to understand and therefore to analyze. Since the formalization of the Response-Time Analysis (RTA) algorithm by [10], [11], many have extended this approach to support multicore parallelism and precedence constraints among tasks [12]. A Directed-Acyclic Graph (DAG) is typically used to represent precedence constraints among sub-tasks that run at the same rate. In particular, [13] extends the traditional RTA with a probabilistic model to estimate response time bounds of a DAG task model. This DAG-based approach however is not sufficient to model all the constraints that emerge from a multi-rate timed dataflow.

In the classical RTA algorithm [11] as well as the probabilistic multicore extension [13], the authors only consider the response time of tasks, i.e. the span from release to completion, whereas the PolyGraph-derived precedence scheme imposes to also model the variability of jobs release times. The Compositional Performance Analysis approach (CPA) [14] uses a set of event curves to model the inter-arrival times.

In this paper, we define an extension to the PolyGraph language to generate the scheduling constraints for large SoA-oriented systems with timing constraints, accounting for their high variability; from these constraints we define an execution model respecting the precedence and real-time constraints, and characterize different analysis methods to determine the schedulability of the resulting system. Our results show that

classical approaches have limitations, and we introduce an extension to the RTA analysis framework to overcome them.

III. CONTEXT

In this paper, we first explore the challenges to overcome during the development of real-time systems using Service-Oriented Architecture (SOA), powerful centralized computing, distributed sensors/actuators, all concepts required for the advent of a Software-Defined Vehicle (SDV). In this section we present the new challenges that emerge in the SDV domain, we recall the main aspects of the PolyGraph dataflow language, then we present some notations useful to the following analyses.

A. The Software Defined Vehicle challenge

The future generation of vehicles or other domains will implement more and more software components distributed over multiple multi-core Electronic Control Units (ECUs) interacting in real time. From the increasing amount of distributed data, the growing number of software components (SWCs) running on a single target and to keep the safety in the complete system, methods of design and validation need to adapt. By combining model-based engineering and timing analysis, we propose a feasibility assessment method based on the PolyGraph formal model to tackle those evolving challenges.

In a modern vehicle, multiple sensors such as cameras, lidars, radars, produce data at different rates; these signals are fed through various software components, some of which support lightweight state machines, and some support very computationally-heavy image-processing or trajectory prediction. Moreover, both computing and network resources are typically shared among both critical and quality-managed loads. PolyGraph allows to formally model all these constraints, defining rigorously the expected behavior (timing wise), and assess latencies, schedulability and deployment possibilities. Through those analysis, we can refine the previous model to have a better accuracy of the final system.

We illustrate our work on the use-case proposed in [18]. Fig. 1 presents the multirate dataflow structure, and Fig. 2 illustrates the vision processing stages. This system represents the perception stage of an ADAS, where images from a front-facing camera are fed by actor *ImgSrc* at 15Hz into two parallel processing chains. The lane-detection subchain detects road lane markings using classical computer-vision algorithms. First, a perspective transform creates a bird-eye-view, a color filter outlines the white and yellow markings. A boxed-search filter identifies lane keypoints, a 2nd order polynomial is fitted through the identified keypoints, so that the extrapolated lane can be drawn on the bird-eye view. This bird-eye view of the lane is perspective-transformed back into a front view for display. Its output is used for lateral control, typically in lane keeping assistance. The object-detection subchain detects and classifies obstacles like vehicles and pedestrians using a neural network ; its output typically drives an automated emergency braking (AEB) system or other longitudinal control

functions. Due to its heavier computation load, the object-detection subchain runs at a lower frequency, processing only every 5th frame. Both subchains are subject to a strict end-to-end latency constraint, imposed here through the *Display* sink actor. Although not representative of the scale of modern automotive SOA applications, this didactic use-case already showcases both some limits of existing approaches and the practical usability of our method in this context.

B. PolyGraph summary

PolyGraph is a formal dataflow modeling language, proposed in [17]. We recall here only the main notions. It defines the expected real-time behavior of *actors* (which model the software components), communicating through read-blocking FIFO *channels* (which model message-based communication). Actors *fire* atomically to consume and produce a fixed number of tokens on the incident channels.

The real-time constraints are modeled by defining the expected firing frequency of the actors modeling sensors and actuators in the multi-rate system. Maximal end-to-end latency on channel paths can be defined by adding phase offsets for the first expected firing. The other actors without explicit timing constraints represent *reactive* software components triggered by their inputs.

The formal model allows to infer inherited timing constraints for the reactive actors. Model-checking can then be applied to verify *consistency* and *liveness* properties (hence, memory-boundedness and absence of starvation, including missing inputs at expected real-time start date).

These abstract properties are verified with the minimum set of parameters derived from system engineering constraints. While this abstraction is an advantage in functional modeling to determine the feasibility and coherence of the timing requirements, it lacks precision when taking into account the possible variations in the software implementation. To account for variability in a concrete execution in software, we introduce in Section IV additional notions to capture execution time and jitter, without impacting the verdict on consistency and liveness.

C. Notations

Following [17], a polygraph is a tuple containing a set $\mathcal{V} = \{V_1, \dots, V_N\}$ of actors and a set $\mathcal{E} = \{E_1, \dots, E_M\}$ of channels. We denote by V_j^n the n^{th} firing of actor V_j in an execution of the polygraph.

A subset $\mathcal{T} \subseteq \mathcal{V}$ of actors are *strictly-timed*. To each actor $V_j \in \mathcal{T}$ is associated a frequency ω_j (equivalently a cycle time Π_j) and phase offset φ_j . For any actor $V_j \in \mathcal{T}$ and any of its firings V_j^n , its cycle time and phase offset determine an exact firing date $\tau_j^n = \Pi_j \times n + \varphi_j$.

From Propositions 1 and 2 in [19], we denote $\rho_i(n)$ the (non-decreasing) count of tokens produced on channel E_i by the first n firings of its producer actor i.e. V_j^1 to V_j^n , and $\sigma_i(p)$ the (non-decreasing) count of tokens consumed by firings V_k^1 to V_k^p .

Depending on the order of the firings in a polygraph's execution, it may be *non-blocking*, i.e. for any channel $E_i = \langle V_j, V_k \rangle$, the firings of V_j and V_k are ordered so that any firing V_k^p has sufficient input data tokens to prevent a blocking read. We denote $\mathcal{H} = \langle J, \prec \rangle$ the directed acyclic graph (DAG) encoding the partial order of the actor firings in the polygraph's non-blocking executions (with an edge if V_j^n must occur before V_k^p to preserve the non-blocking property, i.e. $V_j^n \prec V_k^p$). Fig. 3 represents the structure of the precedence graph for the first jobs of our ADAS use-case.

IV. CONTRIBUTIONS

A. Variations and execution times in PolyGraph

As explained in the previous section, the PolyGraph language reasons on firing instants for the verification of consistency and liveness. In an actual software execution, this restriction is not realistic and maintaining the abstraction comes with advantages and drawbacks (see Section II). To enable reasoning on firing time frames instead of firing instants, we add two modeling parameters: *completion jitter* and *timing budgets*.

Completion jitter allows to define the maximal acceptable deviation from explicit periodicity constraints. For example, an actor V_j with a strict cycle time of $\Pi_j = 100ms$ with a jitter of $10ms$ models a requirement to refresh its output data every $100ms$, plus or minus $10ms$. We denote z_j the completion jitter of a strictly timed actor V_j , and require that $z_j \leq \Pi_j$.

Timing budgets are associated to actor firings, to define the maximal acceptable execution time to process input data for the production of output data. It can for example be defined as an estimate of the maximal execution time, resulting from a static or dynamic worst-case execution time (WCET) analysis, or coarsely estimated through benchmarking statistics (depending on stringency of the actor's constraints). We denote b_j^n the budget of firing V_j^n .

The addition of timing budgets and jitter maintains the decidability for the consistency and liveness properties. For the consistency property, only the topology matrix of the polygraph and the frequencies are relevant, and they are not modified by the addition of budgets and completion jitters. The purpose of the liveness property is to determine the existence of at least one valid execution. Budgets and completion jitters can thus be ignored in the liveness property definition. Budgets indicate a maximal execution time delaying the occurrence of a firing, but the minimum remains 0. The completion jitters indicate a variation of the firing date of strictly timed actors, which can be ignored by choosing the fixed date τ_j^n for any firing V_j^n (equivalently, all jitters being set to 0). The Algorithm 1 in [17] thus remains applicable by choosing these values for the additional parameters. A positive verdict on liveness is a required starting point to further analyze the polygraph with other values for budget and jitter.

Given the precedence graph \mathcal{H} for a polygraph, the completion jitters and budgets allow to generate expected execution windows, or time frames, in so-called *optimistic* and *pessimistic* scenarios. In the optimistic scenario, the execution

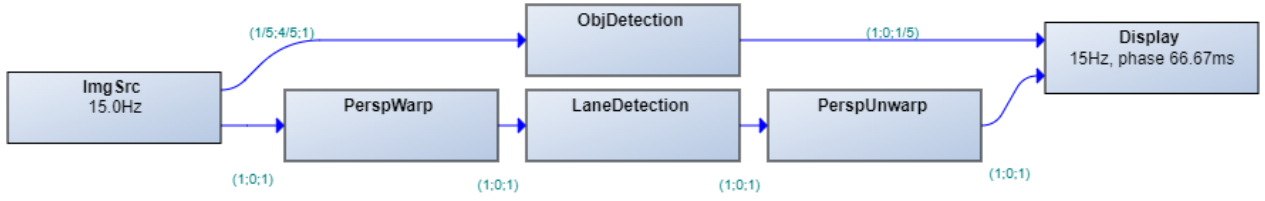


Fig. 1. ADAS use-case dataflow



Fig. 2. ADAS vision processing stages: front view (ImgSrc output), bird-eye view (PerspWarp output), lane detection (2 stages), inverse perspective (PerspUnwarp output), and object detection.

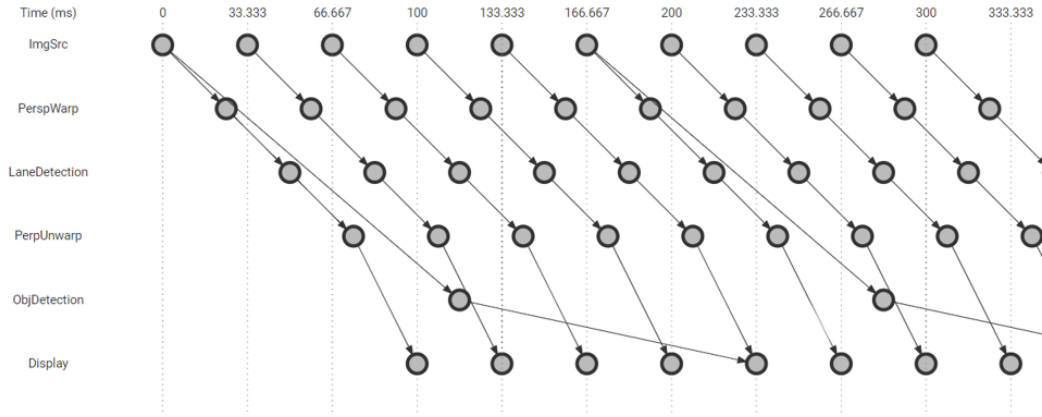


Fig. 3. Jobs precedence graph

times are ignored to define the bounds, and in that case the time frames are maximal. In the pessimistic scenario, the size of the time frames are minimal, considering execution times such that all actors consume the entirety of their timing budgets.

For each firing V_j^n , we define the following time frames:

- **allowed execution:** defines in the optimistic scenario when the actor has all the required data on its input queues at the earliest, and when the data is required at the latest on all its output queues. That frame is thus the maximum time frame during which input data is available for computation and output data is relevant to successors in the precedence graph. We denote al_j^n the lower bound of this frame and au_j^n its upper bound.
- **pessimistic execution:** defines the same instants in the pessimistic scenario, when all firings consume their whole timing budget. That frame is thus the minimum equivalent of the allowed time frame. We denote pl_j^n the lower bound of this frame and pu_j^n its upper bound
- **realization:** defines the instants when the firing (atomic

consumption and production) may occur. We denote rl_j^n the lower bound of this frame and ru_j^n its upper bound.

We detail in the following how to determine these time frames. Before going into more details, Fig. 4 illustrates their definition for a polygraph with 3 actors and 2 channels modeling a simple functional chain Sensor \rightarrow Compute \rightarrow Actuator. Sensor and Actuator are strictly timed, with frequencies respectively $10Hz$ and $5Hz$. Actuator has an offset of $50ms$. Compute is a reactive actor with inherited frequency of $5Hz$. The jitters of Sensor and Actuator are set respectively to $10ms$ and $20ms$. If we assume that Sensor produces every two firings the number of tokens consumed by Compute, the polygraph is consistent and live. The precedence graph is then quite simple, Sensor(1) \rightarrow Sensor(2) \rightarrow Compute(1) \rightarrow Actuator(1). The timing budgets are $10ms$ for the firings of Sensor, $30ms$ for the firing of Compute, and $20ms$ for the firing of Actuator.

In Fig. 4, note first the position of the realization time frame of firing *Sensor(2)* and the allowed time frame of *Compute(1)*. The firing *Sensor(2)* is constrained to be realized between 190 and $200ms$ by the timing parameters of the Sensor actor. In

any valid execution, the data is thus produced within this frame. Since it is required by *Compute(1)* as an input, its allowed time frame cannot start before $190ms$. Then note that in the pessimistic scenario, *Compute(1)* ends at the soonest at $220ms$, which constrains the pessimistic lower bound for *Actuator(1)*.

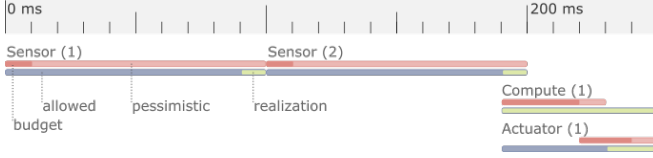


Fig. 4. Firing time frames. The time budget is figured in dark red, the pessimistic frame in pale red, the allowed frame in blue, realization frame in green.

To determine these time frames, the model parameters are used as follows. The allowed time frame for V_j^n is initially defined by interval $[\tau_j^n, \tau_j^n + \Pi_j]$ if V_j is strictly timed. The realization time frame is defined by $[\tau_j^n + \Pi_j - z_j, \tau_j^n + \Pi_j]$. For reactive actors, these frames are initially equal, and defined by $[0, +\infty[$. All pessimistic frames are initially equal to the allowed time frame.

In addition to these initial definitions, every precedence constraint in \mathcal{H} of the form $V_j^n \prec V_k^p$ requires that the data produced by V_j^n be available on time for execution of V_k^p in all scenarios, that is:

$$\begin{aligned} \forall (V_j^n, V_k^p) \in \mathcal{J}, V_j^n \prec V_k^p &\Rightarrow rl_j^n \leq al_k^p \\ &\Rightarrow \max(rl_j^n, pl_j^n + b_j^n) \leq pl_k^p \\ &\Rightarrow ru_j^n = au_j^n \leq au_k^p \\ &\Rightarrow pu_j^n \leq pu_k^p - b_k^p \end{aligned} \quad (1)$$

Then, a propagation of these constraints through the graph \mathcal{H} starting with the initial values (for example using a variant of the Bellman-Ford algorithm) refines the initial time frame definitions to ensure causality of communication and real-time constraints are enforced. If a reactive actor's firing has no successor in \mathcal{H} , the upper bounds for its time frames can be chosen (as long as it is less than or equal to the hyperperiod of the polygraph, to preserve the consistency property).

For a consistent and live polygraph, the initial allowed and realization time frames can always be refined while respecting the inequalities of Equation (1), since \mathcal{H} is an encoding of the partial order of non-blocking executions and there is at least one valid execution for the polygraph from the liveness property. As such, with the completion jitter extension, any sequence of firings ordered by firing date, such that the firings occur within their realization frame, is a valid execution for the polygraph.

Note that if refining the time frames results in pessimistic frames smaller than the budget or frames where the pessimistic upper bound is out of the realization frame, the system is infeasible regardless of scheduling and resources. Indeed, a frame smaller than the budget implies that the actor have

insufficient time to complete in the pessimistic scenario. A pessimistic upper bound out of the realization frame translates in a requirement to produce data sooner that the firing can be realized, which is a contradiction.

For final time frames respecting the inequalities of Equation (1), different approaches can then be used to assess the feasibility of the system timing requirements defining these frames, taking into account a scheduling policy and execution resources. We explore and characterize some approaches in the following sections.

B. Execution model

In this section, we suppose now that a polygraph system is implemented by mapping actors to a set of real-time, partitioned, fixed-priority tasks, connected through first-in, first-out (FIFO) message queues, where each token on channel represents a distinct message. In a service-oriented implementation, these message queues are typically implemented as a publish/subscribe channel supported by a communication middleware such as ROS, DDS, MQTT or ZMQ. We suppose that a common time source is available to all tasks.

We consider that each actor V_j is implemented by a single task, which loops infinitely over the following 4 states. For job index $n = 1, 2, \dots$:

- **1. Wait:** If V_j is strictly timed, i.e. has a period/phase offset constraint specified, block until the allowed time frame lower bound is met. If V_j is loosely-timed, switch immediately to *Pend* state.
- **2. Pend:** block until all the required input data is available, that is for each input channel E_i of actor V_j , block until at least $\rho_i(n) - \rho_i(n-1)$ tokens are available in the corresponding FIFO.
- **3. Compute:** this stage performs the application-specific business logic computation associated with this job. This typically involves processing the payload of the available input messages and computing the payload of the output messages, using CPU time less than or equal to the budget.
- **4. Send:** Block until the start of the realization time frame, then atomically pop required input data from the input queues and push the produced data samples to the output queues, that is on each output channel E_i , $\sigma_i(n) - \sigma_i(n-1)$ tokens.

We further suppose that during the *Wait* and *Pend* phases the processor is left free to run other jobs, i.e. the processor does not waste time in busy-waiting or input polling loops; and that the time spent in the communication stack in the *Send* phase is negligible – or at least, negligible in comparison to the time spent in the *Compute* phase. This is a reasonable hypothesis for several communication stacks such as ROS or ZMQ. For now, we also neglect message transmission times, i.e. a given message is available to its consumer immediately after it was sent by its producer job.

With these assumptions, and given the definition of the time frames in Section IV-A, scheduling a polygraph system

becomes equivalent to scheduling the *Compute* phase of all jobs, subject to the following constraints:

- the release, jitter and deadline constraints imposed on strictly-timed actors,
- the precedence constraints imposed by messages tokens.

Since all these time and precedence constraints can be predetermined off-line from the polygraph system definition, we can lean on the body of knowledge accumulated about precedence-constrained scheduling.

C. Absolute execution window

Scheduling using the allowed time frames (neglecting the budgets) gives the most flexibility to the execution (as the time frames are maximal). When guarantees on the feasibility of a schedule are required, it is best to account for budgets and use the pessimistic frames instead.

When a feasibility test is required, it is mandatory to have a characterization of the execution times of the jobs, we thus consider that BCET and WCET are available.

From there, setting the budgets to the BCET and refining the time frames as defined in Section IV-A provides pessimistic time frames that are the largest possible frames for the Compute phase of the jobs (since they are built considering that all jobs perform as fast as possible without processor resource constraints). We call this time frame for any job V_j^n its *absolute execution window*, noted $[aes_j^n, alf_j^n]$.

As a corollary we have a first *necessary* condition on schedulability: if, for any actor/job indices j, n , the absolute execution window is not long enough for the worst-case execution time, i.e. $aes_j^n + WCET_j > alf_j^n$, then it is not possible to schedule that job within these bounds. Such a system is proven unfeasible.

If this condition is verified, it makes sense to analyze further, by accounting contention on each processor resources.

D. Priority-driven scheduling

If we further suppose that each actor V_j is scheduled, using the Fixed-Priority Preemptive (FPP) policy, on a processor π_j , with distinct priority P_j , then we can refine our estimates of when each job will be executed, and whether processors can schedule all jobs.

In priority-driven scheduling, several jobs might compete for execution time on the same processor. Henceforth a high-priority job might defer or preempt a lower-priority job, hence causing *scheduling interference* (as illustrated in Fig. 5). To model the execution of tasks in the FPP policy, we introduce the following notions.

For any job V_j^n in a FPP schedule, we denote by s_j^n the start time of its Compute phase, f_j^n the completion of its Compute phase, and C_j^n its execution time.

In addition, we denote by if_j^n the amount of *interference* that V_j^n suffers from other, higher-priority jobs (i.e. the total time during which V_j^n is ready to run, but not actually running, on processor π_j). Finally, we denote R_j^n the *response time* $R_j^n = C_j^n + if_j^n$.

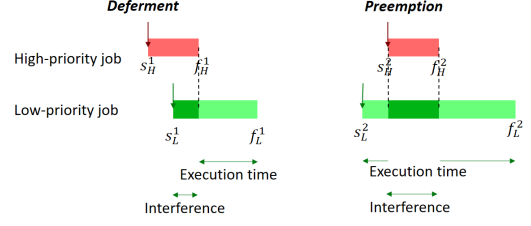


Fig. 5. Scheduling interference caused by a high-priority job HP on a lower-priority job LP. Jobs become ready at \downarrow . Bright green: LP job is running; dark green: LP suffers interference from HP.

The FPP schedule is then subject to the following constraints:

$$\begin{aligned} \forall V_j^n \in \mathcal{J}, f_j^n &= s_j^n + R_j^n \\ BCET_j &\leq C_j^n \leq WCET_j \\ aes_j^n &\leq s_j^n \\ f_j^n &\leq alf_j^n \\ \forall (V_j^n, V_k^p), V_j^n \prec V_k^p &\Rightarrow f_j^n \leq s_k^p \end{aligned} \quad (2)$$

Our task is now to estimate, or bound, the amount of interference that any job suffers.

E. FPP simulation

In order to estimate interference, we developed a simulator for FPP-scheduled polygraph systems. This simulator is derived from Equation (2), using a *discrete-event simulation* strategy: at each scheduling point, we compute the state of each actor, and each processor elects the highest priority job among those in the *Compute* state, to run until completion or preemption by another event. Fig. 6 illustrates the chronogram generated by this simulation.

If actor jobs have constant execution times (i.e. $BCET_j = WCET_j$), such a simulation might provide a sufficient schedulability test. If execution times are variable however, it is not sufficient to simulate a schedule with all actors' $WCET$ to get a worst-case response time estimate. Indeed, in certain circumstances, a job taking less time to execute might cause another job to have a longer response time and even miss a deadline. Fig. 7 illustrates such a *scheduling anomaly*, where the first job of A executes for its full $WCET$ duration, then releases the first job of B after C has completed. During the second run at $t = 40ms$ however, A completes earlier, releasing B sooner. This time, B preempts C , increasing its response time.

To build a higher confidence in system schedulability, we simulate many runs of the system (actually, many hyperperiods) in a Monte-Carlo fashion. For each job simulated, a execution time sample is drawn from a uniform distribution between $BCET_j$ and $WCET_j$. The outcome of the simulation is a chronogram, and a histogram for each actor of its response times such Fig. 8. Remember that the maximum *observed* response time is only a lower bound on the actual maximum response time, therefore this method does not constitute a

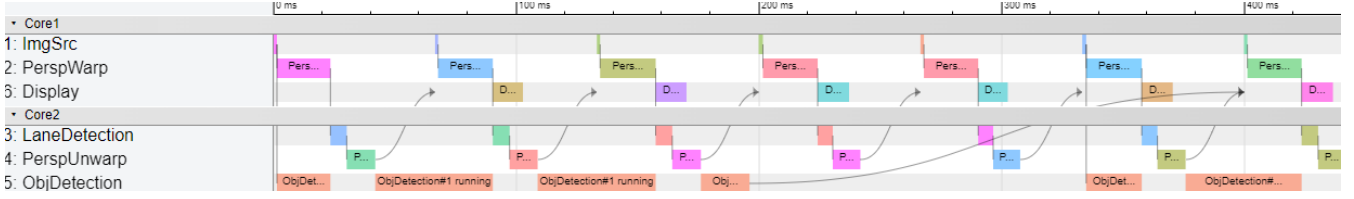


Fig. 6. Simulated FPP schedule

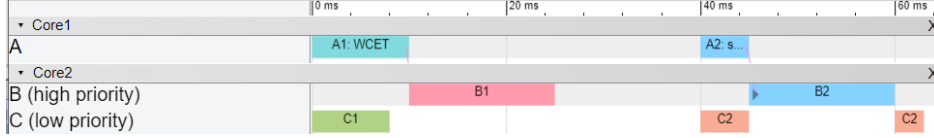


Fig. 7. Illustration of a scheduling anomaly: a shorter execution of actor A at $t = 40ms$ leads to a longer response time of C .

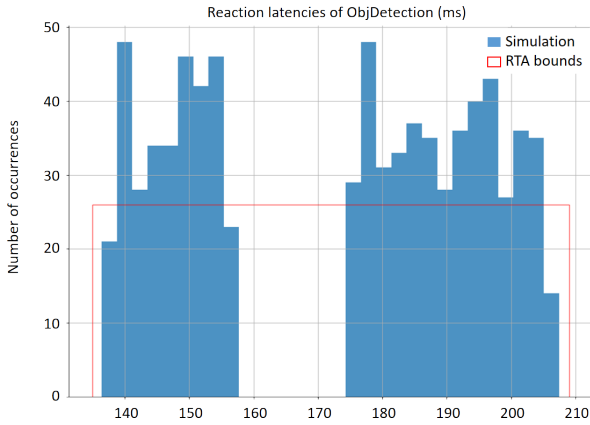


Fig. 8. Response latency histogram of the ObjDetection actor.

sufficient schedulability test. Indeed, a rare interference pattern might remain unveiled during any fixed-length simulated run.

F. Compositional performance analysis

In complement to the simulation approach, which provides a lower bound to job response times, several formal approaches are known to provide upper bounds. The Compositional Performance Analysis method seems an interesting approach, which supports modeling fixed-priority jobs with precedence constraints, deployed over multicore platforms.

We experimented with the open-source pyCPA library however, the response times estimates were very pessimistic. Indeed, current pyCPA implementation does not take into account the phase offsets in job release times. When we attempted to transform a polygraph job graph into a pyCPA precedence-constrained task system, many time constraints were therefore lost in translation, and as a consequence the pyCPA solver accounted many interferences that can not occur in practice. For this reason, we tried to reduce pessimism by adapting another method: the response-time analysis approach.

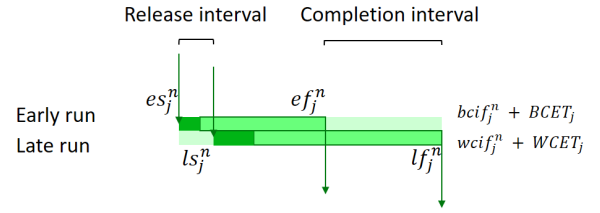


Fig. 9. Interval-based representation of job timing uncertainty. Dark green: best/worst-case total amount of interference; bright: best/worst case execution time.

G. Interval-based response time analysis

From the classical RTA algorithm, we derive an interval-based variant. Its main stage consists in rewriting the execution-time equation 2 using intervals to bound the uncertainty on each operand, then using the precedence graph to propagate these uncertainty intervals to neighboring jobs (similar to the method used for absolute execution windows), and then using these intervals to estimate possible job interference.

If we suppose that the release time of a job V_j^n lies within an interval $s_j^n \in [es_j^n, ls_j^n]$ (early start, late start), and that the total interference is bounded within $if_j^n \in [bcif_j^n, wci_f_j^n]$, then from (2) we can bound the job completion time $f_j^n \in [ef_j^n, lf_j^n]$ (early finish, late finish) with:

$$ef_j^n = es_j^n + BCET_j + bcif_j^n \quad (3)$$

and

$$lf_j^n = ls_j^n + WCET_j + wci_f_j^n \quad (4)$$

Fig. 9 illustrates the notations for early/late start and finish times, and their relation with best/worst-case interference and execution times.

Equation (4) resembles the classic RTA update equation. Note that updating a job's completion date might result in discovering new interfering jobs, so the classic RTA algorithm is applied recursively, until either fix point convergence or a deadline miss is identified. In our case, another argument calls for recursive application: updating the completion interval of a job might also postpone the release time of its successor jobs.

Now let's estimate the interference that a job V_j^n might suffer from another job V_k^p , noted as above as an interval $[bcif(V_j^n, V_k^p), wCIF(V_j^n, V_k^p)]$. Consider two arbitrary jobs, with their execution windows as indicated by their early and late start/finish dates. A job V_k^p might defer or preempt a job V_j^n , therefore cause interference, if and only if:

- both are deployed on the same processor, and the victim job has lower priority than its interferer, i.e. $\pi_j = \pi_k$, and $P_j < P_k$,
- and at some point in time, they are simultaneously ready to run.

Remark that if we're sure that an interferer's release time occurs while the victim is running, then the interferer will *certainly* interfere with the victim. We therefore set $bcif(V_j^n, V_k^p) = BCET_k$ when $[es_k^p, ls_k^p] \subseteq [ls_j^n, ef_j^n]$. In all other case, 0 is a safe lower bound to interference.

Similarly, an interference can occur only if the execution frame of the interferer overlaps that of the victim: $wCIF(V_j^n, V_k^p) = 0$ when $[es_j^n, lf_j^n] \cap [es_k^p, lf_k^p] = \emptyset$, in all other cases, $WCET_k$ is a safe upper bound.

By further noting that two jobs can't interfere with each other if they are linked by a precedence chain, we can reduce the set of potentially interfering jobs to consider:

$$HP(V_j^n) = \{V_k^p | \pi_j = \pi_k \text{ and } P_j < P_k \text{ and } V_k^p \notin Pred^*(V_j^n) \text{ and } V_j^n \notin Pred^*(V_k^p)\}$$

Hence, we define:

$$bcif(V_j^n, V_k^p) = \begin{cases} BCET_k & \text{if } V_k^p \in HP(V_j^n) \text{ and} \\ [es_k^p, ls_k^p] \subseteq [ls_j^n, ef_j^n], & \\ 0 & \text{otherwise} \end{cases} \quad (5)$$

and

$$wCIF(V_j^n, V_k^p) = \begin{cases} WCET_k & \text{if } V_k^p \in HP(V_j^n) \text{ and} \\ [es_k^p, lf_k^p] \cap [es_j^n, lf_j^n] \neq \emptyset, & \\ 0 & \text{otherwise} \end{cases} \quad (6)$$

It follows that a lower bound the total amount of interference suffered by job V_j^n is the cumulative interference from all jobs: $bcif_j^n = \sum_{V_k^p \in HP(V_j^n)} bcif(V_j^n, V_k^p)$. It might be tempting to similarly sum all $wCIF(V_j^n, V_k^p)$, however that would lead to accounting multiple times across a precedence chain the interference from a single job: Fig. 10 shows a schedule where a single high-priority job HP which might interfere with either $LP2$ (at $t = 100ms$) or $LP1$ (at $t = 1100ms$), which are part of a precedence chain $LP1 \rightarrow LP2$; but HP can obviously not interfere simultaneously with both. For this reason, we add to $wCIF_j^n$ only the interference that was not already accounted on predecessors of V_j^n , i.e.:

$$wCIF_j^n = \sum_{V_k^p \in HP^*(V_j^n)} wCIF(V_j^n, V_k^p)$$

with

$$HP^*(V_j^n) = HP(V_j^n) \setminus \{V_k^p | \exists V_l^q \in Pred^*(V_j^n), wCIF(V_l^q, V_k^p) > 0\}$$

This last remark leads to the interval-based response-time analysis Algorithm 1.

Algorithm 1 Interval-based response-time analysis

Input: polygraph system

Output: early and late release/completion times

Initialisation: for all jobs

$$es_j^n \leftarrow aes_j^n$$

$$ls_j^n \leftarrow aes_j^n$$

$$ef_j^n \leftarrow aes_j^n + BCET_j$$

$$lf_j^n \leftarrow aes_j^n + WCET_j$$

repeat

for all jobs in topological order **do**

Update the release interval

$$es_j^n \leftarrow \max(aes_j^n, \max_{V_k^p \in Pred(V_j^n)} ef_k^p)$$

$$ls_j^n \leftarrow \max(aes_j^n, \max_{V_k^p \in Pred(V_j^n)} lf_k^p)$$

Update best/worst case interference

$$bcif_j^n \leftarrow \sum_{V_k^p \in HP(V_j^n)} bcif(V_j^n, V_k^p)$$

$$wCIF_j^n \leftarrow \sum_{V_k^p \in HP^*(V_j^n)} wCIF(V_j^n, V_k^p)$$

Update completion interval

$$ef_j^n \leftarrow es_j^n + BCET_j + bcif_j^n$$

$$lf_j^n \leftarrow ls_j^n + WCET_j + wCIF_j^n$$

if $lf_j^n > al_j^n$, schedulability is not guaranteed

if $ef_j^n > al_j^n$, deadline miss is certain; return.

end for

until fix-point convergence

If this algorithm converges, then the system is schedulable, and for each job V_j^n we have computed a lower- and upper-bound to its release and completion times. Fig. 12 visualizes the outcome in a chronogram: each job is represented by two slices: an "early run" between es_j^n and ef_j^n and a "late run" between ls_j^n and lf_j^n .

H. Evaluation

Our performance analysis was evaluated on both our ADAS-inspired use-case, as well as with the use-case proposed by [13], which illustrates a precedence-constraint task set similar to our model.

Reference [13] presents a DAG task model with 2 tasks, detailed in a total of 8 precedence-constrained sub-tasks which are deployed on a 2-processor machine. By modeling message transmission times as additional non-interfering sub-tasks, this DAG task model injects easily into our PolyGraph language, which allows us to analyze it through both simulation and interval-based RTA analysis. This comparison confirms that for most tasks both [13]'s probabilistic method and our interval-based RTA algorithm gives a tight bound, equal to the maximum response time observed in simulation for most sub-tasks, with an overestimation corresponding to 1 message transmission time for two of them.

If we chose to deploy our ADAS use-case on a 2-core machine with the parameters in Table I, Core 2 is overloaded. As a result, both the simulation and interval-based RTA detect a deadline miss. Since the miss probability is low (see the histogram in Fig. 11), many hyperperiods are simulated before detecting a deadline miss.

If however we change the configuration, for instance by deploying the *PerspWarp* actor on core 1 instead, the system

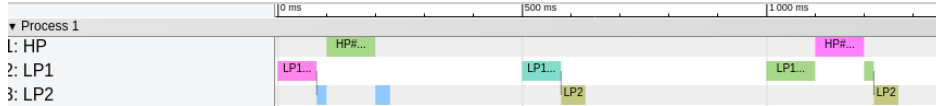


Fig. 10. Example interference on a precedence chain: high-priority job HP may defer or preempt either LP1 or LP2.

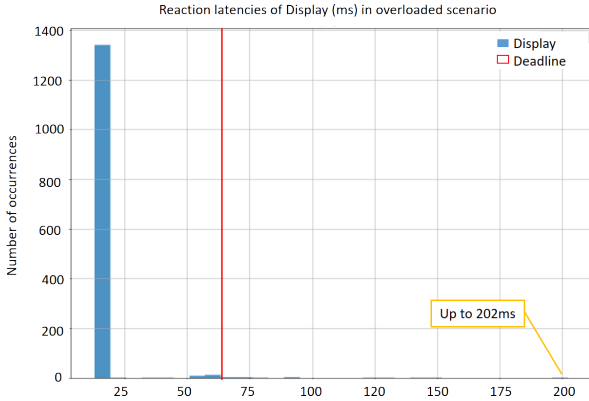


Fig. 11. Response time histogram of the Display actor, in overloaded configuration

Actor	Core	Prio.	BCET - WCET (ms)	Simul. WCRL (ms)	RTA WCRL (ms)
ImgSrc	1	5	1-2	2	2
PerspWarp	2	4	22-23	24.9	25
LaneDetection	2	3	6-7	31.8	32
PerspUnwarp	2	2	11-12	43.8	44
ObjDetection	2	0	100-150	206.7	209
Display	1	1	12-13	37.9	38

TABLE I

ADAS USE-CASE PARAMETERS AND WORST-CASE RESPONSE LATENCIES

becomes schedulable. The last two columns of Table I show the worst-case response latency computed by our interval-based RTA algorithm, and observed over a simulation of 100 hyperperiods. Since the simulation approach provides a lower bound on the actual maximum response times, and the interval-based RTA proves an upper bound, the small gap between both predictions indicates that we have a rather accurate estimation of the actual maximum possible response times.

In addition, from this polygraph system definition we generated a set of task threads communicating through ZMQ messages, respecting the same scheduling parameters. We ran the generated code on a physical multicore target and traced the effective execution during a few minutes. The actual trace of Fig. 13 confirms that each job ran within the early/late bounds observed in simulation and computed with interval-based RTA.

V. CONCLUSION

The PolyGraph language supports modeling of a rich set of SDV applications, typically in the form of reaction chains involving services running at various rates, deployed over multi-core and distributed ECUs. Equipped with the model

of execution proposed in Section IV-B, we have shown that two powerful verification methods can be extended to support the timing analysis of polygraph systems, namely scheduling simulation (Section IV-E) and an interval-based variant of the response-time analysis method (Section IV-G). This approach is validated by comparing the response latencies measured on a simulation, computed with the interval-based RTA method, and measured on an actual target execution.

This paper focuses on the partitioned, fixed-priority preemptive scheduling policy, however both the simulation and response-time analysis tools could be extended as future work, to support other policies such as deadline-driven policies – especially the Constant-Bandwidth Server configuration which is also often used in SDV applications. Similarly, the effect of network scheduling policies on message transmission times (typically in Time-Sensitive Networking configurations) could be integrated in our analysis to refine the estimated response latencies in network-distributed applications. Moreover, we consider adapting some heuristics that have been proposed in literature to assign priorities and explore partitioning configurations.

REFERENCES

- [1] Casini, D., Blaß, T., Lütkebohle, I., & Brandenburg, B. (2019). Response-time analysis of ROS 2 processing chains under reservation-based scheduling. In 31st Euromicro Conference on Real-Time Systems (pp. 1-23). Schloss Dagstuhl.
- [2] Lee, E. A., & Messerschmitt, D. G. (1987). Synchronous data flow. Proceedings of the IEEE, 75(9), 1235-1245.
- [3] Lee, E. A., & Messerschmitt, D. G. (1987). Static scheduling of synchronous data flow programs for digital signal processing. IEEE Transactions on computers, 100(1), 24-35.
- [4] Roumage, G., Azaiez, S., & Louise, S. (2022, December). A survey of main dataflow MoCCs for CPS design and verification. In 2022 IEEE 15th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoc) (pp. 1-9). IEEE.
- [5] Potop-Butucaru, D., De Simone, R., & Talpin, J. P. (2018). Synchronous hypothesis and polychronous languages. Embedded Systems Handbook: Embedded Systems Design and Verification.
- [6] Kopetz, H., & Bauer, G. (2003). The time-triggered architecture. Proceedings of the IEEE, 91(1), 112-126.
- [7] Kirsch, C. M., & Sokolova, A. (2012). The logical execution time paradigm. Advances in Real-Time Systems, 103-120.
- [8] Breaban, G., Stuijk, S., & Goossens, K. (2017, March). Efficient synchronization methods for LET-based applications on a multi-processor system on chip. In Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017 (pp. 1721-1726). IEEE.
- [9] Dubrulle, P., Gaston, C., Kosmatov, N., Lapitre, A., & Louise, S. (2019, April). A data flow model with frequency arithmetic. In International Conference on Fundamental Approaches to Software Engineering (pp. 369-385). Cham: Springer International Publishing.
- [10] Harter, Paul K. Response times in level-structured systems. ACM Transactions on Computer Systems 5 (1987): 232-248.
- [11] Audsley N., Burns A., Richardson M., Tindell K., & Wellings A.J. Applying New scheduling Theory to Static Priority Preemptive Scheduling. Software Engineering Journal, 8(5):284-292, 1993.

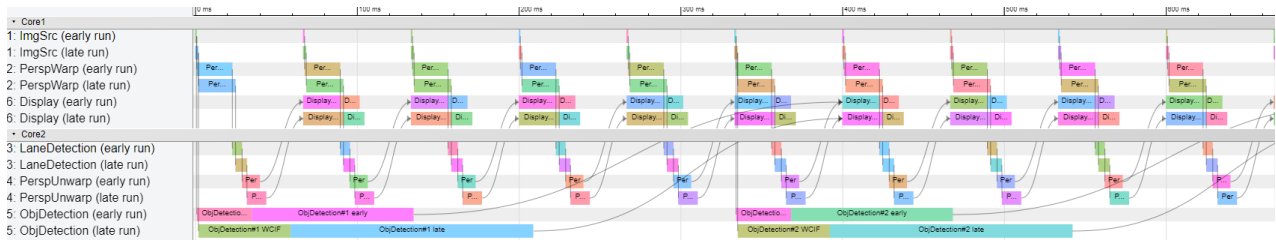


Fig. 12. RTA analysis of our ADAS use-case

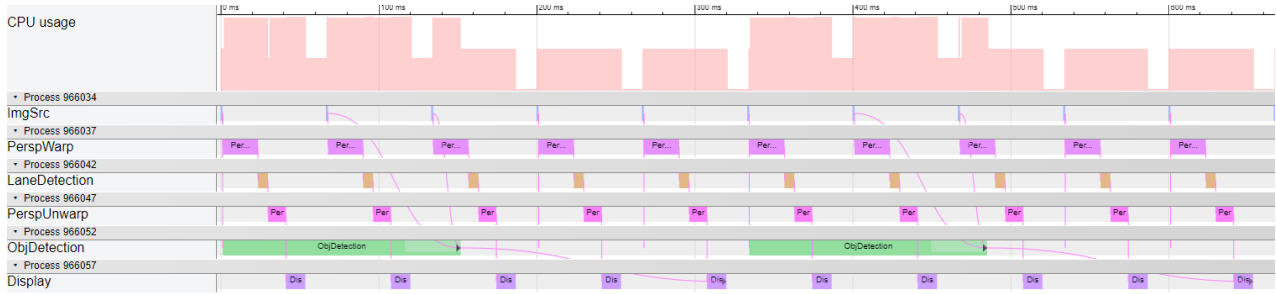


Fig. 13. Trace from a physical execution

[12] Langer T., Osinski L., & Mottok J. (2017, April). A survey of parallel hard-real time scheduling on task models and scheduling approaches, In 30th International Conference on Architecture of Computing Systems.

[13] Slim, B. A., Liliana, C. G., Mezouak, M., & Sorel, Y. (2020, September). Probabilistic schedulability analysis for precedence constrained tasks on partitioned multi-core. In 2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA) (Vol. 1, pp. 345-352). IEEE.

[14] Henia, R., Hamann, A., Jersak, M., Racu, R., Richter, K., & Ernst, R. (2005). System level performance analysis—the SymTA/S approach. IEE Proceedings-Computers and Digital Techniques, 152(2), 148-166.

[15] Carle, T., Potop-Butucaru, D., & Lesens, D. (2015). From dataflow specification to multiprocessor partitioned time-triggered real-time implementation. Leibniz Transactions on Embedded Systems.

[16] Honorat, A., Tran, H. N., Gautier, T., Besnard, L., Bhattacharyya, S. S., & Talpin, J. P. (2023). Real-Time Fixed Priority Scheduling Synthesis using Affine DataFlow Graphs: from Theory to Practice. ACM Transactions on Embedded Computing Systems.

[17] Dubrulle, P., Kosmatov, N., Gaston, C., & Lapitre, A. (2021). PolyGraph: a data flow model with frequency arithmetic. International Journal on Software Tools for Technology Transfer, 23(3), 489-517.

[18] Hamelin, E., Asavao, M., Azaiez, S., Berne, A., Faure, C., & Trabelsi, K. (2022, June). Multilayer monitoring for real-time applications. In ERTS 2022-11th European Congress Embedded Real Time Systems.

[19] Dubrulle, P., Gaston, C., Kosmatov, N., & Lapitre, A. (2019, November). Dynamic reconfigurations in frequency constrained data flow. In International Conference on Integrated Formal Methods (pp. 175-193). Cham: Springer International Publishing.

[20] R. Pathan, P. Voudouris and P. Stenström, “Scheduling Parallel Real-Time Recurrent Tasks on Multicore Platforms,” IEEE Trans. on Parallel and Distributed Systems, vol. 29, no. 4, pp. 915-928, 1 April 2018, doi: 10.1109/TPDS.2017.2777449.

[21] A. Singh, P. Ekberg, S. Baruah, “Uniprocessor scheduling of real-time synchronous dataflow tasks.” Real-Time Syst. vol.55, pp.1-31, 2019.

Towards the Certification of Hybrid Architectures: Analysing Interference on Hardware Accelerators through PML

Benjamin Lesage¹, Frederic Boniol¹, Kevin Delmas¹, Adrien Gauffriau², Alfonso Mascarenas-Gonzalez¹, Claire Pagetti¹
¹ ONERA, Toulouse, France, ² Airbus, Toulouse, France

Abstract—The emergence of Deep Neural Network (DNN) and machine learning-based applications paved the way for a new generation of hybrid hardware platforms. Hybrid platforms embed several cores and accelerators in a small package. However, in order to satisfy the Size, Weight and Power (SWaP) constraints, limited and shared resources are integrated. This paper presents an overview of the standards applicable to the certification of hybrid platforms and an early mapping of their objectives to said platforms. In particular, we consider how the classification of AMC20-152A for airborne electronic hardware applies to hybrid platforms. We also consider AMC20-193 for multi-core platforms, and how this standard fits different types of accelerators.

I. INTRODUCTION

New software paradigms and capabilities drive the demand for additional computing power in avionic systems. Hybrid architectures can, in a small SWaP package, support this demand. They embed on the same platform general-purpose cores, and specialised accelerators which can support some of the additional workload. However, like any other hardware platform, they need to go through a stringent certification process before they are deployed in avionic system.

The European Union Aviation Safety Agency (EASA) and Federal Aviation Administration (FAA) respectively define Acceptable Means of Compliance (AMC) and Advisory Circulars (AC), setting down objectives applicants to the certification process satisfy. The joint A(M)C AMC20-152A and AMC20-193 in particular define objectives for the respective certification of hardware platforms and multi-core processors.

The PHYLOG methodology [1] was proposed as mean of supporting applicants, especially regarding AMC20-193 on multi-core processors. PHYLOG is based on the definition of argumentation patterns for the certification objectives in AMC20-193, with each objective decomposed in supporting claims, strategies, or evidences. At the core of the methodology, the PHYLOG Modelling Language (PML) [2] captures knowledge about a platform, both hardware and software aspects, and their configuration. PML supports analyses to fulfil claims in the certification patterns instantiated for the platform.

The contributions of this paper are to present an overview of the objectives applicable to hybrid platforms. We also identify the issues related to modelling the accelerators in such platforms and propose related PML model templates. This paper is organised as follows. Section II briefly recaps

the PHYLOG methodology, with Section III providing an introduction to PML. An example of accelerator and its hybrid platform is introduced in Section IV to support further discussions and examples. In the context of hybrid platforms, we identified two relevant AMC: AMC20-152A [3] and AMC20-193 [4] discussed respectively in Section V and Section VI. Section VII briefly discusses related work, before Section VIII recaps the discussion and outlines perspectives.

II. PHYLOG METHODOLOGY

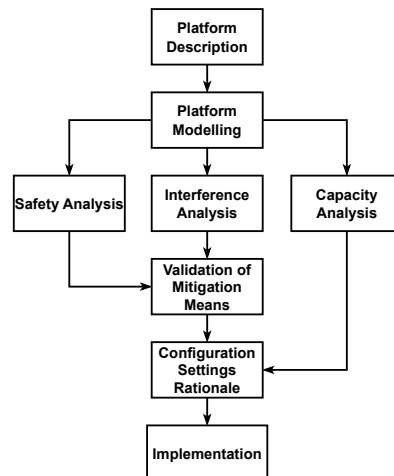


Fig. 1. Overview of PHYLOG methodology

The PHYLOG methodology [1] describes the activities to produce the elements for instantiating the PHYLOG argumentation patterns. These patterns were derived from the objectives defined in AMC20-193, on multi-core processors, to build an argumentation strategy for certification. They decompose the top-level AMC objectives into supporting claims, strategies, evidence, and warrants. An overview of the methodology is presented in Figure 1. It is composed of eight main activities:

- **Platform description** captures the knowledge about the platform characteristics based on the available documents and the applicant's assessments. It also captures the target configuration, including hardware and software settings such as the mapping of applications hosted on the platform to cores.

- **Platform modelling** formalises the platform description knowledge in order to support further analyses. It is based on PML. While not an objective of AMC20-193, it allows running the supporting automatic safety, capacity and interference analyses in order to contribute to said objectives.
- **Safety analysis** identifies and evaluates the failures and alterations which can affect the platform and hosted applications.
- **Interference analysis** enables the identification of interferences via interference calculus and the classification of their effects.
- **Capacity analysis** enables the verification of shared resources' usage, ensuring the demand for resources of the platform never exceeds their capacity.
- **Validation of mitigation means** encompasses the design and validation of mitigation means for failure, interference, and other alterations identified in earlier activities.
- **Configuration settings rationale** justifies that all configuration settings support the requirements on the platform, or are harmless to them.
- **Implementation** concerns the certification of the system implementation on the platform. It is associated with the DO-178C standard and out of the context of PHYLOG.

Note that the activities form an inherently iterative process. As an example, the interference analysis may highlight a misunderstood interference channel, feeding back into the platform description and its model.

We focus in the following on the platform aspects (description and modelling), as they are the most relevant to hybrid platforms. We consider specifically the use of PML, and its limitations, to model accelerators. The use of PML would thus allow for the application of existing PHYLOG-based analyses [1], discussed in other work for interference or safety, to instantiate the PHYLOG certification patterns for hybrid platforms. PML is introduced in the next section.

III. PML

PML, the PHYLOG Modelling Language [2], is a Domain Specific Language embedded with the SCALA language to capture the description of a platform. A hardware platform is modelled in PML as a collection of components, capturing the functional blocks of a multi-core processor, e.g. a core, cache, memory, or bus, and links between components. Composite components encapsulate one or more components, composite or atomic, to allow for the hierarchical specification of a model. *Atomic* components provide generic services to the software hosted by the platform, such as a *load* from the main memory or a *store* to a configuration register.

The relationship between a component and other services of the platform defines its role in the model. *Initiator* components, such as a core, call services from other components on the platform, most often as a result of software running on the initiator, be it a user application or platform-embedded micro-code. *Target* components, such as the main memory, expose services to satisfy transactions from other components.

Transporter components, such as an interconnect, process transactions between an initiator and its target.

A *transaction* is a footprint of a use of the platform by a software component. A transaction more formally captures the set of components, and their services, used by a request from an initiator to a target. A transaction must follow a valid path in the platform, through the links between its components. Services thus model the dependencies between the software and the hardware.

Example 1. To exemplify the use of PML, we consider a representation of the KEYSTONE TCI6630K2L from Texas Instruments. An overview of the KEYSTONE is presented in Figure 2. It is composed of a four C66 DSP pack where cores are characterised by dedicated L1 and L2 caches, and a memory extension and protection unit (MPAX). The platform also comprises a 2 ARM A15 pack where cores are characterised by dedicated L1 caches, memory management units (MMU), and a shared L2. In addition, it includes a central memory system giving access to SRAM and external DDR. Memory accesses are managed by a Multicore Shared Memory Controller (MSMC). A set of I/O and utility peripherals (e.g. GPIO, UART, boot) is also present on the platform and an ultra speed bus (TeraNet) connects the peripherals, the memories, and the cores altogether.

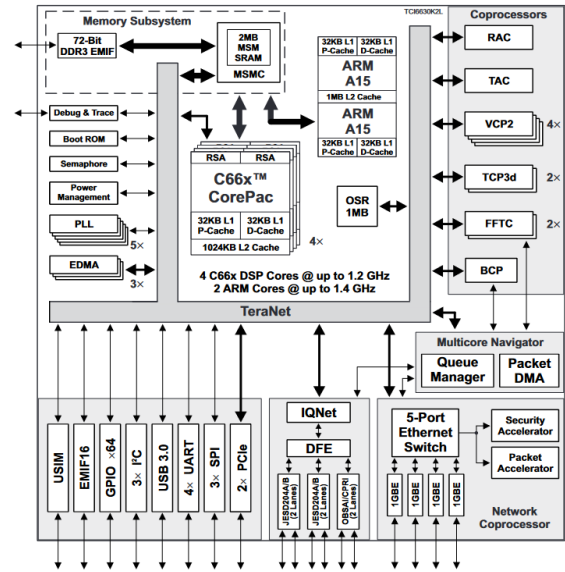


Fig. 2. Overview of the TI KEYSTONE TCI6630K2L

Figure 3 illustrates a PML model for a simplified version of the KEYSTONE¹. This basic model includes:

- Cores as initiators: 4 C66 DSP, and 2 ARM A15 cores;
- Memories as targets: DDR, SRAM, and all caches;
- Peripherals as targets: GPIO, I2C, SPI port, PCIe, etc.;

¹For the sake of brevity, coprocessors have been omitted, as well as implicit links between stacked components. Peripherals have been simply classified as targets.

- *Buses and Memory protection units as transporters: the TeraNet bus connected to the Memory Shared Multicore Controller (MSMC), memory and cache controllers, etc.*

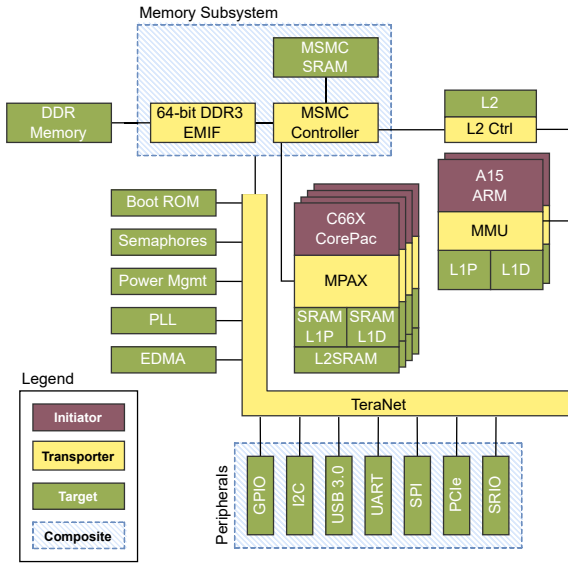


Fig. 3. Simplified PML model for the KEYSTONE platform

IV. HYBRID ARCHITECTURES - THE GPU EXAMPLE

To support the discussion around hybrid platforms, we introduce an example of accelerator: Graphical Processing Units (GPU). Compared to traditional CPUs, GPUs feature numerous cores with simpler control flow but efficient data ones. GPU cores tend to work in a lockstep-like fashion called Single Instruction Multiple Threads (SIMT) in reference to SIMD (Single Instruction Multiple Data). Internal scheduling policies on the GPU aim to maximise core occupancy and throughput. With their focus on high-throughput floating point computation, GPU are well suited to the acceleration of neural network workloads. Their reuse has been facilitated by the advent of General Purpose GPU programming frameworks (GPGPU).

There has been considerable effort to characterise the behaviour of GPU accelerators, in particular work on NVIDIA GPU [5], [6], [7], [8] and the assorted GPGPU CUDA software stack [9], [7], [10], [8]. These efforts highlight the difficulty of characterising complex, multi-core, COTS (Commercially available Off-the-Shelf) platforms. To the best of our knowledge, PasTiS [6] and the hybrid analysis in [11] are some of the few efforts to build a GPU model respectively for static and hybrid WCET analysis. The inherent parallelism at the application-level, as opposed to instruction-level like vectorised arithmetic units [12], [13], can also pose problems for WCET and interference analyses [14].

Example 2. *The NVIDIA Jetson AGX Xavier [15] is a high-performance SoC designed for embedded systems. The Xavier uses an 8-core “Carmel” ARM processor, organised in clusters of 2 cores. The “Carmel” processor complies the ARM v8.2A*

specification, but it is unclear if it is based off an existing ARM design (e.g. the Cortex-A78) and the level of customisation introduced by NVIDIA. The Xavier features amongst other accelerators a GPU using the Volta architecture, highlighted in Figure 4. The GPU is composed of 512 cores, grouped in 8 Streaming Multiprocessors (SM). The Volta GPU shares a memory fabric with other accelerators, and the memory controller with the CPU.

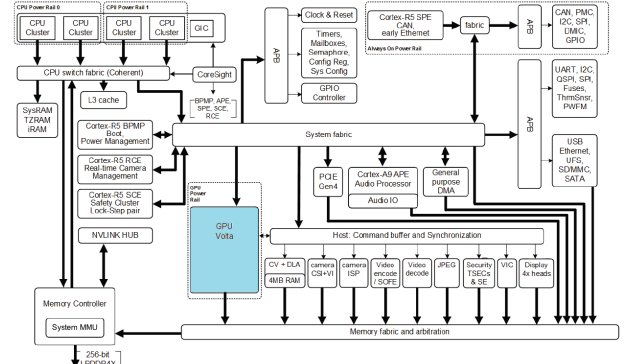


Fig. 4. Overview of the NVIDIA Jetson AGX Xavier

We present in Figure 5 a high-level PML model of the NVIDIA AGX Xavier SoC. Fabrics and backbones act as transporters for the components of the system. The main memory is a target shared by the CPU and the GPU. The cores of the “Carmel” ARM processor act as multiple initiators. As for the KEYSTONE, we currently omit coprocessors and peripherals from the classification. A key question is: How to model a complex accelerator like the Volta GPU? It acts as an initiator, causing interference on the main memory and the controller fabric, and as a target for commands from the CPU.

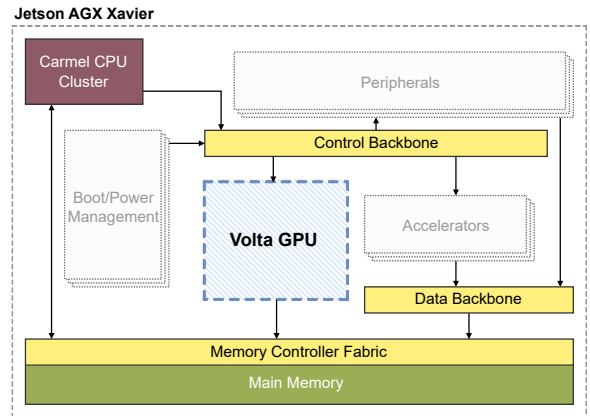


Fig. 5. Simplified PML model for the NVIDIA AGX Xavier

V. AMC20-152A ON HYBRID ARCHITECTURES

AMC20-152A discusses the certification of existing (COTS) or newly-developed platforms, the distinction between the two, and the objectives relevant to each.

A. Overview of the AMC20-152A

The ED-80/DO-254, both dated from the year 2000, define guidance for the design of airborne electronic hardware. The AMC20-152A aims to provide additional guidance and clarification. It is thus complementary to the AMC20-193 on multi-core platforms. The clarifications proposed by the AMC20-152A are important, as devices, especially COTS, become more complex and integrate in a single chip more functions than older ones. The AMC20-152A objectives are classified according to whether they apply to complex custom devices, COTS IP (design functions used to design and implement a custom device, be it a PLD, a FPGA or an ASIC), or COTS devices². Applicants for certification must address them as part of the Plan for Hardware Aspects of Certification (PHAC), or related planning documents.

The first distinction in the AMC20-152A is between COTS and custom functions. COTS functions (IP or devices) are, as the name implies, commercially available, off-the-shelf. The AMC20-152A recognises the risks inherent to the use of COTS, and incomplete or incorrect documentation. COTS may not have been developed within the ED-80/DO-254 standard or avionic applications, nor have sufficient service experience. The development assurance for COTS items (hardware or software components ED-80/DO-254) thus follows different objectives from custom devices. Items developed and fully controlled by the applicant cannot classify as COTS. Those items may however be previously developed hardware, which may take credit from prior deployment and in-service experience provided their new function, usage and environment conditions do not invalidate the original design assurance.

The key objectives of the process for COTS items are 1) identifying used functions, and 2) assessing correct use of the COTS item. The used functions need to support the system requirements on the device. Unused functions, such as unused cores on a MCP (as per AMC20-193), need to be properly deactivated, with means of mitigation to prevent their inadvertent activation. Correct use of a COTS item requires to assess its integration against the operation conditions, such as temperature or input parameter ranges, defined by the manufacturer. This may preclude the use of undefined or undocumented configurations, unless their reliability can be established. The identification of failure modes³ and the item configuration also need to be considered. This includes identifying if any microcode may contribute to a used function. Microcode is a hardware-level set of instructions, typically stored in the COTS item. It may be qualified by the manufacturer, if left unmodified, or require a separate mean of compliance.

Devices are further classified into simple or complex ones as defined by the ED-80/DO-254. The classification captures whether a comprehensive verification of the device is realistic.

²We omit circuit boards assemblies (CBA), as the AMC20-152A in practice redirects to ED-80/DO-254.

³Single Event Effects (SEE) are explicitly omitted from the AMC20-152A scope.

It must be explicit, and justified for simple devices (custom or COTS). The simplicity of a device relies on the simplicity and independence of all its functions, interfaces, building blocks, etc. The composition of simple items may therefore be a complex item.

B. Considerations for accelerator-related objectives

As per the AMC20-152A, most hybrid or multi-core architectures should fall under the definition of complex devices with multiple processing elements interacting. The Platform description and modelling phase for custom models, including any accelerator, will directly benefit from the AMC20-152A objectives' outcome, notably the conceptual and detail designs, and the device verification. For COTS functions, as prescribed by the AMC20-152A objectives, a PML model should be built from the manufacturer specification supplemented by characterisation and verification activities. COTS IP specifically may provide detailed information on the function based on the stage of the design where they are instantiated, from Hard IP, embedded in the silicon by the manufacturer, to Soft ones, captured by a hardware description language. Microcode, if present on used functions, needs to be considered as part of the platform model, as transactions between components.

We identified 4 activities for hybrid platforms and accelerators, per AMC20-152A objectives: **Activity 1:** An assessment should be performed for each device or its integration, as they may fall under different classifications: COTS, custom, soft IP, hard IP, multi-core processor... In particular, one should consider how the device is configured and accessed through hardware and software means, how it interacts with the rest of the system, and whether or not existing analysis techniques and tools apply.

Activity 2: It is necessary to master *complex* core architectures. More specifically stressing benchmarks would be needed in addition to documentation reviews.

Activity 3: The utilisation of COTS must be within the limit of the device manufacturer specification. This means that we need a specification of the COTS and its limits to check the compliance of usage.

Activity 4: It is mandatory to qualify the COTS behaviour and all micro-code, as defined in AMC20-152A (Section V-A).

VI. AMC20-193 ON HYBRID ARCHITECTURES

The AMC20-193 was extensively studied in PHYLOG to define a certification methodology specifically for multi-core platforms [1]. We provide a brief summary of AMC20-193 in the following.

A. Overview of the AMC20-193

The AMC20-193 defines a Multi-Core Processor (multi-core processor) as a device with two or more activated processing cores, with a core being a device that executes software. The AMC20-193 recognises two exceptions to the definition of active cores, cores in lockstep executing the same software and inputs to compare their output; and cores connected solely through data buses typically used in avionics systems.

The AMC identifies both temporal and functional interference. Interference occurs when the behaviour of an application varies over its behaviour in isolation when running in parallel with others. Interference occurs as a result of shared hardware or software resources of the multi-core processor. As an example, interference may cause additional delays due to the arbitration of accesses to the resource or control flow variations due to external modifications of a shared variable. Interference may cause a loss of deterministic behaviour for the application.

All software components should exhibit correct functional and timing behaviours in the presence of interference. The AMC thus defines an interference channel as “a platform property that may cause interference between software applications or tasks”. The impact of interference channels on applications in the system should be assessed. The planning objectives in AMC20-193 require the identification of shared resources, their use by, and their allocation to software applications, where applicable. This aims to first ensure the overall demand for resources at any given time does not exceed the available resources’ capacity, and second to avoid or mitigate interference. Mitigations should be deployed and verified for impactful interference channels. The definition of an interference channel in the PHYLOG methodology is a conservative one, in line with the AMC objectives.

The objectives require all software hosted on the multi-core processor to be identified, including applications, operating systems, hypervisors, as well as libraries and runtime. The AMC20-193 prescribes that any component for which interference is mitigated, possibly at the platform-level through robust partitioning, may be separately analysed and verified. Otherwise, they should be tested on target with all other software components under the final configuration. The PHYLOG methodology, and in particular interference calculus, can help assessing whether a modelled accelerator or a platform supports robust partitioning, by identifying interference channels, their impact, and that of any deployed mitigation (through benchmarking).

The question in the context of accelerators, is whether or not the PML model is suitable to model them, and whether and how it should be extended. Let us now characterize what type of resource is an accelerator. We have identified 3 dimensions to take into account.

B. Dimension 1

The first dimension concerns the number of applications that can simultaneously access the accelerator. We define two categories within that dimension:

- those that can be accessed solely by one application at any given time are called *unitary accelerators*;
- those that can be accessed by multiple applications simultaneously are called *parallel accelerators*.

Note that the classification of an accelerator as *unitary* may be inherent to the accelerator itself, e.g. if it cannot support multiple applications by design, or enforced by the platform, e.g. through application design or partitioning mechanisms.

C. Dimension 2

The second dimension concerns how the accelerators are connected to the core and how the workload is launched. In that dimension, we have identified four categories. The simplest case concerns *tightly coupled* accelerators.

Category 1. *Tightly coupled accelerator.* *The accelerator, as an example a vectorised functional unit, operates in the context of a complex core; all transactions effectively originate from the core operations and transit through the core interfaces.*

Modelling impact on PML. *The core is still modelled as the sole initiator. Such an accelerator can only be unitary, as a core executes only one application at any given time⁴. However transactions caused by an application using the accelerator may present a different profile.*

Example 3 (of category 1). *The ARM A15 [16] cores can include a NEON VAU and floating point execution unit. SIMD Load/Store instructions allow for transfers between NEON registers and the memory. Vector accesses target one or more lanes of the same or of consecutive vector registers. The architecture thus does not guarantee the atomicity of the access to the memory even for scalar accesses. Each instruction can generate multiple transactions depending on the access size, the alignment of the address and the memory segment. Served by the private or shared caches, or the main memory, SIMD Load/Store may be subject to high timing variability and interference.*

The A15 cores in the KEYSTONE presented in Example 1 do feature a NEON VAU. As discussed, the core is still modelled as a single initiator and the model in Figure 3 remains valid even when the NEON is in use.

The second case concerns *passive* accelerators that are controlled by a remote core, e.g. via configuration registers. A passive accelerator cannot generate any transaction to access any shared resource and is thus a target that can be shared by several cores.

Category 2. *Passive accelerator.* *The accelerator is a resource used by the core(s). It behaves from a high level point of view like a DDR that receives requests for load and store.*

Modelling impact on PML. *It can be abstracted as a target. Two or more applications using the accelerator concurrently would be assumed to interfere. Thus it could be unitary or parallel, but in both cases it will be modelled in the same way. The transactions caused by the controlling core may present a different profile.*

Example 4 (of category 2). *The NVIDIA Deep Learning Accelerator (NVDLA) outlined in Figure 6 is an accelerator developed by NVIDIA, with both open-source hardware and software. The NVDLA is a complex COTS device. Tailored to neural network applications, it features functional blocks dedicated to convolution, activation functions, pooling, normalisation, or reshaping operations. The blocks can operate*

⁴AMC20-193 explicitly excludes hyperthreading.

independently, performing memory-to-memory operations, or pipelined, passing data to each other to avoid the memory round-trip. The memory (DBBIF), interrupt (IRQ), and configuration (CSB) interface can be connected to various protocols such as ARM AXI.

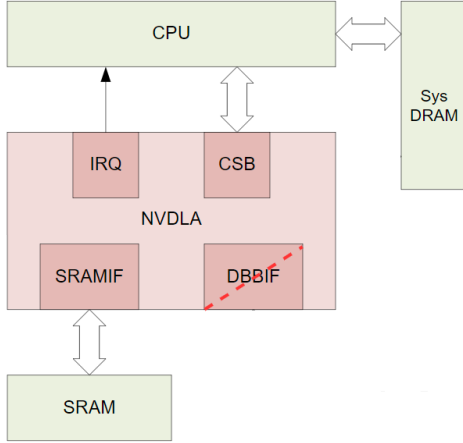


Fig. 6. Integration of the NVIDIA NVDLA in a passive configuration [17]

As a soft IP, the NVDLA exposes all information regarding its internal behaviour which eases the development of a model for timing or interference analysis. The DBBIF, CSB, and target memory subsystem are obviously shared resources between functional blocks. The scope and mitigation of any resulting interference however require more information about the NVDLA integration. The device can be included as part of custom devices or available in future COTS platforms.

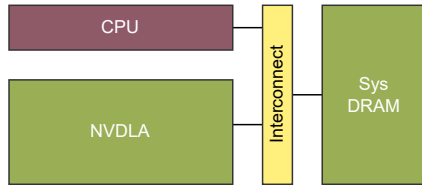


Fig. 7. Simplified PML model for the NVDLA in a passive configuration

Figure 7 presents a PML model for a NVDLA in a passive configuration. The accelerator and all its resources are abstracted as a single target, accessed through the interconnect. Transactions initiated within the NVDLA remain within the device, e.g. from its functional blocks to the CSB or SRAM. As such they would not need to be captured by the model. They are thus implicitly assumed to be non-interfering with external transactions, e.g. from the CPU to the CSB. Such an assumption must be verified during interference analysis.

The third case concerns semi-active accelerators. In that situation, the accelerator is triggered by a remote core but it accesses shared resources (e.g. DDR) to load/store its data. Thus it generates interferences within the hybrid architecture.

Category 3. Semi-active accelerator. The accelerator operates under the control of a core and it behaves from a high

level point of view as a DMA that generates requests for load and store under the impulse of another core. However the precise role of the core needs to be clarified, as well as the interface between the accelerator and the hybrid platform.

Modelling impact on PML. A unitary semi-active accelerator is thus modelled as a single initiator and the profile of the remote core must contain all the transactions needed to configure the accelerator. Parallel accelerators would need more refined analyses to check whether they will be decomposed into one or multiple initiators.

Example 5 (of category 3). An example is the NVDLA in a "Small" configuration as depicted in Figure 8. Compared to the passive configuration of Example 4, the NVDLA accesses resources shared with other initiators in the system. The NVDLA [17] could be modelled, as depicted in Figure 9, using a single initiator with interfaces to the system, as no interface or resource between the NVDLA and the controller is shared with other devices. This model assumes a pipelined configuration of execution on the NVDLA, where a single application may use the NVDLA and components do not interfere on the DBBIF. (Example 9 considers a configuration where each functional block is a separate initiator.)

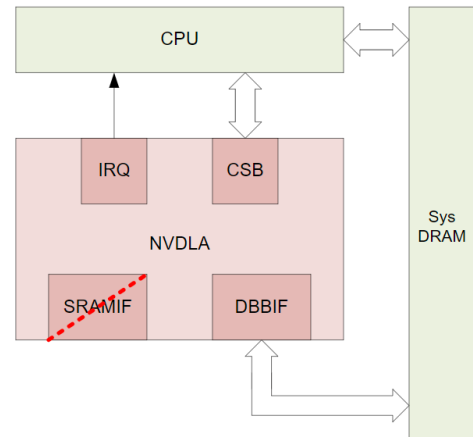


Fig. 8. Integration of the NVIDIA NVDLA in a small configuration [17]

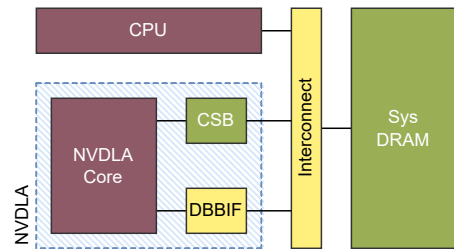


Fig. 9. Simplified PML model for the small NVDLA

Example 6 (of category 3). The i.MX 8M Plus processor from NXP [18] features, amongst other accelerators, a NPU, e.g. a VIP8000 hard IP from VeriSilicon. The NPU is a complex COTS device. The processor reference manual unfortunately

provides little information about the NPU, except for the high-level functional description in Figure 10. It probably features VAU and systolic-like blocks as it supports hundreds of multiply and accumulate operations every cycle. The interface with the processor uses ARM AXI and AHB bus interfaces which might help bound the demand of the NPU on the shared memory, and the interference it generates.

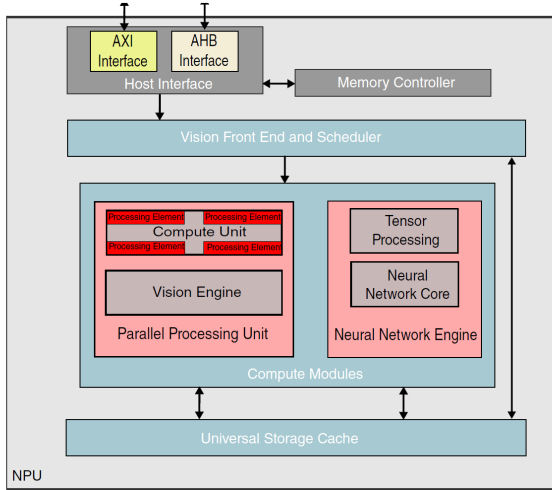


Fig. 10. NPU High-level Block Diagram in the i.MX 8M Plus processor [18]

It is difficult to model such a COTS device with no further information on its functional blocks, or without a characterisation by evaluation. It could be abstracted as a single initiator. This abstraction would need to be supported by limiting the use of the NPU as a unitary accelerator, e.g. through platform configuration. Furthermore, the abstraction will still require an assessment of the nature and volume of transactions the NPU generates.

The fourth case concerns active accelerators. An example of such accelerators are GPU.

Category 4. Active accelerator. The accelerator operates independently and generates many load and store transactions. **Modelling impact on PML.** A unitary accelerator is thus modelled as a single initiator where, as for semi-active accelerators, parallel accelerators would need more refined analyses to check whether they will be decomposed into one or more initiators.

Example 7 (of category 4). When the accelerator is a GPU used by a unique application at a time, it can be modelled as an initiator and single transaction forking to multiple targets should capture the combinations of behaviours of multiple threads running concurrently on the accelerator. Threads from the same application may not be considered as interfering with each other but with other applications in the system. The GPU scheduler decides upon execution of a computation kernel of the allocation of different blocks of threads to cores.

The scheduling policy on most COTS platforms is subject to speculation, and the allocation of threads to cores is dynamic.

In PML, the initiator of a transaction from a given thread would thus be uncertain as well as for AMC20-193. Modelling the GPU as a single initiator abstracts away this uncertainty. This should be a conservative, but sound abstraction for interference analysis between applications. It needs to be backed by the platform to ensure only one task accesses the GPU at any given time.

Example 8 (of category 4). When the GPU is used simultaneously by several applications, the GPU cannot probably be modelled as a single unit. Different threads from different applications may share the GPU cores, interfering on the GPU internal resources and the shared platform resources. Uncertainty may arise in the mapping of threads to cores, and thus the generated interference by an application.

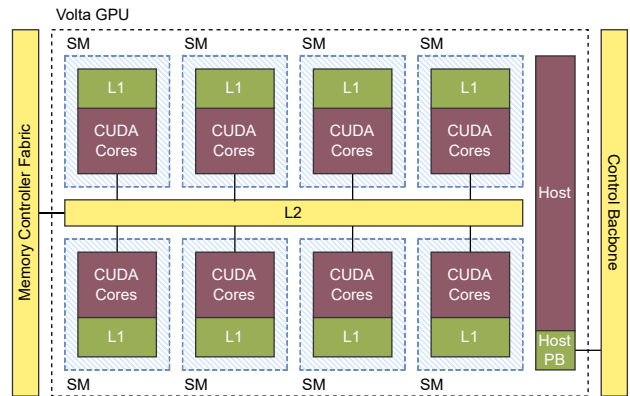


Fig. 11. Simplified PML model for the Volta GPU

However, the exact group of cores where an application is scheduled may not be relevant, provided said group is equivalent to the other groups of core on the platform. Capturing such platform symmetries in the PML models would allow for some level of uncertainty. As illustrated in Figure 11, SM are symmetrical groups of cores on the Volta GPU (Example 2). Each SM has the same number of cores and private resources. Thus a group of threads should exhibit the same behaviour running in isolation in either SM. All SM can access the same shared resources through the same paths on the Volta; the interference suffered and generated by a group of threads is thus independent of the SM where they run. Isolating different applications to separate SM does however rely on undocumented support from the platform [19] (causing issues for Activity 3 in Section V-B).

Example 9 (of category 4). A NVDLA in a "Large" configuration features its own separate microcontroller, depicted in Figure 12, tightly coupled with the accelerator. Where the CPU was in charge in the small configuration of Example 5, the microcontroller drives the accelerator. Modelling the whole as a single accelerator would fail to distinguish transactions originating from the microcontroller and ones originating from the NVDLA functional blocks. Each functional block of the

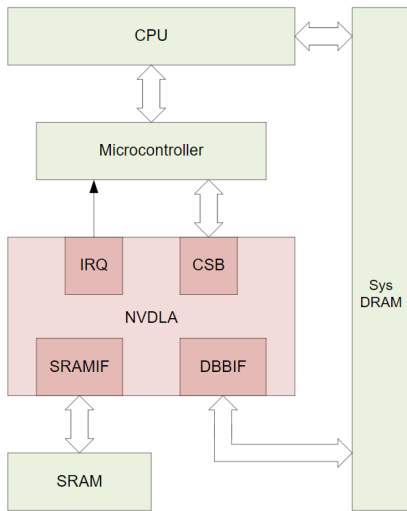


Fig. 12. Integration of the NVIDIA NVDLA in a large configuration [17]

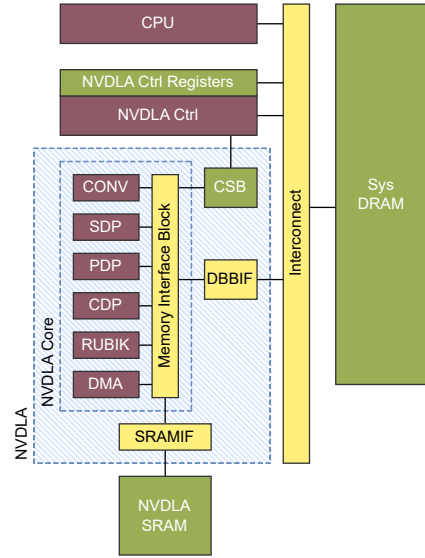


Fig. 13. Simplified PML model for the Large NVDLA

NVDLA can be mapped to its own initiator, as depicted in Figure 13. This abstraction, compared to the one in Example 5, would allow transactions where one or more applications use the different functional blocks without interfering. However each component (CONV, SDP, PDP...) may operate independently and interfere on the DBBIF.

Example 10. The Xilinx ZYNQ-7000 AP [20], outlined in Figure 14, is a FPGA SoC with both Programmable Logic (PL) and Processing System (PS). The PS features a 2-core A9 processor, with a NEON VAU, memory resources, and input-/outputs. The processor offers multiple ports to connect PL devices to resources on the PS. Different ports may reach different or the same resources, through different protocols. Depending on if and how PL devices use said ports, the ports themselves or devices on the PL side may become shared resources and be classified as interference channels.

The PL features three types of ports: 4 general-purpose AXI ports (2 master and 2 slaves), 4 high-performance AXI master ports, and 1 AXI ACP port. The different ports first exhibit functional differences: as master ports cannot be used for the A9 processor to initiate reads from the PL. The AXI ACP port offers a high throughput and limited hardware coherency, as its accesses traverse the processor. However, it may result in serious cache trashing on the processor (as a result of invalidations), and interference on the A9 processor interconnect. The general-purpose ports allow access to most of the SoC interfaces, but share the interconnect with all input/output devices. The high performance ports only support high-throughput accesses from the PL to the main memory.

As a programmable logic device, the model for an FPGA is dependent on the devices and functions that have been configured, and on their use of the available platform resources. As an example, a DMA configured on the PL may solely read memory from the flash controller using a general purpose

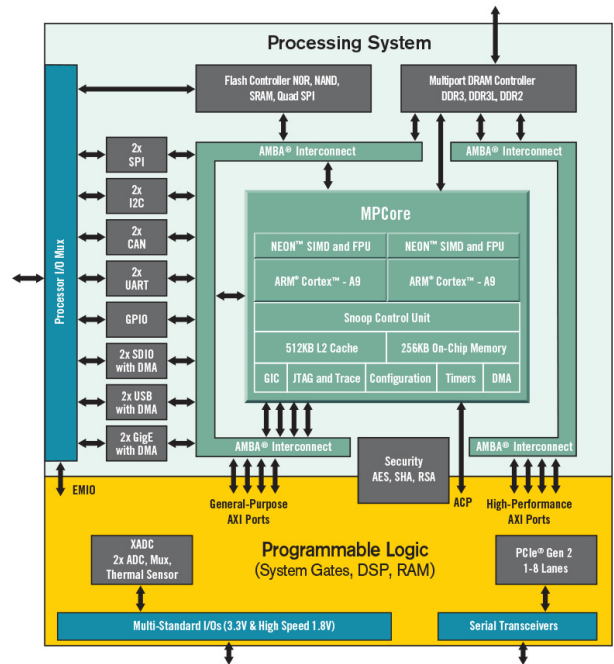


Fig. 14. Overview of the Xilinx ZYNQ-A7000 AP

port. It initiates transactions, contributing to and suffering from interference on shared resources. As such, it should be included as an initiator in the PML platform model. Unless its interference is mitigated, it should further be included as part of the final system configuration during analyses and tests.

Care is thus required upon integrating devices on the PL side. Each configured device should be considered and modelled per the aforementioned cases. The PS can be modelled

as any platform. Existing interfaces to the PL or between the PS and PL should also be considered as part of the model most likely as transporters, based on their use by configured devices.

D. Dimension 3

The third dimension concerns the applicative layers that necessarily come with the accelerator, e.g. a runtime used to offload work from the CPU to an accelerator. They contribute to the interference generated on a platform. As an example the scheduling queue for a device may be shared between different applications, causing delays depending on the scheduler. The transactions generated by an applicative layer also need to be characterised, by assessing their documentation and their use of resources on the platform. The identification and verification must include all software running on accelerators as well as software interfaces or libraries used to program them. Some accelerators may indeed only be addressed through vendor-specific software interfaces.

Example 11. *The definition and execution of kernels, functions running on the Volta GPU, use the CUDA toolkit, or use higher-level libraries and runtimes which themselves offload computation on the GPU through CUDA. CUDA Kernels are written using a superset of a subset of C/C++. That is kernel code supports most of the C language, and the toolkit provides additional syntax for mapping code and data to the GPU, or calling kernels. As such CUDA-enabled code cannot be analysed through existing tools as it may not parse as valid C/C++.*

As part of the CUDA toolkit are the compiler (`nvcc`) and assembler (`ptxas`). The compiler is based on the mature LLVM compiler. The open-source nature of LLVM supports the verification of the generated code, and the development of compiler passes to support further analyses [21]. The assembler, which converts NVIDIA virtual assembly format into an executable binary, is closed. Information relevant for timing or coverage analysis may thus be lost at compilation.

Example 12. *NEON instructions can be exploited through compiler optimisations, intrinsics, or assembly code. Intrinsics are compiler- or vendor-provided functions often used to expose optimisations or vectorisation in languages without such constructs such as C. Compiler optimisations may jeopardize the traceability of the generated binary to the original source [21], and ARM recommends the use of intrinsics over manual assembly code. Intrinsics explicit the use of vectorisation and of the NEON VAU. The added benefit is that the source code only exposes function calls, amenable to analysis.*

Example 13. *The software stack for the NVDLA comprises at its core the User-mode driver (UMD) and the Kernel-mode driver (KMD). The UMD loads a representation of a neural network, maps its inputs and outputs in memory, and informs the KMD that an inference job is ready. The KMD schedules available jobs, allocating DNN layers to function blocks,*

configuring the NVDLA registers, and collecting completed jobs. The KMD (and UMD) can run on the main CPU ("Small" system in Figure 8) or through a dedicated core ("Large" system).

Similarly the open source software stack clearly identifies all required software, and opens the source code for analyses such as coverage or timing. Note that the NVDLA itself does not feature a core which executes user- or vendor-defined software. A NVDLA-enabled platform, depending on the integration, may not fall under the multi-core processor classification. Nonetheless, it still counts as one or more initiators as, once configured through the CSB, each block may initiate transactions to the memory.

Example 14. *The NPU is accessed through an OpenVX Driver. OpenVX [22] is a standard and API which defines reusable computer vision and neural network functions. An OpenVX computation is expressed as a graph. Each node in the graph refers to its parameters and a kernel, the underlying function. The standard defines a number of vision and neural network functions. OpenVX is supported as a backend for numerous neural network runtimes through the Neural Network Runtime middleware [23].*

Nevertheless, the use of such runtimes raises several concerns. The transition from a model (computation graph) to software items is not explicit, and controlled by the runtime itself. This is not in line with the identification of software running on the platform as per AMC20-193. As the NPU supports only a subset of the OpenVX functions, runtimes may further elect to fallback to the CPU to run some software items. Using the NPU through the lower-level OpenVX driver would provide control over software items allocation between cores and the NPU. However, additional characterisation effort is still required to clarify the transactions the NPU might initiate.

VII. RELATED WORK

Worst-Case Execution Time (WCET) analysis methods [24], [25], [26], [27] rely on accurate processor models to produce conservative timing estimates of the execution of applications on a processor. As such, the underlying processor models do often capture a more concrete and precise representation of the processor, e.g. accounting for the internal state of a core. Those are finer-grained models than our transaction-based approach, but validating the underlying models may be a complex process [28]. To the best of our knowledge, PasTiS [6] is one of the few efforts to build a GPU model.

PML takes inspiration from Initiator-Target modelling approaches found as an example in in [29], where paths to shared resources are paramount to the interference analysis. The computation of interfering paths exponentially grows as a function of the number of initiators and targets. To cope with this issue, they propose to introduce reduction criteria (e.g., symmetries).

(Memory) interference analysis approaches fall in two main categories: (1) Request-driven, which is based on a per-(memory) request analysis of an application [30], (2) job-driven, which focuses on the number of (memory) requests

of an application as a whole. Hybrid approaches blend the request-driven and job-driven [30], i.e. considering both approaches jointly in a analysis [31].

Model checking can be used to identify the interference of a platform as done in [32]. To do so, the approach uses formal languages for describing the behaviour of the application and multicore platform and introducing the interference concept and CADP toolbox to evaluate the model.

Interference mitigation techniques are used for minimizing, or even eliminating, the resource contention impact between processing cores. These techniques either make use of space (e.g., cache partitioning, bank partitioning) or time (e.g., scheduling, bandwidth reservation) partitioning to reduce the impact that interference entails. Survey [33] summarizes many of the techniques employed to this end.

VIII. CONCLUSION AND PERSPECTIVES

We discussed the impact hybrid platforms on certification objectives for avionic systems. Hybrid platforms embed several cores and accelerator devices in a small package, to provide high computational power while satisfying strict SWaP constraints. We considered in particular two AMC: AMC20-152A for airborne electronic hardware, and AMC20-193 for multi-core platforms. Both require careful consideration about how devices are used and integrated in the system.

Most accelerators support highly parallel workloads and as such fall into the AMC20-152A *complex* device category, and in scope of the AMC20-193. As such, they require a thorough assessment of their behaviour and their integration in the platform. We thus considered the use of PML to capture and model knowledge about said devices. We identified 3 main dimensions relating to the hardware and software integration of the device in the platform, and proposed a related taxonomy.

We introduced a number of examples of COTS and Soft IP devices to illustrate the proposed taxonomy with PML modelling templates. COTS devices expose little information about their behaviour, and sometimes very limited control on said behaviour. They thus require conservative assumptions and abstractions to comply with certification requirements. Said abstractions have an impact on the performance of the accelerator and they do require backing by the platform configuration, e.g. a single GPU user.

On the other hand, Soft IP (or custom devices), such as the NVDLA, do provide extensive information about their behaviour. They also tend to offer higher configurability than COTS devices. However, they do require separate objectives per AMC20-152A. There might also be a vast amount of implementation and configuration choices to compare to select the most suitable integration w.r.t. to certification and performance objectives.

We did highlight that PML is generic enough to model complex accelerators. However, we also identified venues for improvements. Accelerators such as GPUs cause uncertainty about the allocation of applications (threads) to initiators (cores), and thus the source of transactions. The highly parallel nature of accelerators does also imply a high number of

initiators in the system. This raises concerns about the required granularity of the platform model, the scalability of related analyses, and that of their output.

ACKNOWLEDGEMENT

The work presented in this paper is part of the PHYLOG 2 project supported by the Directorate General of Civil Aviation (DGAC). It is funded by the French government through the France Relance program, based on the funding from the European Union through the NextGenerationEU program.

The work presented in this paper has been funded in part by the Agence Nationale de la Recherche (ANR) under project “ANR-22-CE92-0066-01”.

REFERENCES

- [1] F. Boniol, Y. Bouchebaba, J. Brunel, K. Delmas, T. Loquen, A. Mascarenas Gonzalez, C. Pagetti, T. Polacsek, and N. Sensfelder, “PHYLOG certification methodology: a sane way to embed multi-core processors,” in *10th European Congress on Embedded Real Time Software and Systems (ERTS 2020)*, 2020.
- [2] F. Boniol, J. Brunel, K. Delmas, C. Pagetti, and V. Jegu, “Modelling and analyzing multi-core COTS processors,” in *11th European Congress on Embedded Real Time Software and Systems (ERTS 2022)*, 2022.
- [3] EASA, “AMC (Acceptable Means of Compliance) 20-152A Development Assurance for Airborne Electronic Hardware (AEH),” 2021.
- [4] —, “AMC (Acceptable Means of Compliance) 20-193 on the use of multi-core processors (MCPs),” 2020.
- [5] Z. Jia, M. Maggioni, B. Staiger, and D. P. Scarpazza, “Dissecting the NVIDIA volta GPU architecture via microbenchmarking,” *CoRR*, vol. abs/1804.06826, 2018. [Online]. Available: <http://arxiv.org/abs/1804.06826>
- [6] M. Adalbert, T. Carle, and C. Rochange, “PasTiS: building an NVIDIA Pascal GPU simulator for embedded AI applications,” in *11th European Congress on Embedded Real-Time Systems (ERTS 2022)*, Toulouse, France, Jun. 2022. [Online]. Available: <https://ut3-toulouseinp.hal.science/hal-03684680>
- [7] N. M. Otterness, “Developing Real-Time GPU-Sharing Platforms for Artificial-Intelligence Applications,” Ph.D. dissertation, 2022.
- [8] I. S. Olmedo, N. Capodici, 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, pp. 213–225.
- [9] 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 *ECRTS*, 2018.
- [10] T. Amert, “Enabling Real-Time Certification of Autonomous Driving Applications,” Ph.D. dissertation, 2021, aAI28650154.
- [11] A. Betts and A. Donaldson, “Estimating the wct of gpu-accelerated applications using hybrid analysis,” in *2013 25th Euromicro Conference on Real-Time Systems*, 2013, pp. 193–202.
- [12] R. Pujol, J. Jorba, H. Tabani, L. Kosmidis, E. Mezzetti, J. Abella, and F. Cazorla, “Vector Extensions in COTS Processors to Increase Guaranteed Performance in Real-Time Systems,” *ACM Trans. Embed. Comput. Syst.*, vol. 22, no. 2, jan 2023. [Online]. Available: <https://doi.org/10.1145/3561054>
- [13] I. De Albuquerque Silva, T. Carle, A. Gauffriau, V. Jegu, and C. Pagetti, “A Predictable SIMD Library for GEMM Routines,” in *2024 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2024.
- [14] B. Lisper, “Towards Parallel Programming Models for Predictability,” in *12th International Workshop on Worst-Case Execution Time Analysis*, vol. 23, 2012, pp. 48–58. [Online]. Available: <http://drops.dagstuhl.de/opus/volltexte/2012/3556>
- [15] NVIDIA, *NVIDIA Xavier Series System-on-Chip: Technical Reference Manual*, NVIDIA Corporation, Santa Clara, California, Apr. 2020.
- [16] ARM, *ARM Cortex-A15 Technical Reference Manual*, 2011.
- [17] “NVDLA Primer,” <http://nvdla.org/primer.html>, accessed: 2023-06-28.
- [18] NXP, *i.MX 8M Plus Applications Processor Reference Manual*, 2021.

- [19] J. Bakita and J. H. Anderson, "Hardware Compute Partitioning on NVIDIA GPUs," *2023 IEEE 29th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pp. 54–66, 2023. [Online]. Available: <https://api.semanticscholar.org/CorpusID:259235797>
- [20] S. Ramagond, S. Yellampalli, and C. Kanagasabapathi, "A review and analysis of communication logic between pl and ps in zynq ap soc," in *2017 International Conference On Smart Technologies For Smart Nation (SmartTechCon)*, 2017, pp. 946–951.
- [21] H. Li, I. Puaut, and E. Rohou, "Tracing Flow Information for Tighter WCET Estimation: Application to Vectorization," in *2015 IEEE 21st International Conference on Embedded and Real-Time Computing Systems and Applications*, 2015, pp. 217–226.
- [22] Khronos Group, "The OpenVX Specification v1.3.1," https://registry.khronos.org/OpenVX/specs/1.3.1/html/OpenVX_Specification_1_3_1.html, accessed: 2023-06-28.
- [23] NXP, *i.MX Machine Learning User's Guide*, 2020.
- [24] 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.
- [25] C. Ferdinand and R. Heckmann, "aiT: Worst-case execution time prediction by static program analysis," in *Building the Information Society: IFIP 18th World Computer Congress Topical Sessions 22–27 August 2004 Toulouse, France*. Springer, 2004, pp. 377–383.
- [26] C. Ballabriga, H. Cassé, C. Rochange, and P. Sainrat, "OTAWA: An Open Toolbox for Adaptive WCET Analysis," in *Software Technologies for Embedded and Ubiquitous Systems - 8th IFIP*, ser. Lecture Notes in Computer Science, S. L. Min, R. G. P. IV, P. P. Puschner, and T. Ungerer, Eds., vol. 6399. Springer, 2010, pp. 35–46.
- [27] D. Hardy, B. Rouxel, and I. Puaut, "The Heptane Static Worst-Case Execution Time Estimation Tool," in *17th International Workshop on Worst-Case Execution Time Analysis (WCET 2017)*, vol. 57, 2017, pp. 8:1–8:12. [Online]. Available: <http://drops.dagstuhl.de/opus/volltexte/2017/7303>
- [28] W.-T. Sun, E. Jenn, and H. Cassé, "Validating Static WCET Analysis: A Method and Its Application," in *19th International Workshop on Worst-Case Execution Time Analysis (WCET 2019)*, Jul. 2019, pp. 6:1–6:10. [Online]. Available: <https://hal.science/hal-02924072>
- [29] X. Jean, L. Mutuel, and V. Brindejone, "Assurance methods for COTS multi-cores in avionics," in *2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*, 2016, pp. 1–7.
- [30] H. Kim, D. de Niz, B. Andersson, M. Klein, O. Mutlu, and R. Rajkumar, "Bounding memory interference delay in COTS-based multi-core systems," in *2014 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2014, pp. 145–154.
- [31] M. Hassan and R. Pellizzoni, "Analysis of Memory-Contention in Heterogeneous COTS MPSoCs," in *32nd Euromicro Conference on Real-Time Systems (ECRTS 2020)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), vol. 165, 2020, pp. 23:1–23:24. [Online]. Available: <https://drops-dev.dagstuhl.de/entities/document/10.4230/LIPIcs.ECRTS.2020.23>
- [32] V. A. Nguyen, E. Jenn, W. Serwe, F. Lang, and R. Mateescu, "Using Model Checking to Identify Timing Interferences on Multicore Processors," in *ERTS 2020 - 10th European Congress on Embedded Real Time Software and Systems*, Toulouse, France, Jan. 2020, pp. 1–10. [Online]. Available: <https://inria.hal.science/hal-02462085>
- [33] T. Lugo, S. Lozano, J. Fernández, and J. Carretero, "A Survey of Techniques for Reducing Interference in Real-Time Applications on Multicore Platforms," *IEEE Access*, vol. 10, pp. 21 853–21 882, 2022.

Session Tu.3.C

Hardware Security

Tuesday 11th June

15:00

–

Pastel Room

Onchip Traffic Injection to Counteract Timing Side-Channel Attacks

Francisco Fuentes^{†,‡}, Sergi Alcaide[†], Raimon Casanova[‡], Jaume Abella[†]

[†]Barcelona Supercomputing Center (BSC)
Barcelona, Spain

[‡]Microelectronic and Electronic Systems Department
Universitat Autònoma de Barcelona (UAB), Bellaterra, Spain

Abstract—Security has become a major concern in the last decade, specially with the increment of low-level attack vectors present in COTS MPSoCs. Safety-relevant systems are not an exception, and they are also exposed to security concerns. Side-channel attacks (SCAs) in general, and cache-based SCAs in particular, have gained prominent importance due to the proliferation of cache memories for increased performance. However, there are a plethora of such attacks and effective countermeasures are needed for all of those.

This paper investigates the effectiveness of using hardware traffic injectors to counteract those attacks with the aim of assessing to what extent those can be effective. In particular, we consider the SafeTI, an open source traffic injector developed by us, and assess to what extent attack-specific traffic patterns can defeat Bernstein’s SCA targeting an AES-128 encryption process in a space-relevant platform based on Frontgrade Gaisler’s IPs.

Index Terms—Cyber security, MPSoC, side-channel attack, AES encryption

I. INTRODUCTION

The increasing importance of security in all sorts of computing devices has pushed for the standardization and implementation of secure cryptographic ciphers (e.g., RSA) on modern machines. For instance, the growing RISC-V ecosystem has ratified in the past years two volumes of Instruction Set Architecture (ISA) extensions for the integration of inter-core cryptographic modules [8], [17], providing standardization for high-performance and secure encryption to the RISC-V community. However, these implementations include several components to be treated with security, such as cryptographic keys, intermediate cryptographic operations, etc, that may leak information that an attacker could use for malicious purposes. This is the specific case for side-channel attacks (SCAs), where an attacker without direct access to the desired data, for instance an encryption key, may be able to discover it through indirect methods such as temperature [15], electromagnetic radiation [11], power consumption [10] or timing [4], [14] analysis. Even modern Commercial-Off-The-Shelf (COTS) Multi-Processor System-on-Chip (MPSoC) platforms, using advanced Trusted Platform Modules (TPM2.0) following industry-adopted standard ISO/IEC 11889 [13], are vulnerable to such attacks [18].

This paper aims at assessing to what extent safety-critical platforms are vulnerable to those attacks and whether a programmable traffic injector could be used to counteract those

attacks. In particular, we consider the open source platform SELENE [12], which is based on Frontgrade Gaisler 64-bit NOEL-V processor cores [3] and other Gaisler’s IP, and whose main target is the space domain. We also consider a modified version of Bernstein’s *cache-based timing attack on AES* [4]. Originally, this is a cache-based SCA against Advanced Encryption Standard (AES) [19] symmetric block cipher on a network environment, which we move to occur in an MPSoC (the SELENE platform).

To counteract the attack, we build on the SafeTI traffic injector [21], which we integrate into the SELENE platform. The SafeTI allows programming traffic patterns (i.e., read and write operations with varying parameters) that are injected into the specific interface where the SafeTI is integrated (e.g., the bus connecting the cores to the shared L2 cache). In particular, we focus on the injection of traffic patterns to evict some AES data from the second level (L2) cache of the core performing encryption tasks so that the SafeTI can provide, apart from support for performance validation during MPSoC design [9], security capabilities during operation.

The solution investigated in this paper uses the SafeTI traffic injector for evicting cached data at regular intervals. By enforcing the eviction of a specific AES table from L2 cache periodically, we are able to reduce the amount of information that Bernstein’s attack can discover from a victim in a system without other countermeasures. This particular solution causes a marginal encryption latency increase, around 4% in average at our experimental environment, and since SafeTI is programmable, it can be adapted through software for other applications that may benefit from this solution.

The rest of the paper is organized as follows. Section II provides some background on Bernstein’s attack and existing solutions. Section III introduces the framework used to conduct our case study, which includes the SELENE platform and the SafeTI traffic injector. Section IV provides a summary of the contributions made for this case study, being the tailoring of the Bernstein’s attack for our study environment and SafeTI programming for timing SCA protection. Section V provides result data and explanations on the different SafeTI based protection vectors and Section VI provides a discussion of various subjects related to SafeTI based protection and its applicability. Finally, Section VII provides some final remarks and future work.

II. BACKGROUND AND STATE OF THE ART

Among the attack vectors based on collateral information leakage, timing attacks require special attention due to the high risk level they present on interconnected systems. An attacker with user permissions can perform a timing analysis on a specific task to extract secret information (e.g., cryptographic keys) remotely, without requiring physical access to the target device in comparison to other attacks. In this section, we first introduce the main characteristics of those timing SCAs in Section II-A, and then present the state of the art on protection methods in Section II-B.

A. Side-Channel Timing Attacks

Time based SCAs leverage the dependence between (a) the operation of secret data, where the term ‘secret’ refers to any un-encrypted data, key or information unknown by arbitrary users, with (b) the execution time, or operation latency, of the task using the secret data as an input. Time dependence is a by-product of the operation from two sources; (i) in-processor or accelerator execution of the algorithm (e.g., cryptographic encryption) and (ii) memory access latency. Leakage from both sources can be mitigated by designing time-constant algorithms, basing the logic operations on constant latency instructions with no latency-variant branches, while constraining memory allocation within the same cache level so all data access have an identical time cost. Full compliance with these statements limit the quality of the algorithm (e.g., encryption complexity), reduce the compatibility by targeting specific platform characteristics (e.g., cache capacity, instruction latency), and may hold back performance against using optimal operations (e.g., disable L1 cache). Moreover, some other SCAs would still be possible by, for instance, learning the cache sets or DRAM banks [16], [20] accessed by the protected algorithm. Thus, industry has opted for design policies and certification instead of a single air-tight solution, providing diversity in implementation with low risk of single point failure from a security standpoint.

Statistical analysis of the execution times has proven to be effective regardless of the leakage source. As example, this study uses Bernstein’s attack as a base, whose source code is publicly available [4]. Operations with secrets are identifiable, due to non-constant-time operations and/or non-constant data access latencies, through a classification and correlation of the average encryption timings from two data samples, with a known (attacker) and unknown (victim) cryptographic keys respectively. In particular, the timing measurements are categorized by the value of the plaintext bytes being encrypted in byte segments, which for AES-128 makes a total of 4096 individual metrics (256 possible values for each of the 16 plaintext bytes), taking advantage from the internal byte-wise operation, a common characteristic among cryptographic ciphers. This segmentation allows the reduction of the total number of encryptions, or *sample size*, required by the attack to obtain a clear profile of the timings for each plaintext byte value. Furthermore, the samples obtained by measuring the duration of the cryptographic operation must be from

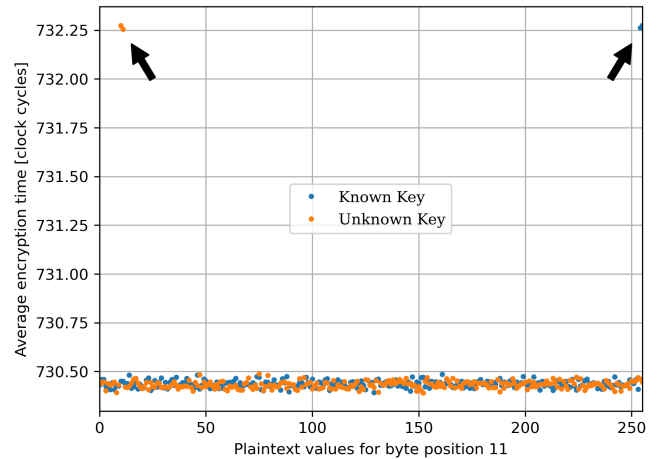


Fig. 1. Average encryption timings, in clock cycles, with plaintext byte position 11 values listed by the x-axis. Data obtained from modified Bernstein’s SCA with a sample size of 2^{27} encryptions for each known and unknown keys. Data values used by the SCA correlation are marked by arrows.

a randomized input, referring to the plaintext in encryption or ciphertext in decryption, avoiding producing timing data with dependencies on the attack itself instead of the target operation.

As a practical example, Figure 1 shows the plaintext encryption timings of a SCA in an unprotected system. The SCA uses a sample size of 2^{27} encryptions for each key, for specifically the byte position 11. In detail, Bernstein’s SCA correlation is a simple observation step, leaving aside standard error calculations, where the most distant timings from the average, around 732.25 clock cycles in the example, are taken as usable data. The idea is that these plaintext values, 10 and 11 for the unknown key and 254 and 255 for the known key marked in the figure, incur in an equivalent systematic latency overhead during the cryptographic execution. Hence, they can be correlated. Since the cipher base operation is the logic *exclusive or* \oplus , the attacker is able of producing a list of candidates for the key to discover by operating with the same operation between the known key and the usable plaintext values.

The actual equation used is $K'_b \oplus P'_b = K_b \oplus P_b$, where K_b is the key byte value, P_b is a plaintext byte value, b the byte position, and the ' apostrophe indicates to be from unknown key byte value or plaintext. Note that the exclusive or neutral element in \oplus is 0 ($x \oplus 0 = x$), therefore, using a zero for the known key simplifies the operation to $K'_b|_{K=zero} = P'_b \oplus P_b$. Hence, all experiments presented in this paper use a zero known key. Following the figure example, the candidates the SCA produces for the unknown key byte position 11 are found as $10 \oplus 254 = 244$, $10 \oplus 255 = 245$, $11 \oplus 254 = 245$ and $11 \oplus 255 = 244$, finding 7 out of 8 bits from the actual unknown key byte position 11, whose value is 244. Note that each byte position has its own timing profile. Therefore, the same operations are made for each byte position.

SCA sample sizes of a low number of encryptions will provide a noisy cloud of timings, with high dispersion and difficult correlation. Thus, for the attack to succeed, it is crucial to work with large sample sizes, so the values with particular higher or lower average encryption times move far away from the average, reducing the number of dots to be correlated. Modifications made to Bernstein’s attack for our study are listed in Section IV-A, while further practical details are explored in Section V-B.

B. State of the Art

Invulnerability against timing SCAs is challenging to achieve since several factors and components interact in non-obvious ways. Solutions to prevent timing SCAs can be categorized into two branches: (i) implementing time-constant cryptographic operations, and (ii) uncorrelating the access latency during operation.

Time-constant operations executed by processors require compliance with design policies for cryptographic security, such as RISC-V ISA extensions [8], [17], which have already been applied in a practical implementation [24]. Accelerator and discrete co-processor solutions, such as TPMs, are also included within this time-constant category, where the security and performance trade-off is apparent due to lacking full time-constant compliance for some products [18]. Software solutions also tend to focus on the operation latency, such as compiler optimizations [23] for avoiding branch prediction and instruction cache attacks that may present non-constant timings, and hence, a side-channel leak. Even if the cipher algorithm was designed to comply with constant-time requirements, fitting within the lowest cache layer, an SCA could still occur by forcing specific cache evictions between the timed cryptographic operations, exposing secret data through cache misses, hence leaking side-channel information. This paper aims at counteracting SCA by focusing on the data access latency.

Uncorrelation of the operating data and access latency, to avoid timing pattern identification of secrets, can be achieved by modifying the replacement policy of the data with custom cache implementations [22], [25], forcing SCA data samples to diverge due to timing diversity, making them uncorrelated, hence, protecting secret data. In detail, the address allocation of the application are encoded with a randomized seed, using the resulting encoding for allocating the data in the cache sets. In order to make the protection effective, it is necessary to ensure the seed is randomized periodically, not to let the attacker learn. Such re-randomization needs to be performed by the Operating System (OS), by the user program or, periodically in an automated manner. One of the key characteristics of these solutions is that the amount of timing data that can be used for correlation purposes by the attacker is proportional to the time a given seed is used for both keys without re-randomization. Therefore, an attack can only be successful if the time elapsed between seed updates is long enough to collect a sufficiently large sample to learn from.

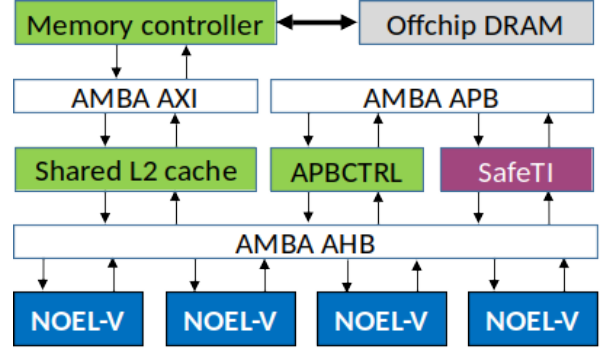


Fig. 2. SELENE platform schematic including SafeTI integration.

In our case, we investigate the effectiveness of a less intrusive hardware solution based on the integration and software programming of the SafeTI traffic injector for protecting against timing SCAs. While the SafeTI is not particularly suited for security purposes, it is a flexible and programmable component that could be leveraged for multiple functions, such as providing protection against multiple attacks, as well as for platform testing.

III. CASE STUDY FRAMEWORK

This section introduces the platform used as research vehicle for our work, as well as the SafeTI traffic injector used to counteract Bernstein’s SCA.

A. SELENE MPSoC Platform

The SELENE platform considered in this work [12] has been released fully integrated and as an open-source platform usable on FPGA [7]. It is based on Frontgrade Gaisler’s technology including its 64-bit NOEL-V processor cores [3], as well as the Advanced Microcontroller Bus Architecture (AMBA) Advanced Peripheral Bus (APB) and Advanced High-performance Bus (AHB) interconnects, and L1 and L2 caches, which are integrated from Gaisler’s GRLIB IP [1].

For this case study, we have integrated a SafeTI module targeting the shared L2 cache level in a SELENE instance with 4 NOEL-V cores (see Figure 2). The 4-way 32 B line 512 KBs L2 cache includes a pseudo-Least Recently Used (pLRU) replacement policy and keeps coherence within the core cluster. The L2 cache is also connected to the off-chip DRAM through an AMBA Advanced eXtensible Interface (AXI) and a memory controller. Each NOEL-V core integrates two individual L1 caches, for instruction (IL1) and data (DL1), of 4 ways and 16 KBs each, implementing a LRU replacement policy, and write-through policy with bus-snooping and an equal cache line size of 32 B to maintain coherence with the L2 cache. Both pLRU and LRU replacement policies offer vulnerabilities in front of SCAs due to their systematic eviction patterns, which attackers can leverage to alter the latency of data allocation during the cryptographic task.

B. SafeTI Traffic Injector

The SafeTI is an open-source hardware component, created in our research group, devised as a flexible, portable and programmable traffic injector [5], developed in VHDL. It is AMBA AHB compatible, and we foresee making it also AMBA AXI compatible in the near future. The SafeTI is programmed through its integrated AMBA APB interface using 32-bit descriptors, which are stored within the internal descriptor buffer, made user-friendly through the public drivers along with the component designs.

Traffic injection is effectively limited by the throughput capable to be generated at the target interconnect. However, if such traffic is injected by software means through processor cores, it is further limited by the transaction size allowed by cores (either a double-word or a cache line) and controlled indirectly by inducing specific hardware behavior with a sequence of software operations. Conversely, the SafeTI can inject precisely any traffic pattern, programmed as a set of descriptors, that the target interconnect accepts, including varying size data requests (e.g., from 1 byte up to 512 MBs), with/without burst mode, read/write, etc., and even introduce specific cycle-accurate delays between traffic injections. Hence, the SafeTI offers the flexibility and controllability needed for our work.

Compared with some previous solutions, SafeTI programmability permits tailoring it for different applications, compatible with virtually any cryptographic algorithm or vulnerable process to timing SCAs under certain conditions (e.g., shared L2 cache access). Integrability is supported by being a standalone module, making it suitable for other platforms as long as it is included with a compatible interface for targeting the desired interconnect. Further discussion is provided in Section VI.

IV. ATTACK CHARACTERISTICS AND COUNTERACTING APPROACH

Our realizations in this paper include the adaptation of Bernstein's attack for a bare-metal execution on the SELENE MPSoC platform, explained in Section IV-A, and the preparation of injection patterns for programming the SafeTI traffic injector to counteract the attack, presented in Section IV-B.

A. Tailoring of Bernstein's Attack

The original implementation [4] allows an attacker to discover part of the AES-128 encryption key stored in another computer server by timing plaintext encryptions with a known key and the unknown victim's key. We modified the attack source code to adapt it for the following evaluation environment: (i) an MPSoC case study on bare-metal without a network connection, meaning that all data and required resources for the encryption are loaded in main system's memory prior to the start of the attack; (ii) AES encryption implemented using the OpenSSL 3.1.2 low-level API [6]; and (iii) programming of the SafeTI injection patterns through calls within the OpenSSL library.

The original source code of the attack is split into individual programs for commodity and presentation, which we packed into a single program and extended to apply an incremental sample size, producing timing data and unknown key candidates every power of 2 encryptions for each key. For instance, whenever we indicate a SCA sample size of 2^{21} encryptions, we refer to a pair of timing data samples, one for each key, of 2^{21} encryptions each. Compiling data following this method allows us to study the evolution of the attack at every step, displaying the SCA sample size needed to retrieve some information about the unknown key (i.e., what key values can be discarded) and how much information is discovered with increasing attack samples, which reduces the size of a brute force attack to explore all remaining combinations (i.e., those that the attack could not disregard).

These modifications have been made to evaluate the protection in the most favorable environment for the attacker. In detail, we reduce the sources of noise that could challenge the effectiveness of the attack by (1) constraining the attack to occur in the cache hierarchy of the cores without any other external interference (e.g., due to peripheral activity) and (2) avoiding the simultaneous execution of any other software within the MPSoC. This includes shrinking the timing to exclusively the encryption operation, data access latencies inclusive, cutting off any algorithm related to networking. Regarding the encryption AES cipher from the OpenSSL library, we selected specifically the 128 bits Electronic CodeBook (ECB) mode due to being the simplest one to attack.

B. SafeTI Injection Patterns

SafeTI countermeasure potential against SCAs has a great dependence on the physical location of the module within the hardware platform, being the best location as close to the processor running the vulnerable operations as possible. However, in this paper we wanted to consider a realistic implementation where the SafeTI does not have access to the DL1 cache, limiting protection capabilities, but instead it has access to the shared L2 cache, expanding protection to any of the processor cores from the MPSoC as Figure 2 shows.

Note that, in order to maximize the protection range, it is required to design an injection pattern tailored for the target environment. For instance, given that the SafeTI is unable to access the DL1 cache, and the DL1 is non-inclusive with the L2 cache in our case study, the data utilized by the vulnerable operation cannot be pre-cached or evicted by the SafeTI at the lowest cache level. Instead, SafeTI protection must be completely based on L2 cache evictions and rely on the vulnerable data to not be fully allocated within the DL1 cache. That way, SafeTI traffic generation may influence the timings of the operation by evicting essential data from the L2 cache, producing eventual L2 misses with increased latency.

Considering the L2 cache implementation characteristics, i.e., pLRU replacement policy and 4-way set associative, it is compulsory, in order to ensure data eviction, to execute at least 4 traffic accesses to each of the cache sets used by the target data (see Figure 3). Therefore, for every targeted

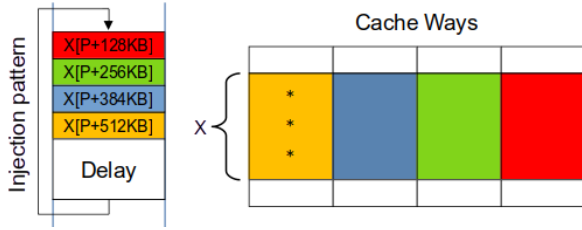


Fig. 3. Traffic generated by SafeTI injection to evict a specific data array, allocated at the cache lines marked with asterisks, from the L2 cache by filling all sets with arbitrary data using a stride of 128 KBs, the L2 cache way size. The arrow out from the end to the start of the injection pattern indicates an execution loop, which is always present in all injection patterns used.

block for eviction, SafeTI injection pattern includes 4 read descriptors with an address stride of 128 KB, the L2 cache way size, targeting the same cache sets as the target data. In detail, if the data desired to evict is allocated starting from address P , the first descriptor is programmed to start the access at address $P+S \cdot 128$ KBs, the second $P+(S+1) \cdot 128$ KBs, the third $P+(S+2) \cdot 128$ KBs, and the fourth $P+(S+3) \cdot 128$ KBs, where S is any integer but -1, -2, -3 and -4 to avoid accessing the eviction target.

In addition, the descriptors include an *access size* field, allowing us to study protection capabilities with varied eviction size with bursty accesses. In Figure 3, the different strided accesses (with starting addresses separated by 128 KBs across accesses) are shown with different colors, namely red, green, blue and yellow. The amount of data fetched by each access is set identical for all accesses matching the amount of cache lines to be evicted from the L2 cache.

The data selected for eviction is data accessed by the vulnerable operation recurrently at a fixed memory address for each execution. This ensures that evicting such data produces a latency overhead generated by the L2 eviction that propagates to the SCA’s timing profile. In this paper, we have used the 4 encryption tables from the AES library as eviction target, from Te_0 to Te_3 , whose size is 1 KB each, being the address of all of them set at compile time. The access of these tables by the ECB cipher depends on both the key and plaintext being encrypted, making several accesses to different segments of the tables for every encryption operation.

Finally, a constant (too frequent) data eviction may be detrimental for the operation latency and/or the protection. Therefore, a *stand-by time*, in clock cycles, is added between target evictions to adjust the eviction rate frequency. To sustain the protection, SafeTI is programmed during initialization and configured in *QUEUE* mode, where the injection pattern is iterated until disabled. In summary, this paper explores a SafeTI L2 eviction based protection against Bernstein’s SCA testing a wide range of injection rate frequencies, varying also the target (a fraction of a table, a full table, or several tables), and considering increasingly large sample sizes for the attack.

TABLE I
NUMBER OF BITS DISCOVERED BY THE SCA
AGAINST ALTERNATIVE CACHE COMPOSITIONS

Enabled caches		SCA sample size (encryptions)				
DL1	L2	2^{21}	2^{22}	2^{23}	2^{24}	2^{25}
✓	✓	2.9	14.3	33.3	70.6	72.2
	✓	0.0	0.0	0.0	0.0	0.0
✓		3.7	14.9	28.1	89.3	92.0
		41.8	64.2	72.6	80.0	80.0

V. EMPIRICAL ASSESSMENT

A. Evaluation Framework

The experiments and data presented in this paper have been produced from software executions on a bare-metal synthesis of the MPSoC SELENE hardware platform on the Xilinx Virtex UltraScale+ VCU118 FPGA-based evaluation kit [26], operating at a frequency of 100 MHz. Software programs executed on the cores have been written in C and compiled with Frontgrade Gaisler AB’s NCC GCC Bare-metal toolchain version 1.0.4 on a Linux system with an O2 optimization level for a RISC-V target. Programs are loaded into memory using the FPGA debug software GRMON3 [2] with the main core set with a specific pointer during the platform booting, matching the program compilation pointer. All secondary cores are left disabled to provide a noise-free environment for the experiments. The program software includes the modified Bernstein’s timing SCA targeting AES-128 ECB cryptographic cipher. Specifically, the attack targets low-level encryption operations of randomized plaintext.

The evaluation presented in this paper is divided per each type of countermeasure tested as follows:

- Disabling platform caches (Section V-B).
- Evicting Te tables by segments in every time interval (Section V-C).
- Evicting a single Te table every time interval (Section V-D).
- Evicting a combination of Te tables every time interval or alternatively in every time interval (Section V-E).

B. Cache Disabling as Countermeasure

Obtaining a base reference of the timing SCA effectiveness on this case study environment is imperative in order to provide a contrasted view of the protection achieved by the SafeTI in the following sections. Therefore, this section presents and reasons about the SCA capabilities in four different environments where DL1 and L2 caches can be enabled or disabled, namely, when both are enabled, when only DL1 is enabled, when only L2 is enabled, and when both are disabled.

Table I shows the equivalent number of bits found by the attacker (out of the total 128 bits of the key) for several sample sizes of the attack in our evaluation platform. These values must be read as follows: if the attacker discovers X bits of the key, it would need a brute force attack exploring 2^{128-X} key values. Note that, in practice, the attacker does not discover specific key bits but discards byte values for different parts

of the key. However, we represent results as the number of key bits that would need to be discovered to match the same cost of a brute force attack to facilitate understanding. For instance, given a key of 16 bits, hence consisting of 2 bytes, if the attack narrows down the value of the key to 37 out of 256 values for one byte and 59 for the other, the subsequent brute force attack would require exploring $37 \times 59 = 2,183$ key combinations (instead of 2^{16}), which we express as having to find 11.1 bits (i.e. $\log_2(2183)$), or equivalently, as having found 4.9 bits.

Starting from the base cache composition of DL1 and L2 caches enabled, the SCA results correspond with the information presented in previous sections. The attacker is capable of discovering more bits of the unknown key by increasing the sample size of the attack, since that way, it is capable of reducing the number of plaintext values highlighted by a higher or lower average encryption times compared to other plaintext value timing averages.

Disabling the DL1 cache but maintaining the L2 cache enabled shows that no information has been discovered by the timing SCA, therefore, we learn 2 things. First, all memory accesses from the AES-128 CBS algorithm have an equal operation latency in this setup. This evidence proves data access latency is the exclusive leakage source from the cryptographic operations being timed, at least on the SELENE platform. Second, not finding timing differences between plaintext values, even when increasing the SCA sample size, indicates all data accessed by the SCA fits within the 512 KBs of the L2 cache.

The complementary case where the DL1 cache is enabled but the L2 cache is disabled denotes slightly more susceptibility to the SCA than the base cache configuration. In this case, DL1 cache misses, instead of hitting in L2 cache, need to access main memory, whose latency is higher than that of L2 cache hits. Hence, those timings that were discrepant in the setup with both caches enabled become even more discrepant when the L2 cache is disabled. Therefore, larger differences provide easier correlation, which in turn provides a smaller list of unknown key candidates, or what is equivalent to, a higher discovery rate.

Last but not least, the disabled DL1 and L2 caches configuration shows to be the most susceptible to the attack from all cache combinations with small samples. Initially, one could expect this case to behave similarly to the prior case with disabled DL1 and enabled L2 caches, given that all data accessed by the timed operations fits within the external DRAM. However, these off-chip components introduce data and access latency dependencies [16], [20], leaking side-channel information whereas the previous case where DL1 is disabled and L2 enabled provides homogeneous latency for all L2 cache hits. In detail, there is an access latency difference when accessing depending on the data accessed due to bank and rank access patterns, generating a data-dependent timing profile due to specific plaintext encryption values requiring extra latency for their accesses than others.

From a countermeasure perspective, the disabled DL1 and enabled L2 cache composition is a strong contender as a

protection solution against Bernstein’s timing attack. However, reducing cache levels results in higher data access latencies, increasing the average encryption time, from the 730 clock cycles of the baseline case (both caches enabled) to 4,763 clock cycles, a considerable 552% overhead. Moreover, it could be argued that a smart attacker may be able to re-enable the attack by evicting parts of the data used by the cryptographic operation to highlight the use of specific data, hence enable correlations in the timing behavior of the cipher algorithm since the accesses latency would depend on the plaintext.

C. Partial Table Eviction with SafeTI

The first countermeasure method using SafeTI for evicting data from the L2 cache consists of evicting AES Te tables, block by block, but only evicting one segment in every time period. For instance, if a table occupies 1 KB of cache space, and it is divided into 4 blocks of 256 B each, SafeTI evicts bytes 0-255 in period P , 256-511 in period $P + 1$, 512-767 in $P + 2$, 768-1023 in $P + 3$, 0-255 again in $P + 4$, and so on and so forth.

Due to SafeTI’s limited descriptor buffer, the injection patterns tested have been constrained to the following 4 different cases: Te0 eviction by 64 B blocks; Te0, Te1 and Te2 eviction by 128 B blocks; all Te tables eviction by 256 B blocks; and all Te tables eviction by 512 B blocks. Between each evicted block, the injection pattern includes a stand-by time in clock cycles, which we refer as *Delay*, that is constant for each experiment so that evictions are homogeneously distributed over time.

Regardless of the Delay or table/s eviction granularity, experimental results show that this protection method is ineffective at counteracting the SCA. The Delay values tested are in the range between 10^3 and 10^5 clock cycles. These values allow full tables to be evicted at the same frequency as the best cases for subsequent experiments where tables are evicted at once instead of block by block. Results show a similar discovery rate of the key by the attack among all 4 protection cases.

Furthermore, the SCA is slightly more successful with this approach than for the base SCA without protection. This negative effect (i.e., the protection helps the attack rather than counteracting it) relates to the fact that the data accessed from the cryptographic operation depends on the plaintext. Hence, evicting single blocks of the Te tables only highlights such plaintext values that access the recently evicted cache lines. Therefore, the attacker learns faster and injection patterns evicting full tables at once are expected to cure this anomaly as analyzed next.

D. Table Eviction with SafeTI

The injection pattern for the experiment in this section is analogous to that of the previous section, but with the evicted block matching the table size of 1 KB. Hence, in every period the target Te table is evicted. Then, SafeTI stands by for Delay clock cycles before looping again.

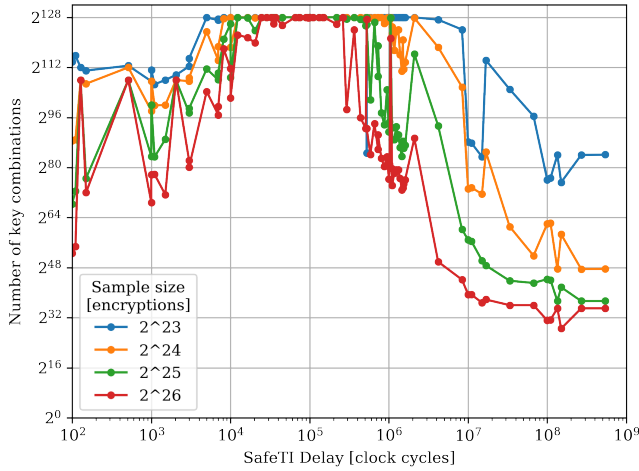


Fig. 4. Remaining key combinations from several SCAs with single table eviction protection, including different Delay values, from 2^{23} to 2^{26} encryption sample sizes.

Figure 4 shows the result of the SCA while a full Te table eviction protection is in place (for Te0 in particular), for different sample sizes for the SCA (between 2^{23} and 2^{26} sample sizes), varying the Delay between full-table evictions. We reach the following observations:

- Results are noisy due to minor modifications in the source code, presenting an intrinsic variability in the execution time measurements and SCA results (shown in Figure 5) as we discuss next.
- There is a (central) range of Delay values for which the SCA is unable to learn anything about the key so that the number of potential key combinations to explore by brute force remains at 2^{128} . However, as the sample size increases, such Delay range narrows down. If the sample is large enough, as we show in later experiments, the range becomes null and the SCA starts learning about the secret key regardless of the Delay value. Still, there is always a particular delay minimizing the amount of information learnt by the SCA.

During our experiments, we noticed that small variations in the code created significant variations in the results for a given Delay and sample size, and concluded that the particular cache alignment of the data has an impact on the results in absolute terms, yet trends hold. This is illustrated in Figure 6, which represents two sets from a 4-way cache, where there is data in static addresses allocated during compile time, and data in dynamic addresses allocated during runtime (e.g., in the stack frame), both marked with *s* and *d* suffixes respectively. The data is ordered from most recently used *A* to least recently used *E*. Focusing on the first case without a filler size (i), the set 0 caches *sA* (spanning across two cache lines), *dC*, *dD* and *dE* data lines, but once a filler size is applied to displace dynamic data by one set at (ii), *dE* is no longer able to fit within the cache, illustrating why some pointer displacements are able to leak more information than others.

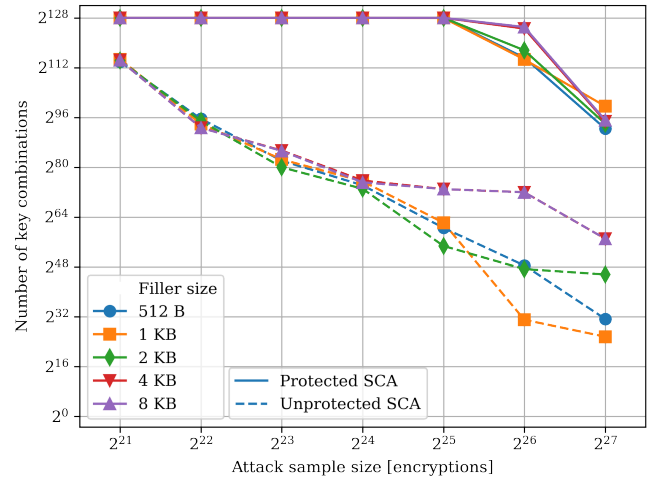


Fig. 5. Remaining key combinations from unprotected and protected SCAs with Te3 table eviction and optimal Delay of 2×10^5 clock cycles for different sample sizes, and varying filler sizes shifting the compile address alignment of useful data.

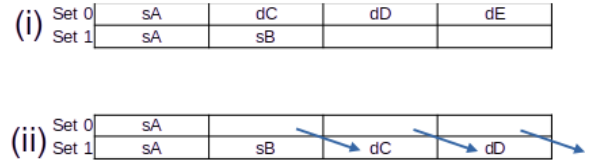


Fig. 6. Associative-set mapping diagram from a 4-way cache in two instances of (i) no filler size and (ii) filler size with one set of displacement. Data is named after being statically *s* or dynamically *d* allocated, from most used *A* to least *E*.

Figure 5 shows both the protected (straight lines) and unprotected attacks (dashed lines) for several compilations of the same program (cipher and attack) but with different filler sizes (between 512 B and 8 KB), which is an unrelated data array used to shift the cryptographic operation pointers for each experiment. The figure shows the diverse results in the unprotected case, ranging between 2^{25} and 2^{27} unknown key candidates, with a sample size of 2^{27} encryptions. In the protected case, variability is drastically decreased, partly because few key combinations are filtered out by the attack. Note that, whenever the filler size is a multiple of the DL1 way size (4 KBs), such as 4 and 8 KBs, results remain the same, confirming the DL1 set-mapping influence over the SCA. In any case, no array for shifting pointers has been used for the remaining experiments in this paper.

As shown before in Figure 4, the degree of protection achieved depends on the Delay value, or eviction period. Such evictions aim at generating arbitrary noise able to remove any correlation that could be used by an attacker. If performed with the right periodicity, evicting a Te table from the L2 cache causes L2 misses, and hence, access latency increases (and so execution time increases) arbitrarily and with enough magnitude to surpass the execution time variability caused by the underlying access patterns that the attacker is trying to

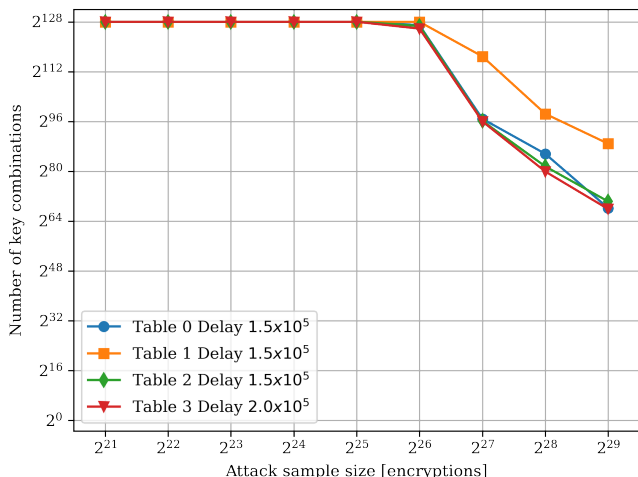


Fig. 7. Remaining key combinations from protected SCAs with Te0, Te1, Te2 or Te3 table eviction and optimal Delay, in order to maximize protection, for each table at different sample sizes.

learn. For instance, a DL1 cache set may contain a Te table line or not depending if it has been recently used. Given that the plaintext encrypted in the recent past has determined, along with the pLRU replacement policy of the DL1 cache, what lines of the Te tables are stored in DL1, Te lines retrieved from L2 are, to some extent, arbitrary. Hence, when those accesses experience higher latencies due to L2 misses caused by SafeTI evictions is, therefore, highly arbitrary. This makes execution times be apparently random because the level of noise introduced is high enough and, apparently, uncorrelated with the key. However, if the eviction period is too small, DL1 misses also miss in L2 highly systematically, which makes overall execution time increase, but noise be low. Similarly, if the eviction period is too high, meaning that evictions only occur seldom, the protection effect SafeTI has on the SCA is very limited.

Finding the optimal Delay for the protection is challenging, due to a dependence with collateral data being evicted from the same sets where the target Te is cached. This makes, in fact, that the optimal Delay varies across Te tables, as shown in Figure 7. Therefore, the only method available to optimize the Delay and choose the value that maximizes the sample size needed by the attacker is through empirical testing. As shown in the figure, the degree of protection achieved across the different tables, even for near-optimal Delay periods, may also vary. For instance, Te1 periodic eviction provides slightly higher protection than that achieved by evicting other tables due to the interactions with other data of the cipher program. Yet, these results also depend on the program pointer shift as shown before in Figure 5.

Overall, the single table eviction protection, once adjusted with the optimal delay, maintains zero side-data leak up to an attack sample size no lower than 2^{25} with an average encryption time of 741 clock cycles, x32 times the attack sample size at the cost of 1% increase in average encryption latency when compared with the unprotected SCA.

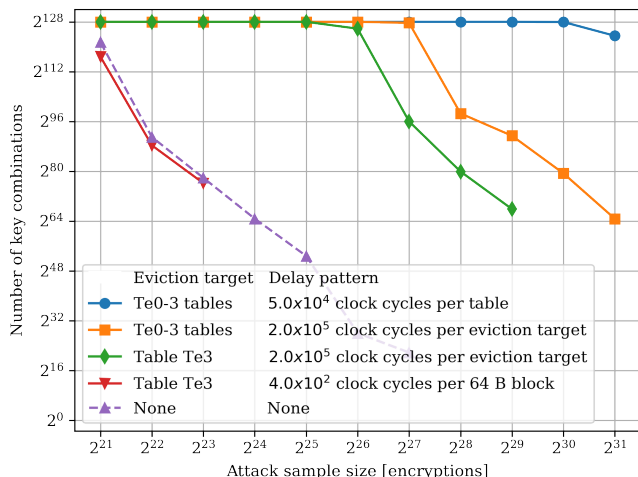


Fig. 8. Remaining key combinations from unprotected and protected SCAs using all 4 methods presented in this paper at different sample sizes.

E. Multiple Table Eviction with SafeTI

This section extends the injection patterns, considering cases where all tables are evicted rather than focusing on one of them. The goal of evicting all tables rather than the very same one systematically is introducing higher entropy, and hence, further challenging the attack.

Figure 8 shows a summary of all SCA cases presented in this paper, focusing on the most favorable setups identified in each case, including scenarios where all tables are evicted. In particular, the configurations evaluated are as follows:

- Unprotected SCA (dashed purple line).
- Te3 eviction by 64 B blocks every 400 clock cycles (red line).
- Full Te3 table eviction every 2×10^5 clock cycles (green line).
- All Te tables evicted simultaneously every 2×10^5 clock cycles (orange line).
- Same as previous one, but instead of evicting all Te tables at once, we evict them in an interleaved manner so that one Te table is evicted every 5×10^4 clock cycles (blue line), which matches the eviction frequency of the previous case where all tables are evicted every 2×10^5 clock cycles.

Focusing on the patterns where we evict all tables, the orange line withstands full protection up to an attack sample size of 2^{27} , with a 4% average latency overhead, whereas the blue line keeps full protection up to an attack of 2^{30} encryptions, with a 12% average latency overhead. Although one could expect the encryption latency overhead to be proportional to the number of L2 misses induced, and thus to the L2 cache evictions performed by the traffic injector, this is not the case between orange and blue lines, which correspond to an equivalent eviction rate, and thus, should cause a similar performance overhead. However, the case of individual table evictions at a higher frequency (blue line) turns out to perform

the evictions clashing with the execution of the encryption function more often, and hence causing higher L2 cache access interference, and increasing encryption latency.

All Delay values present in the figure have been empirically optimized, with the exception of the last case where we evict all tables in an interleaved manner, where the 2^{31} encryptions experiment duration has been 29 hours in our FPGA, making it unreasonable any further increase in the sample size.

VI. DISCUSSION

This section provides light and additional considerations for several topics related to the proposed mechanism, namely, alternative cache compositions in Section VI-A, comparison with related work in Section VI-B, considerations and requirements of the protection in Section VI-C, and protection capacity against alternative SCA sources in Section VI-D.

A. Alternative cache compositions

The case study setup includes a write-through DL1, hence propagating all write operations to the L2 cache. Our eviction patterns evict all data in specific L2 cache sets (i.e. those sets where the target T_e is mapped). Hence, write operations to data in those sets also experience L2 misses due to SafeTI's evictions. If DL1 implemented a write-back policy instead, those other write operations would not be affected by our evictions if they hit in DL1. Hence, the effect of the evictions would be lower and we would expect to need a higher eviction frequency to compensate this effect or, alternatively, being able to plug the SafeTI in a way that it can evict data from DL1 rather than from L2 cache only.

Using a random replacement policy in the L2 cache would challenge to some extent the generation of eviction patterns with the SafeTI to evict full tables, which would only be evicted probabilistically. However, noise introduced would be more random, which would play against the attack.

The SELENE platform used in this work implements 2 levels of cache. Adding further cache levels is expected to be innocuous since all data fits in L2, and hence, it would also do in L3, which would provide analogous behavior to that of the DRAM memory in the current setup.

B. Related work comparison

Being SafeTI a hardware component, the closest solution for a fair comparison would be the caches implementing custom placement policies [22], [25] in order to uncorrelate cryptographic operation input data with its data access latency. These solutions offer a higher protection grade (full protection, indeed) than our protection based on SafeTI, with negligible execution time impact. However, they are intrusive with the original cache components, which would require a new implementation, and verification and validation processes for each affected cache component, hence challenging portability and increasing costs. Our solution, instead, offers a different tradeoff by providing some relevant protection and needing only additional validation of the SafeTI integration, since existing cores and caches remain unaltered.

In terms of resource usage, the SafeTI implementation represents a 1% LUTs and 5% registers with respect the whole platform, or 3.2% LUTs and 12.6% registers with respect to one of the NOEL-V cores, which we consider to be low by supporting all cores.

C. Considerations and requirements of the protection

Eviction patterns caused by the SafeTI are systematic since they repeat specific actions at specific time intervals. This could, theoretically, be leveraged by the attacker to speed up its learning process and decrease the size of the sample needed to retrieve information from the secret key. However, it is unclear how this could be done given that the impact of the evictions caused by the SafeTI vary depending on the plaintext encrypted, as discussed before. Moreover, it would not be difficult extending SafeTI to make Delay across evictions be random while preserving average eviction frequency to further challenge any attack.

Overall, we do not find practical methods where SafeTI-based protection could be defeated other than increasing the attack sample size, or disabling the SafeTI altogether.

All experiments presented show protection capabilities on a SCA targeting AES-128 ECB cryptographic cipher for a specific unknown victim key on encryption operations. In principle, the base of the protection builds exclusively on the SCA profiling dynamic and the encryption timings. Therefore, the protection is agnostic to the key being protected and the calibration may be kept for symmetrical operations such as decryption, offering an equivalent protection level. Tailoring for other vulnerable operations may be achieved through calibration of the inter-eviction Delay time, which in theory makes it capable of tailoring virtually to any vulnerable operation under the following specific requirements:

- The initial address and size of a cryptographic resource, such as the T_e table(s), is required to be known during SafeTI injection pattern programming in order to evict such data during operation.
- Our protection method requires the SafeTI be able to reach a cache memory where the protected process (e.g., the encryption function) performs a relevant number of cache hits since, otherwise, SafeTI evictions would be ineffective.

D. Defense capacity against alternative SCA sources

The focus on this case study has been timing attacks, but other attack vectors exist as mentioned at the introduction, such as power, electromagnetic, temperature analysis, among others. A defining characteristic of SCAs is that, due to being a collateral data analysis, they build on some non-functional metrics from where to infer information about secret keys. We believe that SafeTI patterns can be used in many cases to induce additional activity or alter the activity of the unprotected system in a way that attack vectors other than timing can also be counteracted. Yet, how to tailor SafeTI patterns in each such case is beyond the scope of this work.

VII. CONCLUSIONS AND FUTURE WORK

Security concerns become increasingly significant in safety-relevant platforms. In this paper, we explore the effectiveness of Bernstein's SCA in a space-relevant platform and show how it rapidly discovers encryption key information by exploiting cache latencies. We propose using a programmable traffic injector as a lowly intrusive and adaptable countermeasure and show that it is highly effective and causes very low performance degradation for some configurations, but starts losing efficacy as the sample size of the attack grows. Therefore, we consider this solution is particularly appropriate to be used in conjunction with other defense mechanisms that may take advantage or require the attacker to be staggered in order to provide full protection against timing SCA. This would be the case of, for instance, software solutions whose latency may be substantially higher than that of a hardware mechanism as the one proposed in this work.

The solution proposed in this paper aims at emphasizing the feasibility to use a traffic injector to counteract SCAs, and how it can be easily programmed to challenge the ability of the attacker to learn. However, underlying patterns to be learnt by the attacker still exist and, with a sufficiently large sample, eventually emerge and are learnt. Part of our ongoing research consists of devising approaches to inject traffic with the aim of, rather than adding noise, making emerge fake information so that the attacker is completely fooled and, instead of learning more or less information, it simply learns false information, which would completely defeat the attack.

ACKNOWLEDGEMENT

This work is part of the project (ISOLDE), funded by MICIU/AEI/10.13039/501100011033 and the European Union NextGenerationEU/PRTR under grant PCI2023-143372, and the European Union's Horizon Europe Programme under project KDT Joint Undertaking (JU) under grant agreement No 101112274. This work has also been partially supported by the Spanish Ministry of Science and Innovation under grant PID2019-107255GB-C21 funded by MICIU/AEI/10.13039/501100011033.

REFERENCES

- [1] Frontgrade Gaisler AB. Frontgrade Gaisler AB SoC GRLIB IP library. Retrieved January 16, 2024 from <https://www.gaisler.com/index.php/products/ipcores/soclibrary>.
- [2] Frontgrade Gaisler AB. GRMON3 FPGA debugger software product page. Retrieved January 16, 2024 from <https://www.gaisler.com/index.php/products/debug-tools/grmon3>.
- [3] Frontgrade Gaisler AB. NOEL-V processor webpage. Retrieved November 17, 2023 from <https://www.gaisler.com/index.php/products/processors/noel-v>.
- [4] Daniel J. Bernstein. Cache-timing attacks on aes. 2005. Retrieved November 17, 2024 from <https://cr.yp.to/antiforgery/cachetiming-20050414.pdf>.
- [5] Barcelona Supercomputing Center. Traffic injector SafeTI open IP repository. Retrieved January 18, 2023 from https://gitlab.bsc.es/caos_hw/hdl_ip/bsc_safeti.
- [6] OpenSSL Community. Openssl 3.1.2 library repository. Retrieved November 17, 2023 from <https://github.com/openssl/openssl/releases/tag/openssl-3.1.2>.
- [7] Universitat Politècnica de València. SELENE platform open source repository. Retrieved January 16, 2024 from <https://gitlab.com/selene-riscv-platform/selene-hardware>.
- [8] Ken Dockser, Allen Baum, Barna Ibrahim, Barry Spinney, Ben Marshall, Derek Atkins, Markku-Juhani O. Saarinen, Nicolas Brunie, and Richard Newell. RISC-V Cryptography Extensions Volume II: Vector Instructions. Version v1.0.0, 05 October 2023.
- [9] F. Fuentes et al. SafeTI traffic injector enhancement for effective interference testing in critical real-time systems, 2023. Retrieved November 17, 2023 from <https://doi.org/10.48550/arXiv.2308.11528>.
- [10] Hasindu Gamaarachchi and Harsha Ganegoda. Power analysis based side channel attack, 2018. Retrieved January 10, 2024 from <https://doi.org/10.48550/arXiv.1801.00932>.
- [11] Nilupulee A. Gunathilake, Ahmed Al-Dubai, William J. Buchanan, and Owen Lo. Electromagnetic side-channel attack resilience against present lightweight block cipher. 2021. Retrieved January 10, 2024 from <https://doi.org/10.48550/arXiv.2112.12232>.
- [12] C. Hernández et al. SELENE: Self-monitored dependable platform for high-performance safety-critical systems. In *2020 23rd Euromicro Conference on Digital System Design (DSD)*, pages 370–377, 2020.
- [13] International Standards Organization. *ISO/IEC 11889. Trusted platform module library*, 2015.
- [14] Elmira Karimi, Zhen Hang Jiang, Yunsi Fei, and David Kaeli. A timing side-channel attack on a mobile gpu. In *2018 IEEE 36th International Conference on Computer Design (ICCD)*, pages 67–74, 2018.
- [15] Taehun Kim and Youngjoo Shin. ThermalBleed: A practical thermal side-channel attack. *IEEE Access*, 10:25718–25731, 2022.
- [16] Myoung Jin Lee and Kun Woo Park. A mechanism for dependence of refresh time on data pattern in dram. *IEEE Electron Device Letters*, 31(2):168–170, 2010.
- [17] Ben Marshall, Alexander Zeh, Andy Glew, Barry Spinney, Daniel Page, Derek Atkins, Ken Dockser, Markku-Juhani O. Saarinen, Nathan Menhorn, L Peter Deutsch, Richard Newell, and Claire Wolf. RISC-V Cryptography Extensions Volume I: Scalar & Entropy Source Instructions. Version v1.0.1, ratified on 18'th Feb, 2022.
- [18] Daniel Moghimi, Berk Sunar, Thomas Eisenbarth, and Nadia Heninger. TPM-FAIL: TPM meets timing and lattice attacks. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 2057–2073. USENIX Association, August 2020.
- [19] National Institute of Standards and Technology. *Advanced Encryption Standard (AES)*, May 2023.
- [20] Peter Pessl, Daniel Gruss, Clémentine Maurice, Michael Schwarz, and Stefan Mangard. DRAMA: Exploiting DRAM addressing for Cross-CPU attacks. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 565–581, Austin, TX, August 2016. USENIX Association.
- [21] Oriol Sala, Sergi Alcaide, Guillem Cabo, Francisco Bas, Ruben Lorenzo, Pedro Benedicte, David Trilla, Guillermo Gil, Fabio Mazzocchi, and Jaume Abella. Safeti: a hardware traffic injector for mpsoac functional and timing validation. In *2021 IEEE 27th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, pages 1–7, 2021.
- [22] David Trilla Rodríguez, Carles Hernández Luz, Jaume Abella Ferrer, and Francisco Javier Cazorla Almeida. *Cache side-channel attacks and time-predictability in high-performance critical real-time systems*. Association for Computing Machinery (ACM), Jun 2018.
- [23] Jeroen Van Cleemput, Bjorn De Sutter, and Koen De Bosschere. Adaptive compiler strategies for mitigating timing side channel attacks. *IEEE Transactions on Dependable and Secure Computing*, 17(1):35–49, 2020.
- [24] Aleksander Waage. Secure implementation of a RISC-V AES accelerator, 2022. Retrieved January 10, 2024 from <https://hdl.handle.net/11250/3023096>.
- [25] Zhenghong Wang and Ruby B. Lee. New cache designs for thwarting software cache-based side channel attacks. *SIGARCH Comput. Archit. News*, 35(2):494–505, jun 2007.
- [26] Xilinx Virtex UltraScale+ FPGA. VCU117 evaluation kit. Retrieved January 16, 2024 from <https://www.xilinx.com/products/boards-and-kits/vcu118.html>.

Approach for High-Performance Random Number Generators for Critical Systems

Pascal Hammer, Veronika Krause, Tobias Probst, Jürgen Mottok

Laboratory for Safe and Secure Systems (LaS³)

Technical University of Applied Sciences Regensburg

93053 Regensburg, Germany

{pascal.hammer, veronika1.krause, tobias.probst, juergen.mottok}@oth-regensburg.de

Abstract—In times of digitalization, the encryption and signing of sensitive data is becoming increasingly important. These cryptographic processes require large quantities of high-quality random numbers. Which is why a high-performance random number generator (RNG) is to be developed. For this purpose, existing concepts of RNGs and application standards are first analyzed. The proposed approach is to design a physical true random number generator (PTRNG) with a high output of random numbers. Based on this, the development begins with the analog part of the RNG, the noise signal source and a suitable amplifier for the analog noise signal. Therefore, a special noise diode from Noisecom and an amplifier from NXP were chosen and analyzed in different measurements. From the results of the measurements, it can be concluded that both components are suitable for use in the RNG.

Keywords—RNG, Random Number Generation, Noise Source, Random Processes, Cryptography, Random Sequences

I. INTRODUCTION

Due to the increasing demand of cryptography in communication and other domains, more attention falls to random numbers, which feature high entropy and are evenly distributed, and their generation. One distinguishes between true random processes like thermal noise, quantum mechanical effects or atomic decay processes on one hand and pseudo random numbers which seem genuine but are generated by a deterministic process on the other. True random numbers have a higher quality than pseudo random numbers, and are therefore mandatory for the proper function of many cryptographic processes. In cryptography, random numbers are used, for example, to generate keys for cryptographic procedures or non-deterministic padding. For the correct and secure functionality of these applications, it should not be possible to guess the random numbers or parts of them.

In enterprise environments on server, where many connections are established in a short time, exists a high demand for cryptographic keys. In fact, there are random numbers required in high frequency to seed the key generation. This point is in contrast with the fact that physical true random number generators (PTRNG) require more time to generate random numbers than deterministic random number generators (DRNG), which means they are too slow to meet the requirements of the cryptographic components. PTRNGs however generate random numbers with a higher rate, but have the disadvantage of being deterministic.

This is the main reason why the focus of this research is on the approach of developing a high-performance random number generator (RNG). The first idea of the approach is to evaluate whether a PTRNG can be realized with a suitable performance for these applications. The aim is to get as much performance as possible out of a PTRNG, and then combine

it with a DRNG to cover applications that require even more throughput. The result is a hybrid RNG with a higher performance than a PTRNG and better random numbers than a DRNG. Regardless of the approach, determining the maximum achievable performance of a PTRNG is a suitable first step. Furthermore, the economic viability of the developed solutions has to be considered. This also means evaluating the use of cheaper or off-the-shelf components for the RNG. Irrespective thereof, the different RNGs should be a tradeoff between price and quality.

The Federal Office for Information Security (BSI) is the central authority for IT security in Germany. The objective of the BSI is to preventively promote cybersecurity to enable and support the secure use of information and communication technology in society. The BSI provides support to ensure the issue of IT security, and minimum standards and guidelines are developed and published to support users in avoiding risks or strengthening their systems. Regarding RNGs, there is the technical guideline TR-02102 [2] which contains recommendations for the key length in cryptographic systems but also includes information about the use of RNGs. For certification of RNGs in Germany, the AIS 20 (for deterministic RNGs) and the AIS 31 (for physical true RNGs) are mandatory [3]. These two application notes define the different classes of random number generators, PTG.1 to PTG.3 and DRG.1 to DRG.4, and their mathematical background. An overview of the RNG classes is depicted in Fig. 1. The nomenclature in this paper is based on the naming convention in the AIS20/31.

This paper aims to answer the following research questions:

- How can a high-speed noise signal source be realized?
- How and from which properties of the noise signal can conclusions be drawn about which properties of the raw random numbers (before post-processing)?

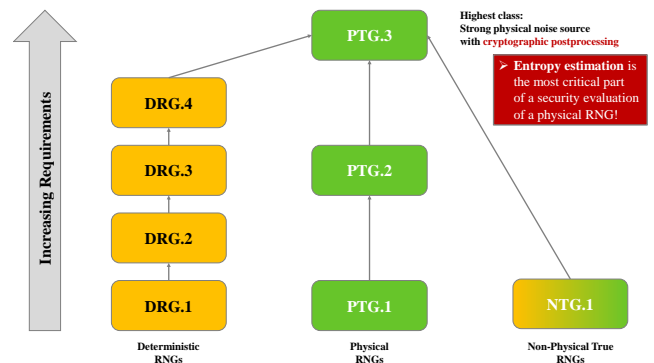


Fig. 1. Overview of the RNG classes [1]

- How must the analog noise signal be amplified so that good digitization is possible?

The paper is structured as follows: Section II shows the current state of the art regarding the RNGs, especially the application notes of the BSI. Chapter III provides a review of existing RNG concepts. Chapter IV gives an overview of the proposed approach of the research work. In Section V, the intermediate results are described. And Chapter VI concludes the paper and gives an outlook to the future work.

II. CURRENT STATE OF THE ART

Random number generators are divided into three types:

- The physical true RNGs (PTRNGs), which are based on physical phenomena like thermal noise or atomic decay processes, for example.
- The deterministic RNGs (DRNGs), which generate pseudo random bit sequences out of one initial random value called seed.
- Non-physical and non-deterministic RNGs, which are used if there is no certain cryptographic hardware available (beyond scope of this paper).

Each type consists of different classes, higher numbers indicate that an RNG provides better security capabilities, while the requirements to be met by the RNGs are consequently also increasing. According to the AIS20/31 workshop in June 2023, the classes DRG.1 and PTG.1 have been dropped because they are no longer sufficient for the required security features [1].

A. Physical true RNGs (PTRNGs)

One type is the physical true RNG based on specific hardware to generate true randomness, which means generating unpredictable random numbers. Calculating previous or subsequent random numbers based on known sequences and the physical environmental conditions at the time of generation must not be possible. The generation of random numbers is based on the unpredictable behavior of electronic circuits, like thermal noise etc. Reducing bias or dependencies between the random numbers can be achieved by a deterministic post-processing of the noise raw data (the digitized noise signals). But post-processing can also have different objectives like statistical inconspicuousness or entropy extraction, for example increasing the entropy per bit. [3]

A common deficit of PTRNGs is the slowness compared to other RNGs due to the fact that the generation of random numbers is more time-consuming due to the amplification and digitization of the raw analog noise signal. The bottleneck is therefore the more complex processing of the analog signals. Changes of the environmental conditions like temperature, electromagnetic fields etc. may impact the the generated random numbers. PTRNGs are also more difficult to evaluate in comparison to DRNGs because, due to the lack of standards of how an RNG should be set up, they can take on many different forms and utilize various physical phenomena as an underlying technology. [3]

The technical guideline TR-02102 recommends using a generator according to PTG.3, if a physical true RNG is required in an application. The recommendation applies in particular to generating keys for calculating signatures and to Diffie-Hellman based key exchange. For some applications, PTG.2 generators are sufficient, e.g. for the production of

keys for symmetric encryption or seed generation for a deterministic RNG of class DRG.3 or DRG.4. Random numbers produced by PTG.2 RNGs feature high entropy but do not foreclose statistical dependencies. PTG.2 generator can be appropriate if it can be proven, that the potential advantage to an attacker caused by these dependencies is difficult to exploit. But nevertheless, it is not recommended to use a PTG.2 RNG directly. An RNG of class PTG.2 can be upgraded to a class PTG.3 generator using cryptographic post-processing, which is usually implemented as a software component. The following example of post-processing, shown in Fig. 2, is based on the Davies-Meyer compression function [4]. The raw random numbers from the RNG are divided into 128 bit blocks M_i where each block is XORed with 128 bit values (z_1 and z_2) from the digitized noise source and AES-128 encrypted afterwards. The results of the AES-128 encryption and the initial 16 byte block are XORed again to produce the final block of secure random numbers. The purpose of the post-processing is to increase the entropy of the random numbers and to eliminate statistic anomalies [5].

RNGs of classes PTG.2 and PTG.3 must comply with the following properties [2]:

- It is possible to describe the statistical properties of the random numbers with a stochastic model, capable of reliably entropy estimating.
- The average increase of entropy per random bit is above a defined minimum (near 1).
- Statistical weaknesses or deterioration must be detected within a reasonable time through statistical tests during operation.
- A total breakdown of the noise source or an unacceptable change of the random numbers must be detected immediately. In this situation, an alarm signal must be triggered. The generation of random numbers must be ceased after a breakdown occurs.
- This property is only relevant for PTG.3 generators: A strong cryptographic post-processing ensures a security level of a DRG.3 generator despite total breakdown of the noise source.

B. Deterministic RNGs (DRNGs)

The following section describes the second type of RNGs, the deterministic generators. DRNGs extend short random sequences, handed over as seed from an entropy source, to very long random bit sequences in a deterministic way. Although the bit sequences look random, the total entropy can never be larger than that of the seed. Depending on the generator, the seed can be renewed during its service life. [3]

The DRNGs have the advantage over PTRNGs to be less difficult to evaluate because the computational security can be evaluated independently of its implementation and there are also some approved standard DRNG mechanism. This is not possible for PTRNGs, where the same design may behave completely different with different hardware. [3]

The inner state of the generator is initialized with the seed value. Within every step, the inner state of the generator is updated, the random numbers are derived from this state and the values are issued as bit sequence with fixed length. Hybrid deterministic RNGs update their inner state in a process called reseed or seed update with true random values. This process can be cyclic or triggered by the application. The inner state of the RNG must always be protected against

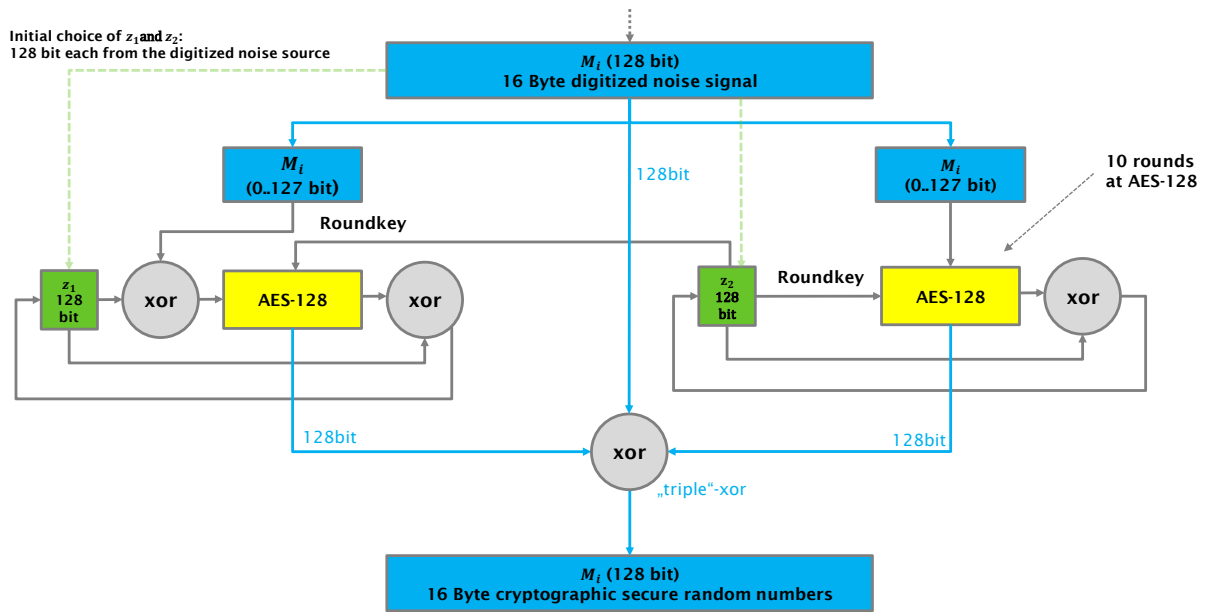


Fig. 2. Cryptographic post-processing based on the Davies-Meyer compression function [4]

access and manipulation, especially during reseed. If a deterministic RNG is required in an application, it is recommended to use a generator of class DRG.3 or DRG.4. For the DRG.3 class, a regular inflow of new entropy is required. However, this property is not sufficient to fulfill class DRG.4. For conformity with DRG.3 respectively DRG.4, the RNG must comply with the following requirements [2]:

- It is not possible for an attacker to calculate the predecessor or successor for a known subsequence of random numbers or to estimate one of them with a higher probability as without knowing the partial sequence.
- For an attacker with knowledge of the inner state, it is not possible to calculate previously issued random numbers or to estimate the numbers with higher probability as without knowing the inner state.
- If the RNG should be upgraded to DRG.4, there is another requirement to be met. Even with knowing the inner state of the generator, the attacker should not be able to calculate the random numbers which are generated after the next reseed/seed update or estimate them with higher probability as without knowing the state. [2]

Using inappropriate RNGs should be avoided because it can weaken strong cryptographic mechanisms. The most important property of the generators is the unpredictability and secrecy of the inner state at all times. For a good quality of the random numbers, they should be evenly distributed on $\{0, 1\}^n$. To achieve this, the individual bits of the random sequences must be independent of each other and the history. [2]

Basically, PTG.3 and DRG.4 generators have greater resistance against side channel attacks in comparison to PTG.2 and DRG.3. In side channel attacks, the principle is to

observe the RNG and find correlations between observed data and generated random numbers. Characteristic information can be obtained, for example, by analyzing the runtime or energy consumption of the RNG. Attacks interfere with the device and provoke errors during execution. Due to the steady inflow of new entropy at PTG.3 and DRG.4 RNGs, side channel attacks regarding the cryptographic post-processing become more difficult. The attacker is not able to combine information about the inner state at consecutive points of time. Besides side channel attacks, there is a higher risk of long-term compromise of RNGs of class DRG.3 compared to DRG.4 and PTG.3 when the RNGs generate long random sequences from one single seed value. [2]

RNG class	Properties
DRG.3	calculation of predecessor or successor of known subsequence not possible, not even with knowledge of the inner state
DRG.4	calculation of predecessor or successor of known subsequence not possible, not even with knowledge of the inner state, calculation of random numbers after reseed not possible
PTG.2	statistical model to estimate entropy, statistical tests during operation, detection of breakdown and alarm signal, automatic deactivation of noise source
PTG.3	statistical model to estimate entropy, statistical tests during operation, detection of breakdown and alarm signal, automatic deactivation of noise source, cryptographic post-processing

C. Non-physical and non-deterministic RNGs

The third type of RNGs are the non-physical and non-deterministic generators (NTG). These are used in particular for cryptographic applications when neither a deterministic nor a physical RNG is available, as these applications are generally run on computers without certified cryptographic hardware. Typically, entropy is gained from system data (timing values, random access memory (RAM) data, etc.) or user's interaction (mouse movement, keystrokes, etc.). NTGs are beyond scope of this research because they are completely based on deterministic random numbers and therefore not suitable for cryptography. [2]

D. Hybrid RNGs

An RNG is called hybrid DRNG if it accepts additional input or if it is able to trigger a seeding/reseeding procedure. Hybrid RNGs use design elements from both DRNGs and PTRNGs. The combination aims to increase the computational complexity of the output sequence and also to increase the entropy per bit. A cryptographic post-processing applies additional security to the RNG in case the entropy per bit is smaller than assumed.

The security of a hybrid deterministic RNG of class DRG.4 is based on the complexity of the deterministic part of the RNG. Backward secrecy and forward secrecy should be ensured by the algorithmic properties of the DRNG alone and without relying on any entropy in the additional input data. Backward secrecy is the assurance that previous random numbers cannot be determined from the knowledge of current or subsequent random numbers, whereas forward secrecy means, it is not possible to determine subsequent random numbers from current or previous random numbers.

Originally, the functionality classes DRG.2 and DRG.3 were designed for pure DRNGs, but the AIS20/31 also covers hybrid DRNG designs. The functionality class DRG.4 defines requirements for all DRNGs, but these can only be fulfilled by hybrid DRNGs. Hybrid random number generators of class PTG.3 utilize a strong noise source and powerful cryptographic post-processing. [2] [3]

E. Test suites

The quality of random number generators can be determined with the help of statistical test suites. The following gives an overview of the NIST- [6] and Dieharder-Suite [7]. The NIST provides a test suite, which consists of 15 statistical tests, freely available on their website [6]. The suite was developed to test the randomness of arbitrarily long binary sequences produced by any type of RNG. Thereby, the tests focus on different types of non-randomness that could exist in a sequence. [8]

The Dieharder random number generator test suite is an open-source project developed and maintained by Robert G. Brown. This suite is the expansion and optimization of the original Diehard test suite introduced by George Marsaglia in 1995 [9]. It also includes tests of the NIST test suite and a variety of tests contributed by users, introduced by the Dieharder contributors or implemented from descriptions in literature. The test suite aims to provide a universal set of tests for random numbers. [10]

However, both test suites cannot definitively determine whether an RNG deliver true random numbers, they can only

detect statistical correlations between the generated random numbers and mark the generators as weak in this case. [10]

F. General PTRNG structure and basic parts

In most cases, PTRNGs are designed and afterward evaluated for their security by independent institutions or companies. As only a limited number of laboratories are approved for certification, it is important and simplifies the process when the PTRNG designer and the certification institution use the same vocabulary and definitions.

Therefore, this section provides an overview of the general structure of a PTRNG and the main components that must be included. The main function of the PTRNG is to produce a series of unpredictable bits or binary numbers. The PTRNG is based on an unpredictable physical phenomenon, the output of which must be converted into a series of bits or numbers.

Since the majority of PTRNGs are based on analog physical effects, a component that performs the analog to digital conversion is an essential component of the PTRNG. For this reason, the following four basis blocks are required for PTRNGs [11]:

- Source(s) of randomness
- An analog-to-digital converter (ADC)
- A post-processor
- Embedded tests

An overview over the general structure is depicted in Fig. 3. The PTRNG usually contains one or more sources of randomness, each generating an analog signal. These analog signals are converted into a stream of bits with the help of an analog-to-digital converter. The ADC outputs a stream of random numbers in bits or multi-bit values, which may still be of poor statistical quality at this point. If necessary, this low statistical quality can be improved by an algorithmic post-processor to obtain a high-quality digital noise.

Using embedded tests according to predefined testing procedures, the quality of the generated random numbers is continuously monitored during the operation of the PTRNG. At least two tests should be carried out: one initial test at startup of the RNG for correct operation, and one function for continuous monitoring. [11]

This section described the different types of RNGs and also how the classification works for RNGs. PTRNGs have some advantages in comparison to DRNGs, but are therefore more complex. Hybrid RNGs combine the benefits of both types and thus offer a good intermediate solution. All RNGs can be tested with the help of statistic test suites. After this overview, the next section gives some examples of existing RNGs.

III. RELATED WORK

This section describes and analyses existing RNG concepts and thus creates a basis for comparison for the newly developed RNG.

The first RNG to be analyzed is the Quantis QRNG engineered by the company ID Quantique [12]. This generator is a physical RNG based on a quantum optics process with a maximum rate of random data of 4 Mbit/s. The device functions by emitting photons one by one towards a semi-transparent mirror and their reflection or transmission events are detected and associated with the bit values 0 and 1. In comparison to other noise sources, quantum RNGs are less vulnerable to environmental perturbations, as the underlying

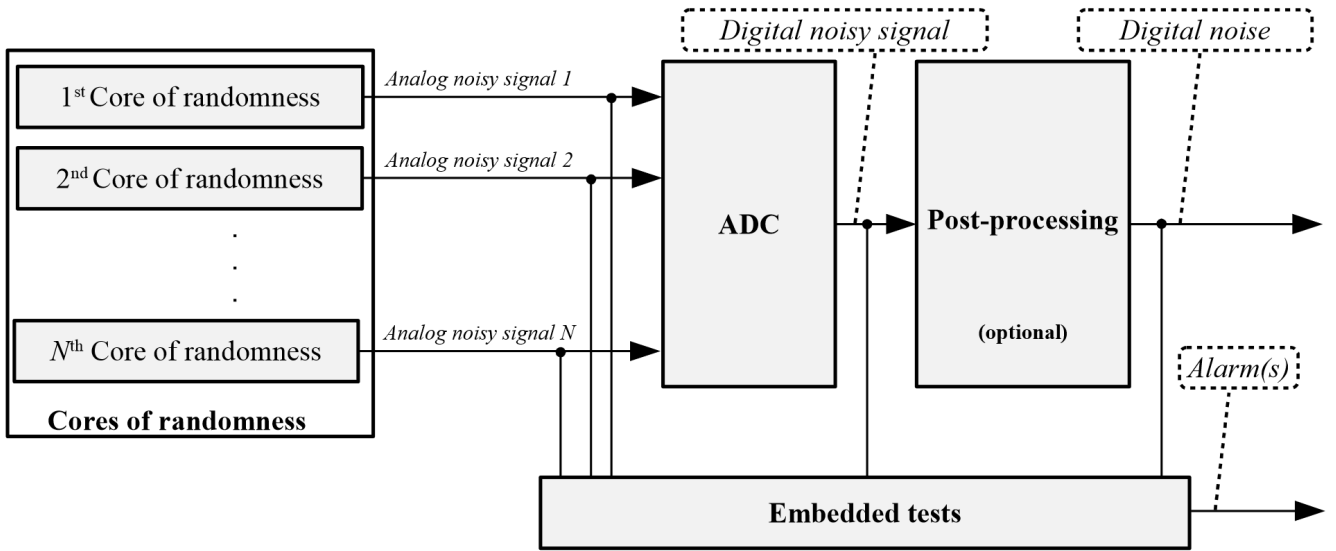


Fig. 3. General structure of PTRNG [11]

processes, for example light, have no dependencies to temperature or electromagnetic fields. On the other hand, as a manipulation attempt, laser light or the sending of photons onto just one of the detectors could compromise the produced noise signal. To avoid this, the hardware of the generator and the random numbers are continuously monitored and if a failure is detected, the random bit stream is immediately disabled.

The next RNG is the PRG310 developed by the IBB engineering office Bergmann [13]. This RNG is based on Z-diodes as thermal noise source for a continuous generation of random numbers for the classes PTG.2 and PTG.3 with a continuous random number bit rate of 300 kBit/s. A permanent monitoring of statistical properties of the digitized noise data verifies the quality of the random numbers.

The PRG700, also developed by the IBB engineering office Bergmann [14], is the third object of comparison. This RNG is designed on a small printed circuit board (PCB) for integration in other applications. It also uses thermal noise as an entropy source and continuously generates random numbers. But the RNG is not as fast as the PRG310, therefore it is only suitable for low bandwidth applications, because the maximum data rate for random numbers is 40 kBit/s for this RNG.

The last tested RNG is a smart card of the company Atos, which is classified as PTG.3 and includes cryptographic post-processing [15]. The smart card is based on the SLE78 security crypto controller developed by the semiconductor manufacturer Infineon [16]. This controller is recommended for applications like ID cards, passports or electronic signatures. For the smartcard, no data rate is specified for the output of the random numbers, but it is to be expected that it can output significantly less random data than the RNGs described above because typical smartcard applications only sporadically require random numbers for cryptographic applications.

The aim of this research approach is to achieve better performance than the RNGs described above. Performance describes in this case the maximum possible data rate in bit per second for random data.

IV. PROPOSED APPROACH

The subsequent section outlines the initial step in the development approach for a high-performance Random Number Generator (RNG). This includes the noise signal source and the amplification of the analog signal. A short preview of the further steps of the development process are presented at the end of the section, but they are part of future work.

Initially, this work focuses on the development of a PTRNG to assess its maximum performance as a standalone solution without integrating it into a hybridized approach. Performance is in this context defined as the number of generated random bits per second. In addition to performance, the quality of the generated random bits, the resilience against external interferences and the reproducible implementation has to be taken into consideration either. If this is not satisfactory, a hybrid RNG is to be developed as a second variant. The aim is to evaluate the impact of the hardware components on the quality of the generated random numbers. Atomic decay processes are not used as noise source in this research as these processes are too slow and radiation sources in RNGs are unsafe because, in this case, the RNG would have to be shielded against so that no radiation reaches the outside.

The initial step of the development process is the selection and analysis of adequate noise signal sources for a TRNG. During our research, the choice fell on the Noisecom NC302BL diode [17] which is to be investigated as a potential noise source for a first approach. The Noisecom diodes are suitable for broadband noise generation because they are optimized for this purpose. Theoretically, all Noisecom diodes have these properties, but to ensure the ideal performance, the best ones are hand-picked for performance characteristics from all those produced. According to the datasheet, the diodes deliver symmetrical white Gaussian noise and flat output power across the frequency band from 10 Hz to 3 GHz. In order to perform tests on the Noisecom diode, a PCB with the reference design as noise source according to the datasheet was developed. This PCB facilitates the investigation of the frequency spectrum of the noise source. The frequency response should be nearly horizontal because

the power should be as independent of the frequency as possible. [17]

The analog signal of the noise signal source only provides low output power. That is why a suitable amplifier is needed to amplify the signal for its digitization, otherwise the amplitude of the signal is too low for the hysteresis of the digitizing circuit. A relevant feature for the selection of the amplifier is the possible bandwidth. The amplifier is tested in combination with the noise source. Since the gain of the amplifier also has a certain dependence on the frequency, this characteristic can be used to improve the frequency response of the noise source. This means that the combination of noise source and amplifier offers a more horizontal frequency response than the individual components.

According to the required features, the NXP BGA2818 was selected as a suitable amplifier for the RNG. It is a wideband amplifier for frequencies up to 2 GHz with a maximum gain of +30 dB. The BGA2818 delivers a nearly constant gain over the complete frequency range. For the initial testing of the amplifier itself and in combination with the noise signal source, both parts are designed according to their reference circuits on individual PCBs. This simplifies the test process and offers the opportunity to use different components if one of them does not fulfil the expectations. The overview of these building blocks is depicted in Fig. 4. This figure also includes the digitization of the noise signal and the microcontroller, which are part of future work. After all components are tested on their individual PCBs, the complete circuit will be united on one PCB.

There are two options for the digitization of the amplified analog signal. Either using a comparator or with the aid of an analog to digital converter (ADC). The ADC needs to be fast enough so that the analog signal is sampled correctly.

The digitized noise signal is processed with the help of a microcontroller. The controller collects bits from the ADC and prepares the bit sequences for the cryptographic post-processing. This is the last step before the random numbers are ready for use in an application, which is done using a software component on the microcontroller. Via a defined interface, the random numbers are provided to the application demanding them. During the whole process, the controller needs to monitor the random bits in the event that errors occur in the process, such as the failure of the noise signal source or a deterioration in the quality of the random data due to external influences like temperature changes or electromagnetic fields.

The proposed approach is described in this section, starting with the analog part of the RNG, the noise signal source and the amplification of the analog signal. During the research, a special diode was chosen as noise signal source according to the characteristics described in the datasheet. To provide a analog signal for digitization, the analog signal of the noise sources needs to be amplified. Therefore, a suitable amplifier was also chosen. The intermediate results with both components are presented in the following section.

V. INTERMEDIATE RESULTS

The following section describes the intermediate results achieved so far. Starting with the noise signal source and the amplifier up to the combination of both components.

A. Noise signal source (*Noisecom NC302BL*)

For the first approach, the Noisecom NC302BL is chosen as noise source to be evaluated because it is developed for this use case. To ensure a comparable test environment, a PCB with the reference circuit mentioned in the datasheet, is created for the Noisecom diode. In addition, the output signal is routed via a SMA connector to ensure a better connection to the measuring device. The operating point of the diode, that is defined by the current flowing through it, is adjustable via a potentiometer.

With the help of this setup, the analog noise signal of this diode could be measured with an oscilloscope depicted in Fig. 5. The figure shows a section of the noise signal produced by the diode with a voltage level of 2 mV peak-to-peak. This voltage level is too low for direct digitization, which means an amplification is required to digitize the signal properly. No direct statement can be made about the quality of the signal on the basis of the chronological sequence. For this, the frequency spectrum of the signal must be analyzed.

The power density spectrum was also measured up to a frequency of 3 GHz with the PCB described before using a spectrum analyzer. The result of this measurement is the almost horizontal frequency response across the entire range depicted in Fig. 6. This is important for the frequency response because the power should be as independent of the frequency as possible, making predictions about the random numbers much more difficult. Otherwise, it would be possible to see at which frequencies more power is transmitted and thus draw conclusions about the noise signal. Since frequencies above 600 MHz are the mobile radio frequencies (which are recognizable in the spectrum) the RNG to be developed must either be shielded or suppress these frequencies using a filter.

To analyze the electromagnetic compatibility (EMC) resistance of the board, it was tested in an EMC laboratory under the influence of electromagnetic fields. The test board is irradiated with fields of different frequencies and field strengths to investigate the influence on the spectrum. The result is displayed in Fig. 7 where the yellow signal shows the maximum, the green one shows the average and the orange signal shows the minimum of the spectrum. The influence of the electromagnetic fields is clearly visible at the two peaks in the low frequency range. This means that the noise signal source can be influenced by electromagnetic radiation and must be shielded against EMC influence. Alternatively or additionally, other methods such as the use of a differential amplifier could be used. This requires two noise signal sources, whereby the difference between the two noise signals is first formed and then amplified. In this way, interference affecting both noise sources simultaneously can be eliminated.

B. Amplifier (*NXP BGA2818*)

Due to the fact that the amplitude of the noise signal is only 2 mV peak-to-peak, the signal needs to be amplified to be digitized properly and with a sufficient resolution. For this purpose, the NXP BGA2818 is selected as amplifier for the RNG. This component is a monolithic microwave integrated circuit (MMIC) wideband amplifier with an internal matching circuit to $50\ \Omega$ and a nearly constant gain of +30 dB over its complete frequency range. The special feature of this type of

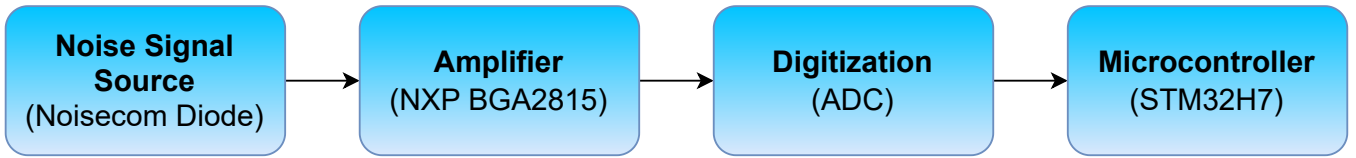


Fig. 4. Building blocks of the hardware development process

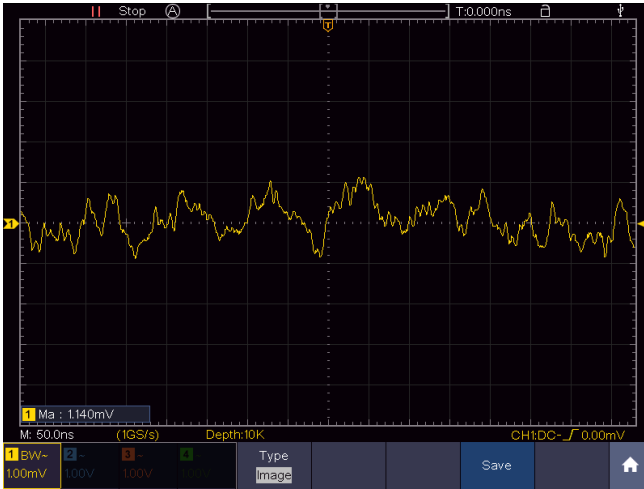


Fig. 5. Noise signal of the NC302BL

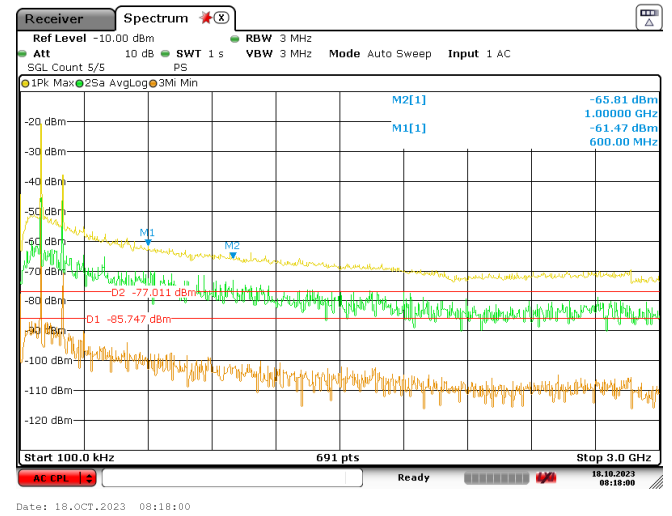


Fig. 7. Power density spectrum of the NC302BL under EMC influence

component is the integration of all active and passive components on a semiconductor substrate. This miniaturization enables the design of circuits down to the millimeter wave range.

However, since an amplification of +30 dB is not sufficient to generate a signal with adequate amplitude, an amplifier stage consisting of two BGA2818 in series was designed on a further test PCB. This corresponds to the procedure mentioned in the previous section of first realizing the individual components as building blocks. After each block is tested individually, they are combined on a common PCB.

To reduce the influence of the power supply to the amplifier and the analog noise signal, the LTM8080 from Analog Devices is chosen as a component for low noise voltage supply. At this stage of development, the LTM8080 is used with the help of a development board. Later on, this will also be integrated on a custom PCB with the other components of the RNG. The LTM8080 can be supplied with a voltage from 6 V to 40 V and generates from this a selectable output voltage from 0 V to 8 V with a ripple in the μV range. [18]



Fig. 8. Spectrum of the BGA2818

In Fig. 8 the spectrum of the BGA2818 with a constant reference signal over the frequency range from the spectrum analyzer is depicted. The reference level before amplification is at -60 dB (light blue horizontal line). There are placed four markers in the spectrum at different frequencies, showing the level at these points of the spectrum. Until marker 4 at 1 GHz, the figure shows a nearly horizontal spectrum, which means that the amplifier works as expected for this frequency range. According to the level at the markers, the gain is slightly below the ideal 60 dB that two amplifiers of this type can theoretically achieve when connected in series.

This means, the BGA2818 is suitable for use in this RNG development. The test PCB amplifies the noise signal up to about 2 V peak-to-peak so that the amplitude of the signal fits for the digitization. However, it must be taken into account that a direct current (DC) voltage offset needs to be added to the voltage signal for digitization. Without this DC offset, the voltage signal can also take on negative values, which can lead to problems during digitization. If digitization is carried



Fig. 6. Power density spectrum of the NC302BL

out using an ADC, for example, it could be damaged by the negative voltages because their operating range is between 0 V and typically 3.3 V.

C. Noise signal source and amplifier in series

After testing the noise signal source and the amplifier individually, the combination of them is to be tested as a next step. The aim is to exclude the possibility of interactions between the two PCBs.

The amplified noise signal has an amplitude of 2 V peak-to-peak. As the signal oscillates around the voltage level of 0 V after amplification, a DC offset of $V_{CC}/2$ should be added to the signal. This means that the analog signal is exactly in the middle between 0 and VCC and can be digitized with an ADC, for example. To protect the digitization circuit, the voltage signal must be also limited to the maximum input voltage of the circuit.

Since the combination of the two boards provides an amplification gain of +60 dB, an emission measurement is then carried out, because interference frequencies are also amplified when they reach the high frequency signal line. This involves measuring whether the circuit boards emit electromagnetic fields and, if so, at what frequency they are located. This ensures that other parts of the RNG or other devices are not influenced or disturbed by the amplifier circuitry.

For this measurement, the structure consisting of the noise signal source and amplifier was placed under a stripline and the radiation of the two boards was measured with its help. The result of the measurement over the frequency range from 100 kHz to 6,25 GHz is depicted in Fig. 9. Up to a frequency range of around 2 GHz, the amplitudes of the signals at the various frequencies are below -80 dBm, which means that the emission of the PCBs in this range is very low. Only two peaks at about 2 GHz and 2,8 GHz attract attention, because the amplitudes at these frequencies are above -60 dBm and thus significantly higher than the remaining frequency spectrum. Although this is noticeable, it is not critical for the time being. The two peaks occur because the matching of the high frequency tracks on the two PCBs is not optimal. This is improved when the components are integrated onto a common circuit board at a later stage.

VI. CONCLUSION AND OUTLOOK

In conclusion, the demand for high-quality random numbers is high due to the increasing use of cryptography. A distinction is made between deterministic RNGs, physical true random generators and non-deterministic RNGs, each consisting of different classes with special properties. For some applications, certain classes are recommended to meet the security requirements. Physical true RNGs are preferable to deterministic RNGs due to the fact that they use unpredictable behavior of specific hardware components like thermal noise as a base for the random numbers. The quality of the random number generators can be evaluated with the help of statistic test suites like the NIST test suite and the Dieharder suite.

The aim of this approach is the initial step in the development approach for a high-performance RNG. Therefore, first a suitable noise signal source is selected based on a frequency spectrum analysis and also a fitting amplifier to prepare the analog signal of the noise signal source for digitization.

For the tests of the first noise signal source, a PCB was developed, and the noise signal measured with an oscilloscope. Measuring the power density spectrum with a spectrum analyzer, the Noisecom diode shows a nearly horizontal frequency response, marking it as a promising candidate. Tests in the EMC laboratory for the influence of electromagnetic fields with different frequencies and field strengths show that the output signal is affected by those fields. Hence, a shielding of the noise source or the whole RNG is necessary.

Since the noise signal has only low power, it needs to be amplified for later digitization. Therefore, the NXP BGA2818 was chosen as an amplifier for the analog noise signal. This amplifier is a MMIC wideband amplifier with a nearly constant gain of +30 dB over its complete frequency range. Due to the low amplitude of the analog signal, two BGA2818 in series are designed on another PCB to test the amplifier individually and in combination with the noise signal source. With this two-stage amplifier circuit, a suitable signal for digitization can be achieved.

This paper focuses on the analog part of the high-performance RNG, thus the noise source and the amplification of the noise signal are of interest. Initially starting with the Noisecom diode and a suitable amplifier, in a later step, other noise sources should be tested and compared.

The next step after amplifying the noise signal is the digitization of the noise data and the statistical evaluation with an associated model. An important aspect is the cryptographic post-processing of the generated random numbers to meet class PTG.3 of the RNGs. In addition, the RNG must also provide online-tests to monitor the correct functioning and quality of the random data during the runtime. The digitization, statistical evaluation, cryptographic post-processing and the online-tests are part of future work.

ACKNOWLEDGMENT

The presented work is part of the research project *KRITIS Scalable Safe and Secure Modules (KRITIS³M)*, which is funded by the Project Management Jülich (PtJ) and the German Federal Ministry for Economic Affairs and Climate Action (BMWK) under funding code 03EI6089A.

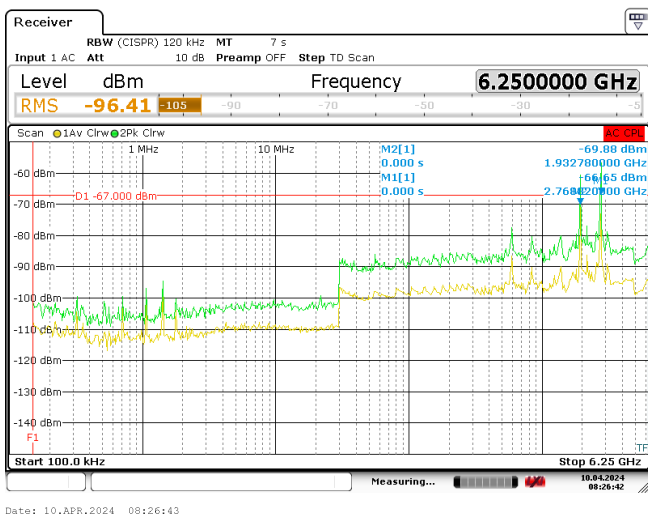


Fig. 9. Emission measurement of noise signal source and amplifier

REFERENCES

- [1] Peter, Matthias and Schindler, Werner. (2023, June) Deterministic RNGs (DRNGs). Publication. Accessed: May 5th, 2024. [Online]. Available: <https://www.nist.gov/system/files/documents/2021/05/28/BSI%20Update-Schindler.pdf>
- [2] Bundesamt für Sicherheit in der Informationstechnik. (2023, January) Technische Richtlinie TR-02102 - Kryptographische Verfahren: Empfehlungen und Schlüssellängen. Publication.
- [3] ——. (2023) A Proposal for Functionality Classes for Random Number Generators. BSI. Accessed: May 5th, 2024. [Online]. Available: https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Certification/Interpretations/AIS_31_Functionality_classes_for_random_number_generators_e.pdf?__blob=publicationFile&v=7
- [4] Frank Bergmann. (2023) Welche Anforderungen werden an kryptografisch sichere Zufallsgeneratoren gestellt? IBB Ingenieurbüro Bergmann. Accessed: May 5th, 2024. [Online]. Available: <https://www.ibbergmann.org/GRUNDLAGEN/>
- [5] A. M. Garipcan and E. Erdem, "A gigabit TRNG with novel lightweight post-processing method for cryptographic applications," *The European Physical Journal Plus*, 2022, Firat University.
- [6] National Institute of Standards and Technology. (2023) NIST SP 800-22: Download Documentation and Software. National Institute of Standards and Technology. Accessed: May 5th, 2024. [Online]. Available: <https://csrc.nist.gov/Projects/Random-Bit-Generation/Documentation-and-Software>
- [7] Brown, Robert G. (2023) Dieharder: A Random Number Test Suite. Duke University. Accessed: May 5th, 2024. [Online]. Available: <https://webhome.phy.duke.edu/~rgb/General/dieharder.php>
- [8] National Institute for Standards and Technology. (2010, April) A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. Publication.
- [9] Marsaglia, George. (2023) Diehard Battery of Tests of Randomness. Florida State University. Accessed: May 5th, 2024. [Online]. Available: <https://web.archive.org/web/20160125103112/http://stat.fsu.edu/pub/diehard/>
- [10] Brown, Robert. (2006) DieHarder: A Gnu Public License Random Number Tester. Publication.
- [11] D. Lubicz and V. Fischer, "Recommendations for the design and validation of a physical true random number generator integrated in an electronic device," *Cryptology ePrint Archive*, Paper 2024/301, 2024, <https://eprint.iacr.org/2024/301>. [Online]. Available: <https://eprint.iacr.org/2024/301>
- [12] ID Quantique. (2023) Standard certified USB Quantum Random Number Generation module. ID Quantique. Accessed: May 5th, 2024. [Online]. Available: <https://www.idquantique.com/random-number-generation/products/quantis-random-number-generator/>
- [13] Frank Bergmann. (2023) Professionelle Zufallsgeneratoren für kryptografisch sichere Zufallszahlen. IBB Ingenieurbüro Bergmann. Accessed: May 5th, 2024. [Online]. Available: <https://www.ibbergmann.org/ZUFALLSGENERATOREN/Mit-Kommando-Interface/PRG310/>
- [14] ——. (2023) Professionelle Zufallsgeneratoren für kryptografisch sichere Zufallszahlen. IBB Ingenieurbüro Bergmann. Accessed: May 5th, 2024. [Online]. Available: <https://www.ibbergmann.org/ZUFALLSGENERATOREN/Ohne-Kommando-Interface/PRG600/>
- [15] Atos SE. (2015, November) CardOS DI V5.3 - The multifunctional smart card operating system with dual interface for the highest demands. Publication.
- [16] Infineon Technologies AG. (2023) SLE 78 SOLID FLASH™ dual-interface and contactless security cryptocontroller Certification to Common Criteria EAL 6+ (high). Infineon Technologies AG. Accessed: May 5th, 2024. [Online]. Available: <https://www.infineon.com/cms/en/product/security-smart-card-solutions/security-controllers/contactless-and-dual-interface-security-controllers/sle-78clfx1m10ph/>
- [17] Wireless Telecom Group. (2023) NC100/200/300/400 Series Chips and Diodes. Wireless Telecom Group. Accessed: May 5th, 2024. [Online]. Available: <https://noisecom.com/products/components/nc100-200-300-400-series-chips-and-diodes>
- [18] Analog Devices. (2024) LTM8080 - 40VIN, Dual 500mA or Single 1A Ultralow Noise, Ultrahigh PSRR Module Regulator. Analog Devices. Accessed: May 5th, 2024. [Online]. Available: <https://www.analog.com/en/products/ltm8080.html>

Considering the Aeronautics Cyber-Security Standards for Multi-Core Platforms

Anthony Fernandes Pires
ONERA
Toulouse, France
anthony.fernandes_pires@onera.fr

Julien Brunel
ONERA
Toulouse, France
julien.brunel@onera.fr

Kevin Delmas
ONERA
Toulouse, France
kevin.delmas@onera.fr

Abstract—New complex functions are emerging for avionic systems. These new functions ask for high performance computing, which mean the need to embed new type of hardware such as hybrid architectures integrating multi or many-core processors. However, these processors are often Commercial Off-The-Shelf and suffer a lack of documentation and predictability. In the all-connected trend of today digital world, these issues can lead to new security vulnerabilities exploitable by malicious people. In the context of the PHYLOG 2 research project aiming at defining a certification framework for multi-core platforms, we study the aeronautics standards ED-202A/DO-326A and ED-203A/DO-356A about airworthiness security. The objective is to take into account these standards at the level of the multi-core processors in order to ensure the compliance of security assessment and development for certification. We present our review and understanding of the standards and their projection at the level of multi-core platforms. In addition, we describe our application on a use case and report our feedback.

Index Terms—cyber-security, multi-core processors, aeronautics, certification

I. INTRODUCTION

New types of hardware are making their way to avionic systems as new complex functions are emerging, such as pilot assistance or flight supervision coupled to machine learning. These new functions are indeed asking for high performance computing. This implies the need to embed hybrid architectures integrating multi or many-core processors and accelerators. However, these processors are mainly COTS (Commercial Off-The-Shelf), so they suffer from low predictability and a significant lack of documentation. This lack of documentation and the complexity of these processors open new vulnerabilities for cyber-attacks. In addition, the avionic systems are becoming more open and connected in the modern digital era (*e.g.* the use of Electronic Flight Bag), leaving these vulnerabilities more accessible to malicious persons.

Standards such as ED-202A/DO-326A and ED-203A/DO-356A offer guidelines, considerations and certification objectives to address airworthiness security. The airworthiness corresponds to the capacity of an aircraft and its systems to operate safely and to carry out their expected function. The airworthiness security consists in the protection of the aircraft against intentional unauthorized electronic interactions. However, these standards have been created to be applicable at the classical levels of aeronautics development: aircraft, system and item. To be able to reason about airworthiness security

for the multi or many-core processors, we need to refine these standards to assess cyber-security at the level of the platform, *i.e.* the processors architecture (hardware) and its executive layer. This is not the system level or the item level, but a level in between.

PHYLOG 2 is a research project¹ supported by DGAC, which aims at defining a certification framework for multi- and many-core hybrid architectures. In this context, we study the security standards ED-202A/DO-326A and ED-203A/DO-356A. The objective is to understand the guidelines and considerations expressed in these standards and how to apply them at a platform level to gain confidence on the airworthiness security of multi-core platforms and ensure compliance with certification.

In this paper, we present the work done to achieve this objective in the context of COTS. It is organised as follows. In Section II, we introduce the cyber-security standards by giving a summary of our understanding of the standards contents and our understanding of the application of a part of the Airworthiness Security Process on the use case example given in the ED-203A/DO-356A. In Section III, we present the challenges brought by the use of multi-core architecture in terms of existing cyber-attacks for processors, and in terms of considered level of development for the application of the standards. In Section IV, we present how we interpret the cyber-security standards at the development level of the platform and we apply this interpretation on a revised version of the Air Management System use case based on a simplified version of a Texas Instruments platform. In Section V, we discuss our feedback and the applicability of the standards at platform level. We conclude in Section VI.

II. UNDERSTANDING THE STANDARDS

A. Overview

Standard ED-202A/DO-326A (Airworthiness Security Process Specification) [14] and companion document ED-203A/DO-356A (Airworthiness Security Methods and Considerations) [15] describe the process, guidelines and regulatory considerations to address airworthiness security. From our understanding, the standards offer different kind of information. First, a definition of the fundamental concepts required to

¹<https://w3.onera.fr/phylog/>

understand and to conduct the Airworthiness Security Process. Second, an overview of the process, activities and suggested methods to be carried out. Third, a list of Security Assurance Objectives to satisfy at the different level of development. Finally, a set of appendixes on methods and examples of application on different use cases. In this section, we focus on giving an overview of the Airworthiness Security Process, introducing the necessary concepts along the way, and we present our understanding of the application of a part of this process on the use case coming from the ED-203A/DO-356A. We finish with an introduction to the Security Assurance Objectives.

B. The Airworthiness Security Process

A simple representation of the recommended process is visible Figure 1.

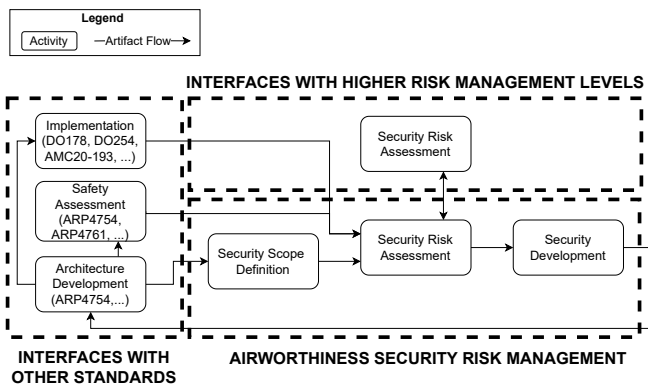


Fig. 1. Airworthiness Security Process preview

It is based on three major activities. First, the security scope definition activity identifies the elements under consideration in the process. The security scope is composed of the assets considered for the airworthiness security, their security perimeter *i.e.* the border between the assets and the external world, and their security environment *i.e.* all the elements external to the security perimeter that can interact with the assets.

The second activity is the security risk assessment that identifies and qualifies the security risks. Based on the definition of the security scope, this activity identifies the threat conditions and their effects, *i.e.* the conditions resulting from potential attacks, the threat scenarios leading to these threat conditions, and the existing security measures. From this information, it allows evaluating the risk for security by calculating the level of threat and evaluating the severity of threat conditions effects. The Level of threat represents the likelihood of a threat condition to occur, while the severity provides a qualitative evaluation of the level of their effect.

The security development activity focuses on the design of security related development (*e.g.*, security measures) adapted to the evaluated security risks and the verification of their effectiveness.

The process always starts by conducting a preliminary Security Risk Assessment at design phase and proceed with

one or many iterations of the Security Risk Assessment once the implementation is available. Indeed, once the security risk is evaluated and the security development to mitigate this risk is achieved by modifying the architecture, it is necessary to re-evaluate the risk to find out whether it is acceptable.

The airworthiness security process also interfaces with other activities. It particularly interacts with the airworthiness security process at a higher level. Moreover, the process interfaces with activities linked to other standards. The architecture issued from the Architecture Development following the standard ED-79A/ARP-4754A [17] is necessary to conduct the airworthiness security process. In addition, the failure conditions coming from the Safety Assessment following standard ED-135/ARP-4761 [16] and the implementation following standards ED-12C/DO-178C [13], ED-80/DO-254 [12] and AMC20-193 are required for the Security Risk Assessment. So these interfaces are mainly represented as inputs to the airworthiness security process.

C. Illustration on The Air Management System from ED-203A/DO-356A

The standards illustrate the application of the different activities of the Airworthiness Security process on practical examples. One of this example is the Air Management System (AMS) described in ED-203A/DO-356A. The standard presents the application of the the Security Scope Definition activity and the Security Risk Assessment activity on this particular use case. The Security Development for this use case is not described in the standards and is out of the scope of this paper. The following is our understanding based on the material available in the standard.

1) *Description:* As described in ED-203A/DO-356A, the AMS of an aircraft fulfils five functions: it provides cabin acclimatization, cabin pressurization, In-Flight information, support for maintenance and support for manufacturing. To manage these functions, the AMS is composed of a Temperature Controller and a Pressurization Controller as depicted in Figure 2.

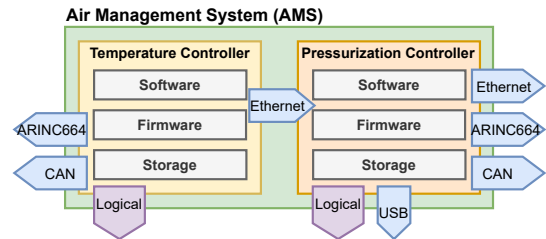


Fig. 2. AMS overview

Each of these controllers contains a software, a firmware and a data storage but also interfaces to the external world. First, there are physical interfaces such as Ethernet interface, ARINC 664 interface, CAN (Controller Area Network) interfaces. The Pressurization Controller is connected to the Temperature Controller via an Ethernet connection. The Pressurization Controller is also the only controller to be equipped with an

USB port. Second, there are logical interfaces that represent digital connections to equipment, such as Aircraft Systems, most of the time via a network of intermediate equipment, e.g. an Ethernet switch. In addition to these descriptive elements, a list of identified Failure Conditions for the AMS is also available in the ED-203A/DO-356A. In this paper, we will primarily focus on the *Loss of pressurization for crew and passengers* which is identified as catastrophic.

2) *Definition of the Security Scope*: The representation of the complete security scope for the AMS is described in Figure 3 (This figure is based on representations from the ED-203A/DO-356A and [19]). It is composed of the assets under consideration, their security perimeter and their security environment.

a) *Assets under consideration*: The security scope definition of the AMS starts by the identification of the assets under consideration. Here, the Temperature/Pressurization controllers are considered in their entirety. Each of them includes, as assets: their constituents (e.g. micro-processor), their functions (e.g. "Provide Cabin pressurization"), their information (e.g. software, firmware, data storage, etc), their interfaces.

b) *Security Perimeter*: The Security Perimeter is the border between the assets and the external world. For the AMS, it is composed of the physical and logical interfaces of the two controllers. The physical interfaces include the interfaces to A664 Switch, to ethernet switch, to Maintenance Ground Support Equipment (GSE) and the USB interface. The logical interfaces are composed of the interfaces to Bleed System, to EFB, to QAR, to IFE File Server, to Avionic Systems and to Airline and Manufacturer network.

c) *Security Environment*: The Security Environment represents the external world of the assets and what can interact with them. It is the place where attacks originate. In the case of the AMS, the security environment includes MRO personnel, pilot, first officer, operator's personnel, operator's maintenance personnel, airline ground infrastructure, manufacturer ground infrastructure and avionic systems. The security environment also covers security assumptions that have been made. One example for the AMS is "The Pressurization Controller can be updated via GSE or embedded Ethernet switch".

3) *Security Risk Assessment*: A part of the security risk assessment activity on the AMS is described in the ED-203A/DO-356A. Here we summarize the threat conditions identification, the threat scenario identification and the security measure characterisation available in the standard. We then carry out ourselves a level of threat evaluation on a security measure of a scenario in order to illustrate this particular part.

a) *Threat condition identification*: An example of Threat Conditions identification on the AMS as presented in ED-203A/DO-356A is given in Table I. It describes a threat condition impacting the asset "Logical interface to the Bleed system" and resulting in the loss of pressurization for the crew and passengers. It is associated to the failure condition coming from the safety assessment, *Loss of pressurization for crew and passengers*. This threat condition considers

the corruption of the pressurization controller leading to the dispatch of misleading commands to bleed system. In terms of impact, each threat condition affects a security attribute of an asset. There are typically three security attributes which are considered: confidentiality, integrity and availability. They are referred as CIA. Here, the threat condition is considered as a loss of integrity for the logical interface to the Bleed system and the severity of the effect is identified as catastrophic.

TABLE I
EXAMPLE OF A THREAT CONDITION FOR THE AMS

Threat Cond.	Asset	Attribute (CIA)	Description	Effects	Severity
TC.1	Logical Interface to Bleed system	Loss of Integrity	Misleading commands to bleed due to Pressurization Controller corruption	Loss of pressure control for crew and occupants	Catastrophic

b) *Threat scenario identification*: An example of threat scenario identification is given in Table II. It represents a scenario where a criminal, terrorist or insider uses the Wireless Connection to bypass security measures², to access the Pressurization Controller Storage and to achieve the threat condition presented in Table I.

TABLE II
EXAMPLE OF A THREAT SCENARIO FOR THE AMS

Threat Sc.	Threat Sources		Attack Path	Security Measures		Threat Cond.
	Attacker	Attack Vector				
TS.1	Criminal, Terrorist, Insider	Wireless connection	Wireless Bridge, Ethernet switch, Pressurization Controller storage	Flight (SR1), Bridge Control	Phase Wireless Access	TC.1

c) *Security Measures Characterization*: In the ED-203A/DO-356A, the Wireless Bridge and its access control is used as example for Security Measures Characterization, as it is on the attack path of the scenario defined in Table II. A summary of this characterization extracted from the ED-203A/DO-356A is presented in Table III. It offers password protection to access the wireless network but its main vulnerability is that it comes with default login and password at the delivery of the aircraft. If the credentials are not changed by the operator, an attacker with knowledge of the default settings can exploit this vulnerability.

d) *Level Of Threat Evaluation*: Following the examples of methods available in ED-203A/DO-356A, the assessment of the level of threat can be carried out in different ways. Here, we consider the assessment of the level of threat based on the evaluation of the effectiveness of protection. This kind of evaluation depends on three criteria. First, the preparation means *i.e.* is previous knowledge required to conduct the

²In the table, SR1 represents the Security Requirement 1 as defined in ED-203A/DO-356A: "The Pressurization Controller shall only accept external connections routed via Wireless Bridge when the aircraft is on-ground and engine is off."

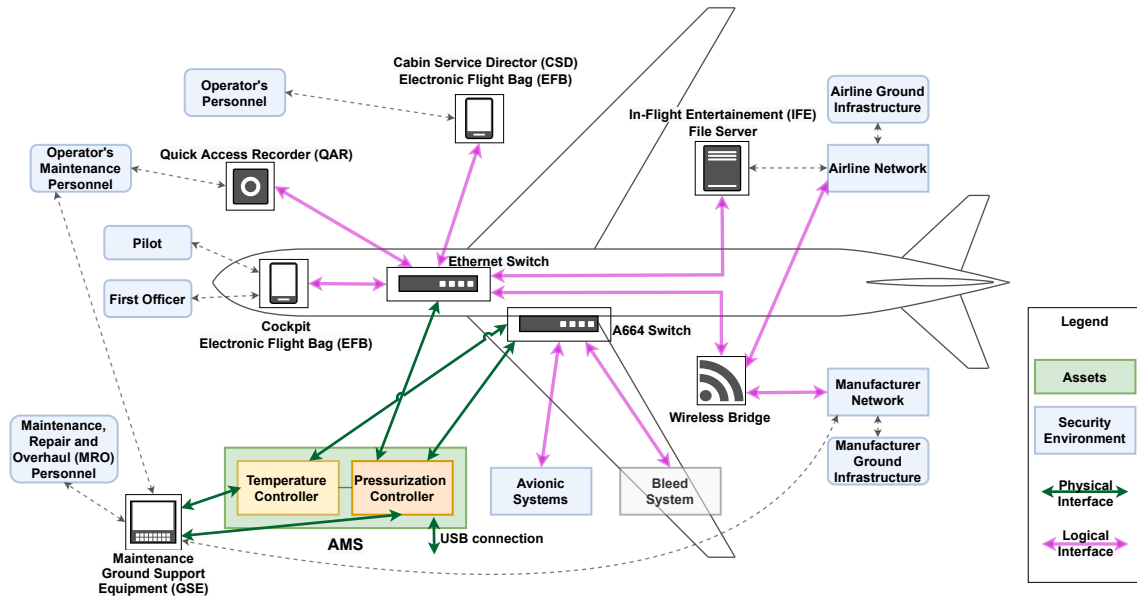


Fig. 3. AMS Security Scope from ED-203A/DO-356A

TABLE III
EXAMPLE OF CHARACTERISATION OF THE SECURITY MEASURE
WIRELESS BRIDGE ACCESS CONTROL

Description	Protected Assets	Capability	Type of effect	Position in the architecture	Known vulnerability	Dependencies
Standard wireless access point with optional channel encrypting and access controls	AMS system, functions and data	Provide password protection, interface hardening, etc	Preventive	In the security perimeter border	Delivered with standard user name and password	Need to enforce password definition after delivery or replacement

TABLE IV
LEVEL OF THREAT DEPENDING ON EFFECTIVENESS SCORE

Level of threat	Effectiveness	A
Very High	None	< 7
High	Basic	≥ 7
Moderate	Moderate	≥ 13
Low	High	≥ 19
Extremely Low	Very High	≥ 25

attack? Second, the window of opportunity *i.e.* when will the attack be possible? Finally, the execution means, which prerequisites need to be carried out?

For each of this criterion, a score table is presented in ED-203A/DO-356A in order to give a score depending on predefined answers for the associated question. By combining the three scores, the user obtains an effectiveness score of the protection, denoted A. The level of threat can then be decided following the score table presented in Table IV. It is then up to the user to combine the effectiveness scores of the different security measures, following specific rules defined in ED-203A/DO-356A annex E, to obtain the total score of the effectiveness of protection and so the level of threat of the threat scenario.

For the AMS use case, we tried to carry out this assessment on the Wireless Bridge Access Control in the context of the scenario presented in Table II, as the illustration on this example does not seem to be available in the standards. In terms of preparation means, the Wireless Bridge Access Control corresponds to a standard equipment and uncontrolled information, as the equipment is delivered with standard login

and password. The obtained score for this criterion is then 2. In terms of window of opportunity, the access to the wireless bridge is only available on ground engine off which is a reduced window. The evaluated score is then 2. In terms of execution means scale, the attacker needs to be proficient with standard equipment to gain access. The given score is then 4.

The total computed effectiveness score for the Wireless Bridge Access Control is 8. Following Table IV, it corresponds to a basic effectiveness and a high level of threat for this protection. However, other security measures are available for this particular threat scenario and would need to be taken into account to obtain the final level of threat.

D. Assurance Objectives

Finally, the standards also defined security assurance objectives for certification. There are 39 objectives organised following 13 categories (*e.g.* security risk assessment, design, verification, etc). Each objective is tagged with its scope of application, *i.e.* Aircraft development, System development or Item development and its level of application according to the Security Assurance Level (SAL). In the same vein as the DAL, the SAL represents the level of rigor to demonstrate, in terms of security, for a product and its development process. It goes

from level 0, no protective effect, to a maximum level of 3. It is determined based on the severity of threat condition effects to which the product is exposed to and it is assigned to the security measures and assets of the product. SAL is considered out of the scope of this paper, as we reviewed every objective in our work.

To give an example of Assurance Objective, let us consider for the remainder of the paper the objectives O1.1, O1.2 and O1.3 from the Security Risk Assessment category, all applying to the Aircraft and System development level. They are respectively: The security scope is established and validated (O1.1), the Threat Condition Identification and Evaluation is complete and validated (O1.2), the Preliminary Aircraft/System Security Risk Assessments and Aircraft/System Security Risk Assessments are performed and consistent with related aircraft/system safety assessments (O1.3).

III. THE SECURITY CHALLENGES BROUGHT BY MULTI-CORE PLATFORM

A. Cyber-Attacks at the level of the processors

Many works of the literature have identified cyber-attacks targeting processors, including multi and many-core processors. For instance, [21] presents a sophisticated attack that retrieves information on a secret cipher key by observing the shared Last Level Cache (LLC) of a platform. This attack is called *Flush + Reload* and illustrates a larger category of attacks called side channel attacks. Side channels attack principle is to infer information by observing a phenomenon correlated to the computation (e.g. power consumption, execution time, sound, etc.). Here the attack context is a multi-core platform with shared LLC. On one core, a victim program encrypts a message with its secret key. On another core, an attacker program seeks to retrieve the key to decrypt the message. All encryption programs are considered accessing the same memory location where its instructions are stored. The cache structure of the platform is said to be inclusive, *i.e.*, when a user flushes a line in cache, the line is flushed from all cache levels and for all users. The encryption algorithm used is a non-protected version of RSA. The RSA algorithm is composed of three basic operations: square, multiply, modulo. In a non-protected version of RSA, the order of execution of these operations depends on the bit stream of the secret key. Hence, the order of the operations executed by the victim informs on the secret key. The scenario of the attack is the following. When the victim is encrypting data, only one out of the three operations will have a reduced access time, meaning that it has been loaded in cache by the victim. After the attacker measured the time of access to all the commands and found the one executed by the victim, it flushes all the three operations from the cache and waits for the victim to execute another operation. Repeated during the whole encryption, this process gives the sequence of executed commands, which informs on the secret key used by the victim. In terms of security measures, it is quite easy to protect against some of this type of attacks. However, it is very difficult to protect

against all of them. Some security measures may need hardware modification or could be only software. For instance, an easy way of protection for the case described in [21] is to make the algorithm load all the operations at each step whatever the one it executes currently. The observation of the exploitable phenomena is then mitigated.

Other versions exist such as *Prime + Probe* [10] or *Flush + Flush* [5] that are all based on the same concept of observable phenomena but with different configurations and efficiency. Additional ways to exploit those phenomena via side channel attacks are described in [22]. From another perspective, work such as [20] studies protection mechanisms against side channels attacks in the context of Simultaneous Multi-Threading (SMT) processors. Indeed, SMT architecture allows multiple threads to compete for shared resources on a single core. Even if it offers performance benefits, it comes at the cost of higher security risks, especially from side channels attacks which exploit shared resources. The protection mechanism described in [20] is based on a spatial and time partitioning of the execution units and ports to prevent side-channel execution on this kind of processor.

Side channel attacks can also be part of more recent sophisticated attacks, such as Meltdown [7] or Spectre [6]. Both of these attacks take advantage of a specific feature that is implemented in modern processors called out-of-order execution. This feature allows a processor to speculate on future operations and schedule them to idle execution units. If the speculation becomes correct, the changes induced by the operations are applied otherwise they are discarded. It increases performance but provides observable side effects which can be exploited. For instance, Spectre attacks trick a victim into speculatively performing operations that would not happen during the nominal program execution and which leak confidential information. The side effects can then be exploited to retrieve this information via side channel attacks.

In a different vein, [9] describes an attack on the Input-Output Memory Management Unit (IOMMU) of a platform that, in a particular configuration, gives a malicious hardware access to the whole memory space. On some processors, Direct Memory Access (DMA) can make direct accesses to the shared memory. It allows to bypass the CPU (and thus the OS if any) so that other devices, such as Field-Programmable Gate Arrays (FPGA), could access in parallel the memory. For example, an FPGA dedicated to cryptographic operations could encrypt and decrypt messages in parallel to the main OS running. But such an unlimited access is not without dangers if the code on the device is malicious or simply flawed. An IOMMU can then be implemented to filter memory access requests. It serves both as a protection against illegal access and as an abstracted interface since it translates virtual addresses to physical addresses. However, at the start of the system, the IOMMU is not immediately configured while the DMA access is enabled immediately after the CPU initialization. The IOMMU is set after some time in the boot process, which leaves a time window for unlimited memory access. This is exploited in this attack. The attack context is

a heterogeneous architecture with one or more CPUs and a Linux OS. The architecture also includes hardware devices like FPGA, external to the OS. These hardware devices are connected to the main memory via DMA and an IOMMU is set to protect the main memory. It is considered that one of the hardware devices is infected by a malicious program at startup. The scenario is the following. On startup, the Linux kernel creates the configuration table of the IOMMU that contains the access policy rules. This table is in the main memory before it is loaded in the IOMMU internal register, where it becomes protected. However, while this table is in the main memory and before it is loaded, the malicious hardware will rewrite it to give itself access to the whole memory space. This scenario is based on the assumption that the DMA access is enabled by default at startup. Nowadays, this is true for many platforms because of legacy reasons, as explained in [9]. Concerning security measures, this work also illustrates the vulnerability that a security mechanism at the level of the platform (the IOMMU) can have and that could be exploited to bypass the protection.

In [4], the authors list a number of attacks targeting Network-on-Chip (NoC). A NoC is a network between System-on-Chips (SoC) which are integrated circuit containing processors. A NoC design combines notions from the network and the hardware domains, including their security vulnerabilities, and creates a completely new set of attack possibilities between processors. For example, we present one of the attacks described in [4], the attack concerning packet corruption at routers. In this attack, a Trojan hardware is inserted in a router of the network and is able to either copy and resend packets, send forged packets or tamper the data inside a transiting packet. The context of the attack is based on two CPU clusters on the network, a sender and a receiver, which need to communicate. A hardware element has been introduced into a router of the network that is on the path between the sender and the receiver. There is no encryption or authentication mechanism implemented on the network and the Trojan is sufficiently stealthy to not be detected by traditional means of network monitoring. The attack scenario is quite simple. The message from the sender to the receiver is intercepted by the Trojan and is modified or additional messages are forged. In terms of security measures, detecting a Trojan hardware is very difficult if not impossible. Current research works are more focused on protecting the network in case of such compromising. Protection mechanisms include what is already done in networks, i.e. error-correcting codes, encryption, authentication, etc. However, this attack requires the installation of hardware on the network, which makes the feasibility of the attack more difficult.

Despite the numerous processor level attacks identified in the literature, we did not find any example of attacks in the aeronautics domain or any practical applications of ED-202A/DO-326A and ED-203A/DO-356A on a multi-core platform.

B. The need to consider a development level of the platform

In this paper, we consider that a platform is composed of several hardware and software items interacting between them. It includes a processor architecture, a platform configuration, which can be hardware and/or software, and an executive layer, i.e. the hypervisor. Cyber-attacks reported in the literature show that it exists many attacks happening at the level of processors and their interactions. Security measures are also defined at this level of details. In this case, it becomes essential to consider the whole content of a platform to carry out the Airworthiness Security process. As seen in Section II, the standards ED-202A/DO-326A and ED-203A/DO-356A consider the traditional development level of an aeronautic development. However, the platform is neither a system nor an item. It is something in-between. It is then necessary to consider an alternative level of development to apply the recommendations of the cyber-security standards. We suggest studying their application at the level of the platform.

IV. INTERPRETATION OF THE STANDARDS AT THE PLATFORM LEVEL

Based on our review of the standards ED-202A/DO-326A and ED-203A/DO-356A and the need to reason at platform level, we first extracted the concepts needed to carry out the activities, along with their definitions and relationships. Accordingly, we designed a representation of the security process as we understood it and we outlined it at the level of the platform, along with the Security Assurance Objectives. In addition, we took the AMS example described in Section II-C and we brought it at the platform level to review the activities at this specific level. The work achieved for the process, the assurance objectives and the use case is summed up in this section.

A. Interpretation of the Standards at the Level of Multi-Core Platforms

1) *Interpretation of the Airworthiness Security Process:*
We present our understanding of the Airworthiness Security Process in Figure 4.

In particular, we describe the precise links between the different activities of the Airworthiness Security Risk Management introduced in Figure 1. Three activities are out of the Airworthiness Security Risk Management. The architecture development provides the architecture to the three main activities of the Airworthiness Security Risk Management. The Platform Safety Assessment provides the failure conditions to the Platform Security Risk Assessment and the Implementation activity provides the implementation and derived requirements. From this point, the Security Scope Definition is carried out and supplies the security scope (Assets under consideration, security perimeter, security environment and security assumption) to both the Platform Security Risk Assessment and the Platform Security Development. Based on the provided artefacts, the Platform Security Risk Assessment is first conducted and supplies security requirements, the evaluation of the security risks and their level of threat to enable the

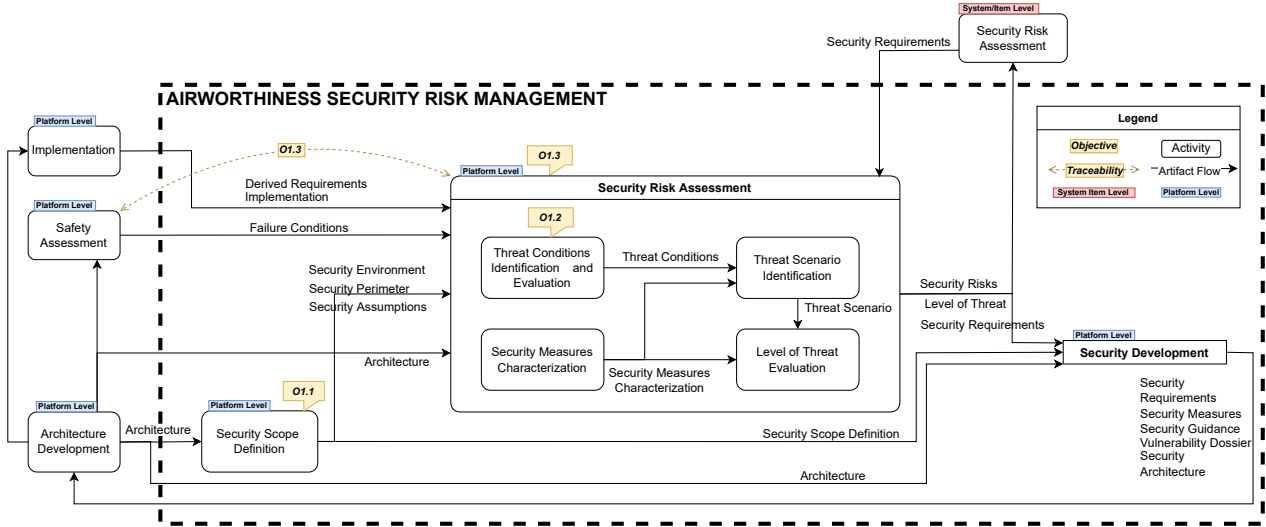


Fig. 4. Extract of our interpretation of the standards of the Airworthiness Security Process

Platform Security Development. The detailed activities of the Security Risk Assessment follow the representation available in ED-203A/DO-356A. Once the Security Development is conducted, it provides all the necessary information to the architecture development in order to take into account the required modifications to mitigate the security risks.

All the mentioned activities happen at platform level. Following Figure 1, there are also links between the activities of the Airworthiness Security Risk Management and the Security Risk Assessment carried out at an upper level. Here we consider that this upper level could be the System level or the Item level.

2) *Interpretation of the Assurance Objectives:* Assurance objectives have been interpreted for the platform regardless their SAL application. The idea was to obtain an overview of their relevance at the level of the platform. Our review shows that most of them can be interpreted as they are specified at platform level, as long as they now specifically mentioned the platform level. For example for O1.3 mentioned in Section II-D, it becomes *The Preliminary Platform Security Risk Assessments and Platform Security Risk Assessments are performed and consistent with related Platform Safety Assessment*. Only two objectives are deemed not interpretable at the level of platform. They correspond to objectives linked to configuration management process and credentials, which are not specific to the platform, but to the development process of the overall system.

In addition, each objective was mapped to the activity we judged it was related to in our process interpretation. For example, the objectives O1.1, O1.2 and O1.3 mentioned in Section II-D are positioned on the process presented in Figure 4. The three of them are mapped to the respective activities they referred to. In addition, O1.3 mentioned the need of consistency between the security risk assessment and the related safety assessment. This need is interpreted in

Figure 4 as a traceability link between both activities.

B. Bringing the AMS use case to the platform level

1) *Description:* To reason at platform level, the AMS described in Section II is adapted on a simplified representation of the KEYSTONE TCI6630K2L multi-core platform from Texas Instruments composed of only three cores. This representation is presented in Figure 5. It describes the structure of the platform itself and the location of the assets of the AMS. The representation of the structure of the platform follows an initiator-target modelling described in [3]. This modelling is based on three types of components. The initiator component can initiate a transaction, *i.e.* a request for resources. The target component is the final destination of the transaction. The transporter component routes the transaction from the initiator to the target.

In terms of assets, the two controllers composing the AMS are hosted on two different cores. The Pressurization Controller is hosted on a C66 DSP core, namely *CorePac0*. Its software and specific data (computation data, part numbers, certificates, cryptographic keys) are stored in the core SRAM. The Temperature Controller is hosted on an ARM core. Its software is stored in the MSMC SRAM and the regular controller data (Health Monitoring, CAN Messages, ARINC messages, configuration) are stored in the external DDR memory, along the regular controller data for the Pressurization Controller. The firmwares of both controllers are stored in the Boot ROM. In addition to these two cores, only one another core C66 DSP core, namely *CorePac1*, is present in the platform. The core is unused by the AMS but still present for the sake of the example.

2) *Security Scope Definition:* We conduct the security scope definition on the use case by identifying the assets under consideration, the security perimeter and the security environment.

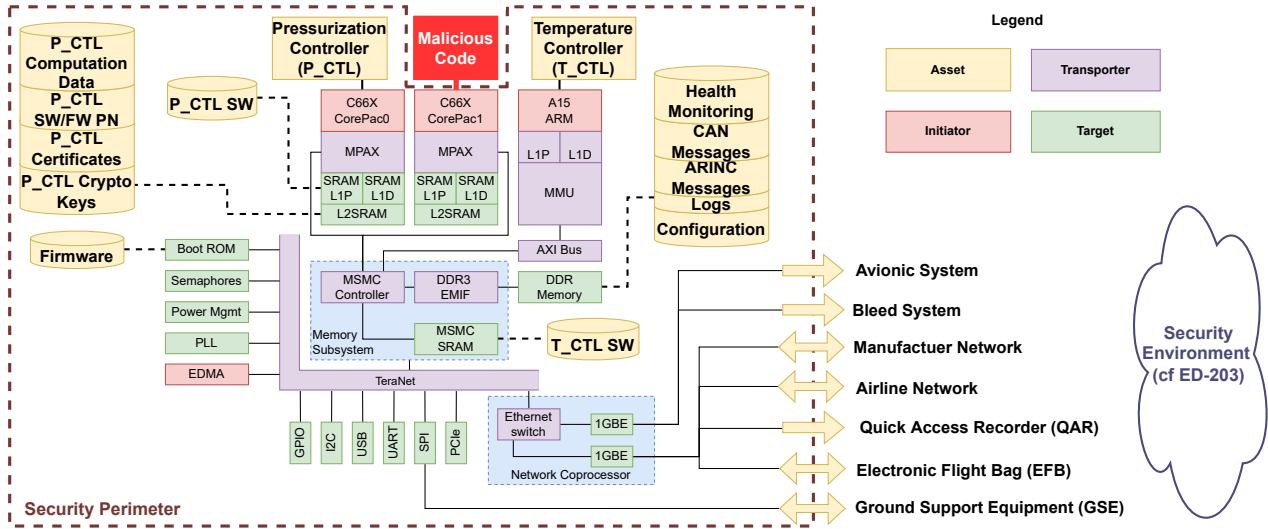


Fig. 5. AMS adapted on a simplified keystone platform

a) *Assets under consideration*: The assets under consideration correspond to the same assets as in the original AMS example in ED-203A/DO-356A described in Section II, but mapped on different elements (*e.g.* the cores). In this way, we consider both controllers in their entirety (constituents, data and interfaces) as assets.

b) *Security Perimeter*: Here the security perimeter includes the security perimeter of the AMS defined in ED-203A/DO-356A. It contains the physical and logical interfaces to different equipment, to Bleed System, to IFE File Server, to Avionic Systems and to Airline and Manufacturer network. In addition, at platform level, the security perimeter includes the logical interface with an external code, most likely malicious, present in the unused C66 DSP core CorePac1. This choice is based on an assumption that the external code, which is not part of the AMS, is already uploaded on the platform and executes on CorePac1. This assumption has been made in order to reason about cyber-security at platform level.

c) *Security Environment*: The security environment is the same as the original AMS example described previously with the difference that it now includes the malicious code.

3) *Security Risk Assessment*: We conduct the four steps of the Security Risk Assessment on this use case. We start by the identification of threat conditions. We identify two threat conditions representative of our problematic. We continue by the identification of threats scenarios. Finally, we give an example of security measures characterisation and evaluate the level of threat.

a) *Threat condition identification*: An example of Threat Conditions identification on the AMS platform example is described in Table V. The first threat condition, TC.1, is similar to the one presented in the original AMS example in Section II-C3. It considers the sending of erroneous data to bleed system. It is considered as a loss of integrity for the logical interface to the Bleed system. The second threat con-

TABLE V
EXAMPLES OF THREAT CONDITIONS FOR THE AMS PLATFORM EXAMPLE

Threat Cond.	Asset	Attribute (CIA)	Description	Effects	Severity
TC.1	Logical Interface to Bleed system	Loss of Integrity	Pressurization controller sends erroneous data to the Bleed system	Loss of pressure control for crew and occupants	Catastrophic
TC.2	Logical Interface to Bleed system	Loss of Availability	Pressurization controller does not send data to the Bleed system	Loss of pressure control for crew and occupants	Catastrophic

dition, TC.2, has been identified especially for this use case. It considers the blocking of data sent by the pressurization controller to the Bleed system. It is considered as a loss of availability for the logical interface to the Bleed system. Both threat conditions can lead to the loss of pressurization for the crew and passengers and are identified as catastrophic.

b) *Threat scenario identification*: Two threat scenarios related to the previously described threat conditions are identified and given in Table VI. The first scenario deals with threat condition TC.2 and considers that the malicious code executing on CorePac1, modifies the MPAX (Memory Protection and Address eXtension) configuration register in order to block CorePac0, *i.e.* the pressurization controller, to access the Ethernet switch of the platform and so to communicate data to the Bleed system. The second threat scenario is associated to threat condition TC.1. It also considers that the malicious code present in CorePac1, modifies the MPAX configuration register but in this case to give itself writing rights on CorePac0 and to corrupt the pressurization controller. In this case, use of side channel attacks can inform the malicious code on where are stored the Pressurization Controller Computation data to corrupt. Note that these scenarios follow the assumption that

TABLE VI
EXAMPLES OF THREAT SCENARIOS FOR THE AMS PLATFORM EXAMPLE

Threat Sc.	Threat Sources		Attack Path	Security Measures	Threat Cond.
	Attacker	Attack Vector			
TS.1	Criminal, Terrorist, Insider	CorePac1	1- Malicious code modifies MPAX configuration register to block CorePac0 access to Ethernet Switch 2- Logical Interface to Bleed system is blocked	MPAX, OS Access Control	TC.2
TS.2	Criminal, Terrorist, Insider	CorePac1	1- Malicious code modifies MPAX configuration register to give itself writing rights on CorePac0 L2SRAM 2- Logical Interface to Bleed system is corrupted	MPAX, OS Access Control	TC.1

the malicious code has already been uploaded on the platform. Also note that both the MPAX, which is in charge of enforcing the rules included in the configuration register, and the OS, which is in charge of loading the configuration table into the MPAX configuration register, are countermeasures.

c) *Security Measures Characterization*: An example of security measure characterization on the platform is conducted on the OS access control. It is a preventive security measure that corresponds to a Linux Operating System (Symmetric Multi Processing). The access control protects the memory storage of the platform but presents a known vulnerability at boot time, as described in Table VII and as seen in Section III.

TABLE VII
EXAMPLE OF CHARACTERISATION OF THE SECURITY MEASURE OS ACCESS CONTROL

Description	Protected Assets	Capability	Type of effect	Position in the architecture	Known vulnerabilities	Dependencies
Linux Operating System (Symmetric Multi Processing)	Memory storage of the platform	Provide memory access protection	Preventive	Inside the security perimeter	During activation of access control mechanisms at boot time, it is possible to modify configuration tables [9]	N/A

d) *Level Of Threat Evaluation*: For this use case, we used the same type of effectiveness of protection assessment as in the section II-C3, but carried out at a global level on the platform for the considered scenarios. In terms of *preparation means*, the platform can be considered as a special equipment with a specific configuration to host the AMS. So insider knowledge or significant preparation time would be needed for the attack. The obtained score for this criterion is then 6. In terms of *window of opportunity*, the identified attack can only be carried out during a very restricted time slot independent from the flight phase, e.g. during system reboot. The evaluated score is then 8. In terms of *execution means* scale, the attack requires experts in multiple domain in order to be carried out on the platform. The given score is then 12.

The total computed effectiveness score for the platform and the considered scenarios is 26. Following Table IV, it

corresponds to a very high effectiveness of protection and a very low level of threat.

V. FEEDBACK AND DISCUSSIONS ON THE CONSIDERATION OF THE INTERNAL PLATFORM FOR SECURITY

We presented our understanding of the standards ED-202A/DO-326A and ED-203A/DO-356A through the AMS use case and we adapted this use case to the platform level in order to reason about cyber-security in the context of the PHYLOG 2 project. The AMS as described in ED-203A/DO-356A lacks some details that made it challenging to understand. The projection at platform level was not an easy task either, example of attacks, or application of the standards, on aeronautic use case were not available in the literature. We had to make assumptions to reason about cyber-security at platform level.

The strongest assumption was made on the attack origin: the malicious code is already uploaded on the platform and executes on CorePac1. The question of how to get to this situation is the most difficult one. The starting point of the attack pathway is in fact external to the platform. We can make the same observation on the attacks on processors described in the literature. As we have seen, the work presented in [9] and [4] consider that the infected hardware device is already present in the architecture. From our point of view, adding a malicious element to the platform would require a lot of effort to bypass a number of existing security measures, e.g. physical access to the platform, ability to load malicious code, etc. This particular problem of loading malicious code challenges the plausibility of the attack as we understand it today. However, cases of more sophisticated attacks on information systems, *i.e.* attacks composed of a series of attacks on different assets to target another specific asset, are appearing today. For instance, the supply-chain attack reported in [11] started by gaining access to the source code of a particular software developed by a company to directly insert malware in it. In this way, the malware was automatically distributed to all the customers in the next update via official channels, allowing the attackers to gain access to customer's information without directly attacking them. The update containing the malware was installed by thousands of customers, including numerous U.S. federal agencies. We could imagine a similar sophisticated scenario happening for the update of a COTS platform. We could also imagine the same scenario for an Open Hardware platform where the shared high-level hardware description of the platform is targeted to impact the final users. This increasing sophistication of cyber-security attacks reinforces our idea of the necessity to study cyber-security at the level of multi-core platforms in aeronautics in order to gain confidence on the protection against attacks reaching this level. However, the security analysis cannot start at the platform level. It needs first to be conducted at an upper level to get a global picture of the origin of the attack and the attack path itself.

Considering the security risk of a multi-core platform comes with challenges. It would first mean study the vulnerability,

threat conditions, threat scenarios and the security measures at this level. In Section III, we reported on existing attacks, scenarios and security measures on processors, including multi-core platforms. In the general case, hardware and software cyber-attacks have been documented in cyber-security databases online. For instance, as mentioned in [8], there are databases dedicated to vulnerabilities found in existing systems such as CVE [2]. This database provides a catalogue of known vulnerabilities in applications, operating systems and hardware products. There are also databases reporting attack patterns as mentioned in [8] and [18], such as *Common Attack Pattern Enumeration and Classification* (CAPEC) [1]. The CAPEC database provides more than 500 known cyber-security attack patterns, including hardware-related ones. While all these databases are not dedicated especially to multi-core platforms, they offer a more global overview of actual known attacks in the cyber-security domain. But vulnerabilities, attacks scenarios and security measures at the level of multi-core platform might be very well dependent of the studied architecture. As we have seen in the example in Section IV, they seem specific to the architecture we used. The use of COTS and their lack of predictability and documentation may also make the task more complex. Further work would be necessary to understand precisely all these challenges by studying the application of the airworthiness security process on a larger panel of multi-core platforms, including Open Hardware solutions.

VI. CONCLUSION

To conclude, we presented our understanding of the Security Airworthiness process as described in ED-202A/DO-326A and ED-203A/DO-356A and assurance objectives that are relevant at platform level. Our main feedback is that even though the standards provide rich details on the different cyber-security concepts and lay the foundation of a sound assessment process, the process as described in these standards may lack some details that made it challenging to understand for non-insiders. The standards would benefit from a use case which would serve as a common thread for an application of the Security Airworthiness process in order to give an end-to-end example to the future applicant. In addition of our understanding, we applied our interpretation of the Security Assessment process on a platform use case and we opened the discussion on the usefulness of considering the internal platform in the context of cyber-security analysis. Our work is limited to the study of a fictitious use case inspired from ED-203A/DO-356A and to an exploratory overview of the current literature on attacks on processors. Nevertheless, it introduces an example of application of the standards in the case of a multi-core platform. In this sense, it gives a first glance on the difficulties behind an attack inside a multi-core platform, but the increasing sophistication of today cyber-attacks suggests it might be worth starting discussing the application of the standards at platform level.

ACKNOWLEDGMENT

The work presented in this paper is part of the PHYLOG 2 project supported by the Directorate General of Civil Aviation (DGAC). It is funded by the French government through the France Relance program, based on the funding from the European Union through the NextGenerationEU program.

REFERENCES

- [1] CAPEC. <http://capec.mitre.org/>. Last access: 2024-03-27.
- [2] CVE. www.cvedetails.com/vulnerabilities-by-types.php. Last access: 2024-03-27.
- [3] F. Boniol, Y. Bouchebaba, J. Brunel, K. Delmas, C. Pagetti, T. Polacsek, and N. Sensfelder. A service-based modelling approach to ease the certification of multi-core COTS processors. In *SAE AEROTECH@ Europe*, Bordeaux, France, Sept. 2019.
- [4] S. Charles and P. Mishra. A survey of network-on-chip security attacks and countermeasures. *ACM Computing Surveys*, 54(5), 2021.
- [5] D. Gruss, C. Maurice, K. Wagner, and S. Mangard. Flush+flush: A fast and stealthy cache attack. In J. Caballero, U. Zurutuza, and R. J. Rodríguez, editors, *Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 279–299, Cham, 2016. Springer International Publishing.
- [6] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom. Spectre attacks: Exploiting speculative execution. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1–19, 2019.
- [7] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, M. Hamburg, and R. Strackx. Meltdown: reading kernel memory from user space. *Commun. ACM*, 63(6):46–56, may 2020.
- [8] N. Messe, N. Belloir, V. Chiprianov, I. Cherfa, R. Fleurquin, and S. Sadou. Development of secure system of systems needing a rapid deployment. In *2019 14th Annual Conference System of Systems Engineering (SoSE)*, pages 152–157, 2019.
- [9] B. Morgan, E. Alata, V. Nicomette, and M. Kaaniche. IOMMU protection against I/O attacks: A vulnerability and a proof-of-concept. *Journal of the Brazilian Computer Society*, 24, 12 2018.
- [10] D. A. Osvik, A. Shamir, and E. Tromer. Cache attacks and countermeasures: The case of aes. In D. Pointcheval, editor, *Topics in Cryptology – CT-RSA 2006*, pages 1–20, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.
- [11] S. Seisert, B. Schneier, H. Okhravi, F. Massacci, T. Benzel, C. Landwehr, M. Mannan, J. Mirkovic, A. Prakash, and J. B. Michael. Perspectives on the solarwinds incident. *IEEE Security & Privacy*, 19(2), 2021.
- [12] RTCA, Inc / EUROCAE. DO-254 / ED-80 Design Assurance Guidance for Airborne Electronic Hardware, 2005.
- [13] RTCA, Inc / EUROCAE. DO-178 / ED-12C - Software Considerations in Airborne Systems and Equipment Certification, 2011.
- [14] RTCA, Inc / EUROCAE. DO-326A / ED 202A - Airworthiness Security Process Specification, 2014.
- [15] RTCA, Inc / EUROCAE. DO-356A / ED 203A - Airworthiness Security Methods and Considerations, 2018.
- [16] SAE / EUROCAE. ARP-4761/ED-135 Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment, 1996.
- [17] SAE / EUROCAE. ARP-4754A/ED-79A Guidelines for development of civil aircraft and systems-enhancements, novelties and key topics, 2011.
- [18] A. Shaked and Y. Reich. Model-based threat and risk assessment for systems design. In *Proceedings of the 7th International Conference on Information Systems Security and Privacy - Volume 1: ICISSP*, pages 331–338. INSTICC, SciTePress, 2021.
- [19] P. Skaves. Electronic flight bag (EFB) policy & guidance information, paper # 263. In *2011 IEEE/AIAA 30th Digital Avionics Systems Conference*, pages 1–30, 2011.
- [20] D. Townley and D. Ponomarev. Smt-cop: Defeating side-channel attacks on execution units in smt processors. In *2019 28th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 43–54, 2019.
- [21] Y. Yarom and K. Falkner. FLUSH+RELOAD: A high resolution, low noise, 13 cache Side-Channel attack. In *23rd USENIX Security Symposium*, pages 719–732. USENIX Association, 2014.

- [22] Y. Zhou and D. Feng. Side-channel attacks: Ten years after its publication and the impacts on cryptographic module security testing. *IACR Cryptology ePrint Archive*, 2005:388, 01 2005.

Session Tu.4.A
Assurance Case for ML

Tuesday 11th June

17:00

–

Auditorium

Assurance Cases to face the complexity of ML-based systems verification

Vincent Mussot^{*‡}, Eric Jenn[‡], Florent Chenevier^{*§}, Ramon Conejo Laguna^{*‡}, Yassir Id Messaoud^{*}, Jean-Loup Farges^{*†}, Anthony Fernandes Pires^{*†}, Stephen Creff^{*}, Florent Latombe[¶]
^{*}IRT SystemX, [†]ONERA, [‡]IRT Saint Exupéry, [§]Thales, [¶]Obeo

Abstract—The verification and validation of AI-based systems raise new issues that are not easily addressed by existing practices and standards. We think that this gap is actually an opportunity to introduce new practices and establish a clearer and more formal link between the engineering activities and artefacts, the expected properties of the system, and the verification and validation evidence.

Therefore, in this paper, we describe and illustrate an approach integrating (i) the definition and modelling of an AI-based system engineering workflow, (ii) the identification of the trustworthiness properties, and (iii) the argumentation demonstrating the satisfaction of these properties. This approach is centred on the model of Assurance Cases, a semi-formal representation of argumentation which supports the claim of system trustworthiness. In addition, we present supporting tools for this formalism that enable the automatic production of Verification and Validation plans for specific properties of AI-based systems.

Index Terms—Assurance Case, Machine Learning, Robustness, V&V

I. INTRODUCTION

There is an obvious and strong willingness to leverage the capabilities of Machine Learning in all domains of industry, including those delivering business- or safety-critical services. However, the adoption and deployment of this technology remain slow, for we fundamentally lack confidence in these methods.

In the Confiance.ai program¹, we address this problem by tackling all stages of the development of systems involving Artificial Intelligence (AI), spanning from defining engineering workflows to implementing and deploying ML-related components on hardware platforms. This process revolves around the notion of trustworthiness, which becomes increasingly crucial as AI plays a larger role in the system. Besides, the program's effort is essentially directed toward identifying and addressing the novel challenges that emerge during the integration of AI into such systems. More specifically, focus is placed on specific attributes of ML-based systems, such as model robustness,

explainability, or fairness, with the broader goal of improving our confidence in the final system. Besides, the justification of this confidence forms a necessary condition of the safety case for such system. It requires a global approach to explicit the link between the different parts of this safety case: the Engineering Items produced during development, the Activities that produce them, the expected Properties of these items and the justifications that these properties actually contribute to the confidence on the ML-based system, all these used in a structured argumentation allowing to demonstrate the satisfaction of those properties. This approach is needed to guarantee the completeness, transparency and auditability of the argumentation. It also supports traceability and impact analysis.

The approach is based on Assurance Cases (ACs) [1], a method that associates the property of interest to be demonstrated with the evidence supporting it through convincing and valid reasoning. The Goal Structuring Notation standard (GSN) [2] used in this work is one of several formalisms (e.g. Claims Arguments Evidence (CAE) [3] or Structured Assurance Case Metamodel (SACM) [4]) designed to model an assurance case. It consists of decomposing, according to a specific strategy, a high-level claim representing a property of interest into elementary sub-goals that can be easily proven with evidence. This formalism has already been adopted in several industries, particularly under the specific form of Safety Cases [5] (when the argued property is safety). It is also a practice recommended by several international standards such as IEEE and ISO [6], [7]. By providing clear and explicit reasoning to demonstrate the property of interest, assurance cases simplify the tasks of reviewing the argument, as well as correcting it or completing it if necessary.

We argue that this approach is particularly suited for ensuring properties about an ML-based system for which the guarantees provided by conventional engineering practices are sometimes insufficient and often simply not applicable [8]. The approach is not specific to any particular industrial domain, but its application

¹<https://www.confiance.ai/en/>

clearly makes more sense in domains where significant stakes are involved. These stakes may relate to the impact on individuals (such as aeronautics, industry, or automotive), costs, or overall image (across all domains). **Our contributions are:** • A process and the associated tool that enable (i) building a generic argument to demonstrate that the system actually complies with some expected overarching properties thanks to appropriate development practices and/or V&V activities, (ii) formalising the relationship between this argument and the development artefacts of the system (engineering activities, engineering items and their properties). • A set of argumentation templates covering some major properties expected for a system involving AI. • A means to (i) derive a generic argument based on the contribution of each of its elements and (ii) produce the complete workflow including both the development and the V&V activities determined by the argument.

All these contributions are supported by a dedicated *tooling support* section which details how the method used is effectively implemented in our framework and how the tool can be used to reproduce each step or activity.

The remainder of this paper is structured as follows. Section II gives an overview of our approach which is then further decomposed into the 5 sections: Section III presents an extract of a development workflow and the associated engineering items of interest and Section IV identifies a set of properties for which guarantees are expected. Section V provides several generic assurance cases to ensure these properties and Section VI details how these assurance cases can be instantiated. The choices made during this phase lead to the selection of Development and Verification and V&V activities that must be integrated into the workflow and compiled into a V&V plan as presented in Section VII. Finally, Section VIII provides an overview of the related works, and Section IX concludes the document and opens some questions and future work.

II. APPROACH OVERVIEW

Our main objective is to exhibit the elements to be provided to demonstrate the satisfaction of a safety-related property of the system, considering the activities conducted to build it, and the artefacts produced in the process. The demonstration in this case is neither formal nor mathematical, but it shall nevertheless provide the argumentation and evidences necessary to convince – for instance, a regulation authority – that the property actually holds.

One necessary condition to support this objective is to define precisely the concepts involved in the

development, verification and validation of a system, and their relationships. Towards that goal, we rely on the Model-Based Systems Engineering (MBSE) approach where activities and Engineering Items are well defined (*i.e.*, they comply with a metamodel) and are associated with modelling artefacts.

Figure 1 presents the different steps that compose our approach to verifying the safety-related properties of the system. The numbering corresponds to the one used in the sections of this paper.

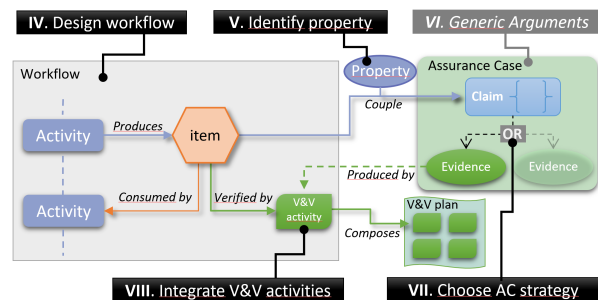


Fig. 1. General approach to demonstrate the satisfaction of safety-related properties

The starting point of this approach is the engineering workflow, from which we will extract the engineering items of interest. On each of these engineering items, we will then identify properties related to requirements on the system. The couple (item, property) will therefore constitute the root of an Assurance Case, claiming that the property is satisfied on this particular item. We will provide in Section V a set of generic Assurance Case patterns for specific ML-related properties. However, these generic patterns cannot be used as is. They may contain contexts and elements that need to be instantiated, typically with engineering items from the current system of interest. Besides, these argumentation trees may contain alternative branches in their decomposition, sometimes mutually exclusive, which implies that a choice has to be made in the argument by the user. This step is crucial as the resulting argument will dictate which Verification and Validation activities must be performed and integrated in the workflow. These new activities will finally be extracted in the form of a Verification and Validation plan which can be followed to ensure the expected property.

Tooling support details

Within the Confiance.ai program, the MBSE approach is implemented in the Capella² solution: an open source

²<https://mbse-capella.org/>

extensible Eclipse³ application dedicated to Systems Architecture modelling. The solution implements the Arcadia method [9] that promotes the use of dedicated modelling perspectives: “Operational Analysis”, “System Analysis”, “Logical Architecture”, “Physical Architecture”, “EPBS architecture”. The Confiance.ai program approach extends the “Operational Analysis” (OA) one, to provide a methodological end-to-end engineering approach to support the particularities of engineering activities related to critical AI-based systems [10]. It defines an “*Engineering Activities for trustable AI*” Capella Viewpoint, in which: Engineering Process, Process Sequence Flow, Process Activity, Process Item Flow, Engineering Role, Engineering Activity, and Engineering Exchange concepts, are mapped to Capella objects.

III. WORKFLOW DESIGN

The development *workflow*, which models the development activities and the corresponding Engineering Items (more specifically *Exchanged items* in Capella) they produce, is the starting point of our approach. Figure 2 is an excerpt of the comprehensive workflow for developing ML-based systems produced in the Confiance.AI program [11]. It presents a view of the activities and sub-activities commonly considered during the model engineering phase [12]. The ML model development is typically absent from conventional software workflow, as it is an ML-related activity. There are 3 possibilities to ensure a specific property on this activity:

- 1) Ensure that property holds for the item resulting from this activity (here the `trained ML model`),
- 2) Rely on a development activity which guarantees the property *by design* (or *by construction*),
- 3) Verify that the property holds on the item *before* the activity, and rely on guarantees that the activity preserves the property.

For this part of the workflow, we focus on the first two solutions, as the last solution implies to preserve a property during the training phase. This can be particularly challenging due to the absence of control over the optimisation process used during ML model training. Moreover, some of the properties we are interested in, such as robustness, are typically stemming from the training process itself, making it impossible to ensure any property preservation from a prior activity in this case.

³<https://www.eclipse.org/>

On the opposite, the first solution is typically based on additional verification and validation activities. Providing guarantees by design can also be effective for specific properties but is often insufficient and requires additional verification activities to obtain the appropriate level of guarantees, especially for critical systems.

In the following, we will therefore focus on the `Trained ML Model`, our engineering item of interest, and on the `Train ML Model` activity that produces it.

IV. PROPERTIES IDENTIFICATION

In a safety-critical context, one major concern is the system’s trustworthiness. Therefore, this paper focuses on trust-related properties that are specific to Machine Learning ([13]) and particularly difficult to verify. Historically, these properties are determined by the potential threats to which each activity is exposed. However, in the case of ML-based systems, the list of new threats is considerable [14], and highly dependent on the system of interest and the different activities involved in the development workflow. Therefore, in practice, we rely on specific regulation [15] or general guidance [16] to express these properties. Among the key aspects often considered in these documents, we selected *Robustness*, *Explainability* and *Fairness* as the three main properties to ensure on the `Trained ML Model` engineering item, as illustrated in Figure 3.

Furthermore, for these properties to be meaningful and carefully considered during the design of a system, they must be refined into low-level requirements that directly relate to the item of interest (the `Trained ML Model` in our case). For now, it is up to each industrial to interpret these guidelines and produce his own refinement of these properties in coherence with his use case. This may imply to consider the Operational Design Domain (ODD), which is a set of conditions in which the system is designed to operate [17], [18]. Indeed, some of these conditions can be propagated through the system down to the ML component and expressed as input constraints. Depending on the condition considered, this propagation may establish a link with a specific property like *robustness* or *fairness*. A typical example of operational conditions impacting the robustness of a trained ML model could be scenarios where a camera system is subject to vibrations, resulting in blurred images, or foggy environments leading to noisy images. These effects could then be translated into quantifiable metrics in the image space, for instance in maximum acceptable perturbations expressed with regard to the L_∞ -norm.

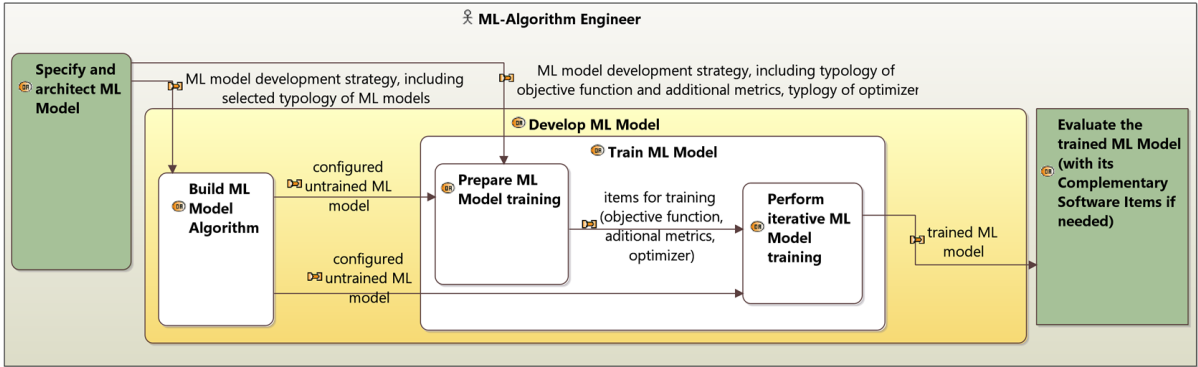


Fig. 2. Generic ML workflow focused on Model Engineering

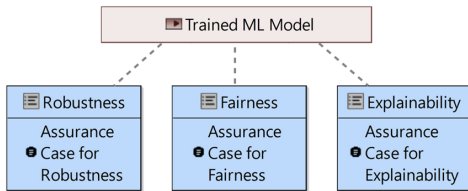


Fig. 3. Focus on the Trained ML Model Exchange Item in an extended Capella Class Diagram [CDB] with *Engineering Properties*

Although these analyses are typically application-dependant, we argue that, at least for the *robustness* property, a first level of refinement can be made generically, aligned with the capabilities of existing tools for verifying these properties or ensuring them by design.

Low-level refinements

For the *Robustness* property, we consider the following definition of *local robustness*: “A *Trained ML Model* is *locally robust* for a single input x to a *perturbation radius* λ if it produces the same output for any perturbation x' with $distance(x, x') \leq \lambda$ ”, where *distance* can be the l_2 -norm or the l_∞ -norm commonly used in that context.

Leveraging this definition which focuses on a single input, we can express global robustness criteria using three possible metrics:

- **Percent robust:** The percentage of samples that are locally robust for a fixed λ
- **Max robust:** The maximal λ for which all samples are locally robust
- **Mean robust:** The mean of the maximal λ for which each sample is locally robust

These three possible criteria expressed at the level of the ML component, coupled with our engineering item of interest –the *Trained ML Model*– form the root

goals of three different argumentation trees. This helps separate methods depending on whether they support the corresponding norms and metrics, although these root goals must still be instantiated with the appropriate λ and l -norm.

The *Robustness* property can be formalised using mathematically grounded concepts and formulas, which make it suitable for the refinement presented above. However, it is less direct for more softer properties. For instance, considering the main usages of *Explainability* in our case, we refined this property in two main requirements: providing explanations for successful model decisions or ensuring the absence of bias in the model’s decision-making process. These two aspects reflect the separation between local and global explanations, which consist of either explaining a single decision or explaining a set of decisions. Moreover, we further divided the local explanations into success-case and failure-case explanations. However, the primary usage of failure-case explanations is for the ML-Algorithm Engineer (see Figure 2) to find, during training, the reasons for a model failure, and to use these insights to retrain it and correct it. Conversely, the verification and validation of the *Trained ML Model* resulting from the application of the assurance case consider the model to be in a final, stable version. In that state, there should be no more failure cases to explain, but only success cases, which would still need those local explanations for increased trustworthiness. Therefore our main refinement for local explanations is expressed as “*The correct decisions of the model are explained*”.

Global explanations, on the other hand, serve to detect general biases in the model, which might reflect a problem during training or even an issue with the training dataset. Therefore we proposed “*The model is unbiased*” as a second refinement of the *Explainability*

in a separate Assurance Case.

Finally, the expression of the *Fairness* property was made more explicit with the following refinement: “The ML model does not contribute to any undesired discrimination”.

Tooling support details

In our approach, this decomposition of properties into low-level refinements is supported by a dedicated tool named *pure::variants*⁴ entirely integrated into the Capella environment (more details are provided in the following sections). This tool handles both the workflow model and the assurance case trees and will be used during several more or less complex steps of our approach. Notably, it offers a filtering functionality (configuration process of *pure::variants*) that extends up to the selection of the appropriate low-level requirements, as illustrated in Figure 4.

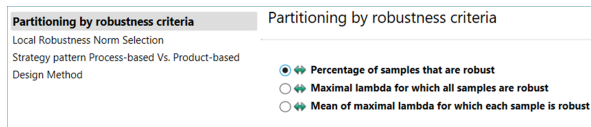


Fig. 4. Illustration of the filtering capability of *pure::variants*.

V. GENERIC ASSURANCE CASES

We provide in a git repository⁵ a set of generic assurance case patterns for the identified properties, namely *Robustness*, *Explainability* and *Fairness*, modelled in GSN. As explained in previous section, we refined these properties into more specific requirements, each resulting in a new Assurance Case. We present two extracts of these argumentation trees in Figures 5 and 6.

These argumentation trees are modelled in Capella using a dedicated *Assurance Case Viewpoint* supporting an enriched version of the GSN meta-model, and integrated in the Arcadia method.

The first extract in Figure 5 shows the decomposition of a robustness criterion according to several strategies, including the type of method used (*by design* or *by evaluation*) and the choice between these methods which is materialised in the tool by a diamond-shaped node. In these assurance cases, no choice has been made yet between the multiple branches, as these trees remain generic and thus can be considered and adapted to a variety of use cases.

⁴www.pure-systems.com/purevariants

⁵github.com/AssuranceCasesEC6/ML-based_Assurance_Cases

The second extract, presented in Figure 6 shows a generic argument for the local explainability of a *Trained ML model*. It focuses on the use of attribution methods for computer-vision tasks, ensuring that, for each decision, the part of the images used in each image is relevant. The main branch of this argument relies on the validation of the method results by experts whose legitimacy is verified in a specific subgoal. However, the AC also contains a specific branch dedicated to ensuring that the explainability method used is trustworthy, which will require additional V&V activities.

Using these generic assurance cases in our approach consists of instantiating the generic contexts and resolving the pending choices.

Tooling support details

As introduced before, the Capella environment is enhanced with an *Assurance Case Viewpoint*. It provides new Capella modeling elements, and new and enhanced diagrams at operational analysis level. The modeling elements added to support this Viewpoint are twofold:

- The concepts from the GSN standard in the version 3, cf. [19]. These are added to the “Operational Analysis” metamodel part and gathered into “Assurance case” elements in “Assurance Cases Pkg” ones. A specific GSN diagram implements the diagrammatic concrete syntax of the assurance case, composed of goals, strategies, solutions, etc. The relation with engineering operational activities is made via “Engineering item” and “Engineering item elements” referenced from GSN Solution elements.
- Some Glossary entries are also added to the “Operational Analysis” metamodel part and grouped into a “Glossaries pkg” element. Indeed, each GSN element is described (in rich text format) with hyperlinks to definitions in the glossary, or to external referential.

VI. AC INSTANTIATION AND CHOICES

Two tasks must be completed to instantiate an AC: The choices between branches must be made and the generic contexts must be instantiated. These tasks cannot be performed independently since, the selection of one branch might depend on the availability of the corresponding contexts, and on the opposite, only the instantiation of contexts of selected branches is mandatory to obtain a complete, *instantiated* AC.

An example of choice is provided on the extract of the *Robustness* AC in Figure 5. The cardinality of choice (here [1..4]) indicates that each method provided below in isolation is able to provide a certain degree

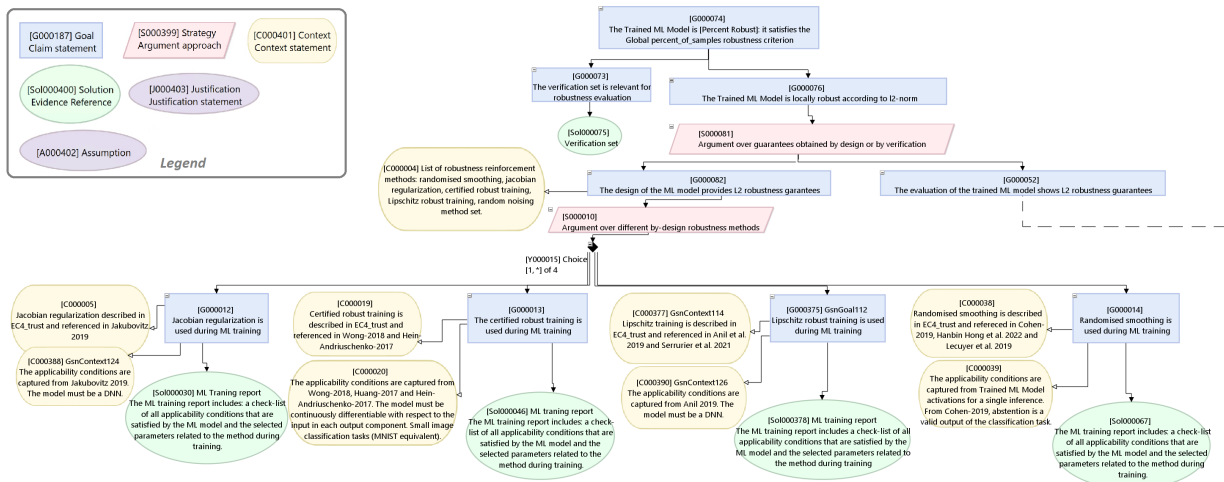


Fig. 5. Extract of the Assurance Case for the robustness of the Trained ML Model, considering a *Percent robust* metric with a l_2 -norm, with a focus on the methods providing guarantees *by-design*. The whole assurance case is provided on the following [git repository](#)

of guarantee for the l_2 -norm robustness. This choice can be based on several criteria, such as the criticality of the property of interest, the cost resulting from the deployment of a certain strategy/approach (i.e., time, effort, computational power, etc.) or other considerations related to the profile/experience of the engineer or its company. For instance, a critical property in a critical system will require the approach, or the combination of approaches, that brings the most guarantees to the satisfaction of the property of interest. On the other hand, numerous approaches [20], [21], [22] of confidence/uncertainty assessment of assurance cases, based on experts' judgements, can be used as a selection criterion. Indeed, a user may choose the strategy or the approach which provides the most (resp. the least) confidence (resp. uncertainty) to the satisfaction of its property.

In the generic arguments provided, nothing prevents the user from selecting several, if not all methods, regardless of their compatibility. Despite this issue, which needs to be verified on a case-by-case basis, it is even often recommended to use all possible methods available for a given choice to increase the overall confidence in the top-level argument. However, this will naturally come with increased costs, appearing when producing the corresponding V&V plan.

On the right of the *Explainability AC* extract in Figure 6, the context C000139 (in yellow) is an example of node requiring instantiation: The list of attribution methods to verify is not known in advance as it will be highly use-case dependant and may change

over time, which is why only a few suggestions can be provided. The context node C000176 also requires instantiation, as it is used in the attached goal as the set of "selected metrics" which must be evaluated for each explanation method considered.

These choices can be made in the *assurance case viewpoint* in Capella, where the user can either directly select the appropriate method for his use case, or use the tool *pure:variants* presented earlier, as illustrated in Figure 7.

The Assurance Case is considered *instantiated* when these two tasks are completed. However, this instantiation (especially the choices) structurally removes branches from the AC, leaving only the subset of V&V activities required to produce the expected pieces of evidence. Hence, the instantiation process impacts both the workflow and the AC. Depending on the structure of the workflow and the argumentation, and the potentially dual role of engineering items – i.e., playing a role in the argumentation and in the development workflow – this chain of effects can propagate back and forth between the workflow model and the argumentation model.

Tooling support details

In summary, GSN and Capella extended OA models are defined exhaustively, i.e. considering every possible goal, strategy and solution for any ML-based component and any property. However, an argumentation, in the end, has to be fit for a specific purpose. This means methodically removing specific model elements (On

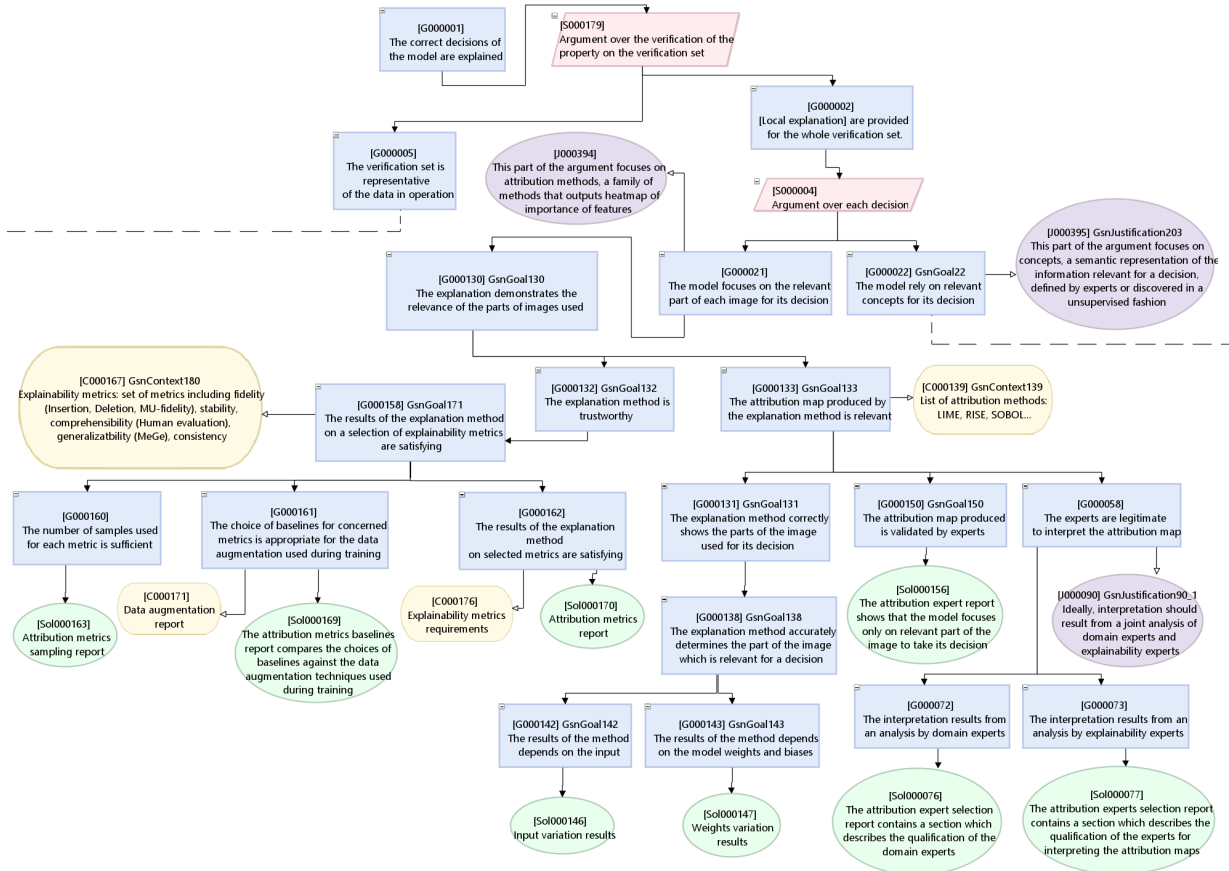


Fig. 6. Extract of the *Local Explainability Assurance Case* with a focus on the attribution branch. The whole assurance case is provided on the following [git repository](#)

Partitioning by robustness criteria	Design Method
Local Robustness Norm Selection	<input type="checkbox"/> Jacobian regularization
Strategy pattern Process-based Vs. Product-based	<input checked="" type="checkbox"/> Lipschitz training
Design Method	<input type="checkbox"/> Certified robust training
	<input type="checkbox"/> Randomised smoothing
	<input type="checkbox"/> Random noising

Fig. 7. Illustration of the functionalities of *pure::variants* for the selection of AC choices.

both Assurance case and Capella engineering parts) from the initial exhaustive models.

One community has already addressed these implementation problems for another intent: reuse and rationalisation of product families. Indeed, Software product line (SPL) [23] techniques aim at deriving a tailored product from a set of features [24]. The variability between these features is consistently managed, defining options, alternatives (AND, OR, XOR), and mandatory and exclusive features.

The implementation follows a so-called negative variability [25] (or annotative [26]) approach, which uses some form of explicit or implicit annotations in the models. This approach is here, as previously introduced, implemented with the use of a commercial SPL tool named *pure::variants*.

In our case, the exhaustive assurance case modelling represents a so-called 150% modeling [27] that acts as a base model, including all supported variability. Options or alternatives that are not selected for a specific argumentation during the configuration phase are then removed from the base models in the AC *instantiation* phase.

The implementation is done as depicted in Figure 8:

- (A) In the feature models (.xfm files) are encoded the valid possible configurations of the tactics that are applicable to a given property. An example for the robustness AC is given in Fig. 9. The feature model defines first-order logical constraints

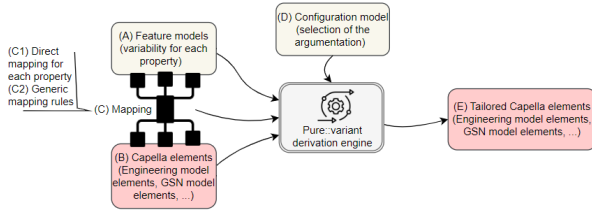


Fig. 8. Pure::variant implementation and workflow overview

between features with operators (e.g. the AND operator between “Local Robustness” and applying a “strategy pattern” as represented in the Figure), and direct constraints (e.g. selecting “Property satisfied by design” requires to apply a “Design Method”). It gathered all possible choices available in the configuration, from the AC selection property to consider (Non-exclusive OR between “Robustness”, “Fairness”, “Explainability”), to the partitioning choices (cf. 4), to the different strategy choices into the GSN model (i.e. the cardinality of choice in the GSN model, e.g. [1..4] in Figure 5 has to be consistent with the one in the feature model – Non-exclusive OR with cardinality constraints).

- (B) In the Capella model is represented exhaustively (150%) the modelling element (engineering and assurance cases in GSN).
- (C) The mapping between the external feature model is performed at two levels: (C1) a specific mapping between one feature model and the Capella model (.ccfm file); (C2) generic deletion (propagation) rules. Indeed, in the extended Capella, different element types are semantically related to each other. Propagation rules utilise these semantic relations to simplify the mapping of Capella elements to conditions. Basically, a propagation rule ensures that if Element A is removed during transformation, also Elements B, C, and D are removed. We defined 16 customised rules.
- (D) The configuration model drives the possible choices according to the feature model logic (.vdm file). It also stores the final configuration. This configuration is graphically customised via the use of a configuration wizard model (.vcwm file) to provide a graphical interface as illustrated in Fig. 4 or Fig. 7.

With the information given in all these files, the *pure::variants* derivation engine produces a new Capella project containing only the instantiated AC, as

represented in Figure 8.

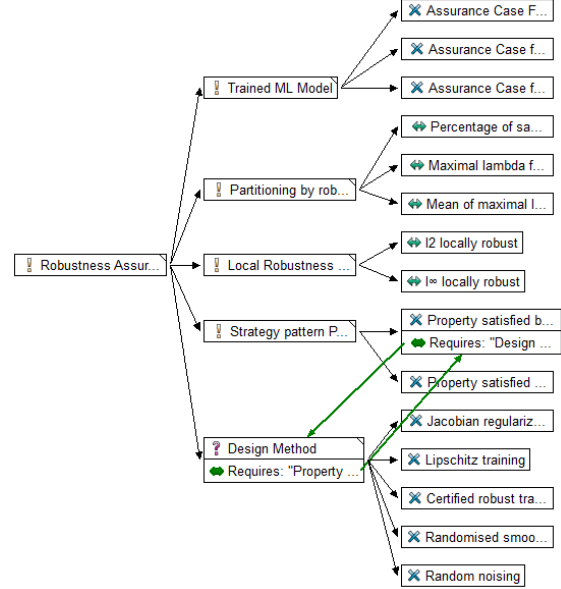


Fig. 9. Extract of the *Pure::variants* feature model on the robustness AC

VII. V&V ACTIVITIES INTEGRATION

The instantiated AC provides a list of evidences that must be produced through specific V&V activities which must therefore be added to the development workflow⁶. These activities produce the required evidence as V&V Engineering Items, and they should take as input either development Engineering Items or other V&V Engineering Items. At this stage, the **Workflow** contains both the development activities and the V&V activities, as illustrated in Figure 10.

The AC serves as an intermediate yet important model for building a comprehensive and convincing V&V plan⁷, by combining the new V&V activities needed to produce the evidences required. This plan can thus be automatically generated from information related to the engineering item, the property to be verified and all other elements carried by the argument. It also includes other information, such as the glossary, which defines all key terms used in the argumentation (e.g., local robustness, l_2 local robustness, etc.). Only the part of the V&V plan dealing with specific solutions which require

⁶In practice, the V&V activities that produce the evidences required by all the branches of the AC are added a priori to the workflow and simply selected depending on the choice made in the AC

⁷Please note that we use the term V&V plan in a broader sense as it may include both design elements and V&V activities

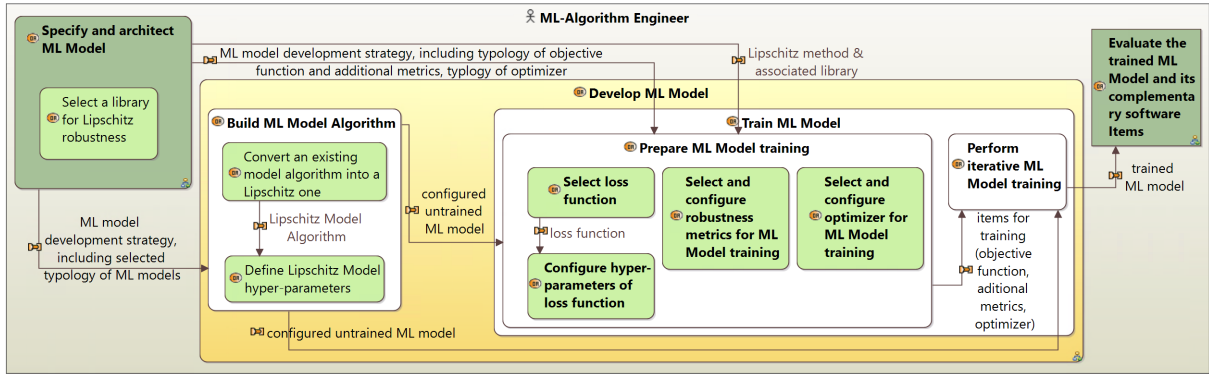


Fig. 10. Example of enrichment of the generic workflow of Figure 2 with the new activities (in light green) required by a specific branch of the robustness assurance case (robustness by design using Lipschitz training).

precise knowledge of the methods to be used and how to implement them needs to be manually produced by the experts of the method. Figure 11 shows an extract from such a plan, related to the robustness property, for the implementation of a Lipschitz network.

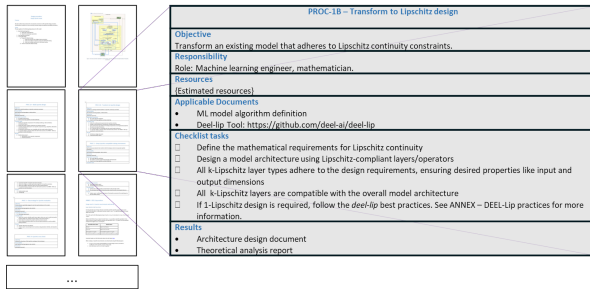


Fig. 11. Extract from the V&V plan related to robustness property

VIII. RELATED WORKS

This section focuses on existing works that applied ACs to ML-based systems. Indeed, as mentioned previously, Assurance Cases appear as relevant tools for addressing certification-related issues and ensuring systems safety [28]. Applying this approach to ML-based systems seems like an appropriate solution, as no consensual method exists in this domain and standards are not yet established [29].

A first step in this direction was made by Hawkins *et al.* with AMLAS [30], a general guidance on the use of ACs for ML-based system. Their analysis covers a generic ML development workflow, which they enriched with high-level ACs at each main step of the development.

While this paper is of major interest for the application of ACs to an ML-based workflow, the arguments provided rely heavily on requirements *to be defined* and contain numerous elements (variables) that need *to be instantiated*. This approach is highly generic and handles any type of workflow, as the AC's solutions are mainly assessments of the satisfaction of the requirements. Yet, for this reason, the decomposition of main goals remains limited to a few steps, since refining argumentation until concrete evidence needs requirements to be defined and variables to be instantiated. Nevertheless, this paper provides a solid foundation for developing ACs further. In particular, this approach is complementary to ours: it could be used as a common high-level argument for linking all the main properties of interest of ML-based systems such as robustness, fairness or explainability presented in our work. This would provide additional steps of decomposition in the assurance case before reaching the step where the requirements need to be instantiated.

In addition to this general guidance, a few recent studies provide argumentation patterns applicable to ML-based systems. These generic solutions to common problems are intended to be reused and instantiated in various contexts for similar problems. However, they often highlight the fact that a concrete operational context or risk analysis is needed in order to go down in the argumentation until reaching final evidence.

For instance, the work of Picardi *et al.* [31] defines ACs for the deployment of ML-based systems in a medical context. Their approach focuses on the interpretability of the system's outputs, going beyond conventional performance measurements. They propose an argumentation pattern that encompasses the entire context of the ML component, from datasets to model

architecture. This pattern also covers domain-specific contexts, such as information about experts involved and the technical tasks they realise. In [32], they refine their generic pattern with a more precise taxonomy and focus on providing additional guarantees on the confidence in the Trained ML Model (called “Machine Learning Learnt” in their paper) and on the confidence in the data. However, these new arguments are composed of a single decomposition step and thus remain particularly generic and high-level. Finally, the authors extend their work in [33] to reflect better the relationship between the ML models and the safety of the system. In the process, they also develop an assurance process for the engineering of ML components built upon existing best practices. This process facilitates the instantiation of the confidence argument patterns through consideration of the required activities to be undertaken and the artefacts to be generated at each stage in the ML lifecycle. They also propose a generic way of decomposing ML requirements (“desideratas”) into property-specific arguments such as model performance, model robustness and model interpretability. However, these properties are only decomposed one step further, and their subgoals remain “to be developed”, which contrasts with our approach that proposes a multi-step decomposition down to concrete evidence.

Although previous literature provides a significant foundation in the development of assurance cases for ML-based systems, its traceability with respect to the ML development workflow is still exploratory, and none of them provide the associated tooling support.

In the automotive domain, Bloomfield *et al.* [34] propose an AC template for an experimental autonomous vehicle and its social context. The decomposition of their argumentation follows a top-down approach, from the system to its components, including the ML model. To ensure the trustworthiness of their system, their systematic approach explicitly considers sources of doubt and vulnerabilities in the system’s behaviour. To this aim, their pattern is designed to identify gaps and challenges during the justification of system behaviours, as well as gaps within the assurance framework itself. This approach is complementary to ours and could be used to identify new gaps and properties on specific engineering items of the complete workflow of an ML-based system, providing a starting point for developing new arguments.

Among the most recent studies, Dong *et al.* [35] present a specific ‘end-to-end’ AC applied to an ML model. First, they introduce a framework called the Reliability Assessment Model (RAM) that assesses the reliability of a classification model, covering both its

robustness and its operational profile. Then, all evidence produced by the RAM is represented with an AC that tackles the argumentation from a probabilistic point of view and ends with quantitative evidence. This rigorous approach provides a complete, vertical argumentation for an ML property. While we share the authors’ argumentation approach, our work differs in two key aspects. First, we do not focus on a specific property including all the mathematical formalism. Our focus is set on an end-to-end assurance case with a model-based formalism. In other words, an assurance case whose evidence and contextual information are mapped onto the engineering workflow. Each necessary element of the assurance case considers a pre-established workflow of activities alongside the injection of V&V activities to be performed. Second, our proposal develops assurance cases horizontally, while they include the selection of multiple sub-arguments for the user, which allows them to select different demonstration approaches.

IX. CONCLUSION

Throughout this paper, we presented a tool-supported process that enables the systematic derivation of V&V plans, specifically tailored to address the unique challenges posed by the introduction of ML-based systems in critical domains. We addressed these challenges across the entire lifecycle of AI-based systems, integrating our approach into an engineering workflow specific to such systems. This workflow constitutes a pivotal resource for identifying relevant properties and integrating the V&V activities required by our assurance case patterns. Indeed, our contributions extend beyond the mere formulation of a process and offer a set of argumentation templates, focusing on key properties of these systems, such as *robustness*, *explainability* and *fairness*. Nevertheless, combining these arguments under the same overarching property is an open problem. A first step in this direction is provided with the AMLAS [30], which can be used as a common high-level argument, but conflicts between branches may still arise. However, while additional research is required to explore this subject, our tooling and the choice mechanic offer a partial solution to this problem since the branches are structurally removed from the argument, and conflicts will thus appear in the form of non-supported goals during the instantiation of the assurance case.

Acknowledgments

This work has been supported by the French government under the “France 2030” program, as part of the SystemX Technological Research Institute.

REFERENCES

- [1] National Research Council et al. *Software for dependable systems: Sufficient evidence?* National Academies Press, 2007.
- [2] Tim Kelly and Rob Weaver. The goal structuring notation—a safety argument notation. In *Proceedings of the dependable systems and networks 2004 workshop on assurance cases*, volume 6. Citeseer Princeton, NJ, 2004.
- [3] S Gan and JA Ryan. Claims arguments evidence. www.adelard.com/asce/, 2019.
- [4] Object Management Group (OMG). Structured assurance case metamodel (sacm). www.omg.org/spec/SACM/, 2013.
- [5] George M Cleland, Mark-Alexander Sujan, Ibrahim Habli, and John Medhurst. *Evidence: using safety cases in industry and healthcare*. The Health Foundation, 2012.
- [6] Systems and software engineering - Systems and software assurance - Part 2: Assurance case. Standard, International Organization for Standardization, Geneva, CH, 2022.
- [7] R. Palin, D. Ward, I. Habli, and R. Rivett. ISO 26262 safety cases: compliance and assurance. In *6th IET International Conference on System Safety 2011*, pages B12–B12, Birmingham, UK, 2011. IET.
- [8] Rasmus Adler and Michael Klaes. Assurance cases as foundation stone for auditing ai-enabled and autonomous systems: Workshop results and political recommendations for action from the examai project, 2022.
- [9] Pascal Roques. MBSE with the ARCADIA Method and the Capella Tool. In *8th European Congress on Embedded Real Time Software and Systems (ERTS 2016)*, Toulouse, France, January 2016.
- [10] Afef Awadid., Boris Robert., and Benoît Langlois. Mbse to support engineering of trustworthy ai-based critical systems. In *Proceedings of the 12th International Conference on Model-Based Software and Systems Engineering - MODELSWARD*, pages 280–287. INSTICC, SciTePress, 2024.
- [11] Juliette Mattioli, Agnes Delaborde, Souhaïel Khalifaoui, Freddy Lecue, Henri Sohier, and Frédéric Jurie. Empowering the trustworthiness of ml-based critical systems through engineering activities. *arXiv preprint arXiv:2209.15438*, 2022.
- [12] Rob Ashmore, Radu Calinescu, and Colin Paterson. Assuring the machine learning lifecycle: Desiderata, methods, and challenges. *ACM Computing Surveys (CSUR)*, 54(5):1–39, 2021.
- [13] Wulf Loh, Andreas Hauschke, Michael Puntschuh, and Sebastian Hallensleben. Vde spec 90012 v1.0 - vcio based description of systems for ai trustworthiness characterisation. 04 2022.
- [14] B Caroline, B Christian, B Stephan, B Luis, D Giuseppe, E Damiani, H Sven, L Caroline, M Jochen, Duy Cu Nguyen, et al. Securing machine learning algorithms. 2021.
- [15] European Commission. Laying down harmonised rules on artificial intelligence (artificial intelligence act) and amending certain union legislative acts, 2021.
- [16] EAT Force and A Daedalean. Concepts of design assurance for neural networks (codann) ii. In *Concepts of Design Assurance for Neural Networks (CoDANN)*. EASA, 2021.
- [17] Philip Koopman and Frank Fratrick. How many operational design domains, objects, and events? *Safeai@ aai*, 4, 2019.
- [18] Krzysztof Czarnecki. Operational design domain for automated driving systems. *Taxonomy of Basic Terms “, Waterloo Intelligent Systems Engineering (WISE) Lab, University of Waterloo, Canada*, 2018.
- [19] The Assurance Case Working Group (ACWG). Gsn community standard v3, May 2021. Accessed on March. 21, 2024.
- [20] Ewen Denney, Ganesh Pai, and Ibrahim Habli. Towards measurement of confidence in safety cases. In *2011 International Symposium on Empirical Software Engineering and Measurement*, pages 380–383. IEEE, 2011.
- [21] Rui Wang, Jérémie Guiochet, Gilles Motet, and Walter Schön. Safety case confidence propagation based on dempster–shafer theory. *International Journal of Approximate Reasoning*, 107:46–64, 2019.
- [22] Yassir Idmessaoud, Didier Dubois, and Jérémie Guiochet. Confidence assessment in safety argument structure-quantitative vs. qualitative approaches. *International Journal of Approximate Reasoning*, 165:109100, 2024.
- [23] Klaus Pohl, Günter Böckle, and Frank J. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer, 1 edition, 2005.
- [24] Kyo Kang, Sholom Cohen, James Hess, William Novak, and A. Peterson. Feature-oriented domain analysis (foda) feasibility study. 01 1990.
- [25] Markus Voelter and Iris Groher. Product line implementation using aspect-oriented and model-driven software development. In *11th International Software Product Line Conference (SPLC 2007)*, pages 233–242, 2007.
- [26] Christian Kästner, Sven Apel, and Martin Kuhlemann. Granularity in software product lines. In *Proceedings of the 30th International Conference on Software Engineering, ICSE '08*, page 311–320, New York, NY, USA, 2008. Association for Computing Machinery.
- [27] Krzysztof Czarnecki and Michał Antkiewicz. Mapping features to models: a template approach based on superimposed variants. In *Proceedings of the 4th International Conference on Generative Programming and Component Engineering, GPCE'05*, page 422–437, Berlin, Heidelberg, 2005. Springer-Verlag.
- [28] John Rushby. The interpretation and evaluation of assurance cases. *Comp. Science Laboratory, SRI International, Tech. Rep. SRI-CSL-15-01*, 2015.
- [29] Fateh et al. Kaakai. Toward a machine learning development lifecycle for product certification and approval in aviation. *SAE International Journal of Aerospace*, 15(01-15-02-0009), 2022.
- [30] Richard Hawkins, Colin Paterson, Chiara Picardi, Yan Jia, Radu Calinescu, and Ibrahim Habli. Guidance on the assurance of machine learning in autonomous systems (AMLAS). *CoRR*, abs/2102.01564, 2021.
- [31] Chiara Picardi and Ibrahim Habli. Perspectives on Assurance Case Development for Retinal Disease Diagnosis Using Deep Learning. In David Riaño, Szymon Wilk, and Annette ten Teije, editors, *Artificial Intelligence in Medicine*, pages 365–370, Cham, 2019. Springer International Publishing.
- [32] Chiara Picardi, Richard David Hawkins, Colin Paterson, and Ibrahim Habli. A pattern for arguing the assurance of machine learning in medical diagnosis systems. In *38th International Conference on Computer Safety, Reliability and Security – SafeComp 2019*, April 2019.
- [33] Chiara Picardi, Colin Paterson, Richard Hawkins, Radu Calinescu, and Ibrahim Habli. Assurance argument patterns and processes for machine learning in safety-related systems. In *Proceedings of the Workshop on Artificial Intelligence Safety, NY, USA, 2020*, volume 2560 of *CEUR Workshop Proceedings*, pages 23–30. CEUR-WS.org, 2020.
- [34] R. Bloomfield, H. Khlaaf, P. Ryan Conmy, and G. Fletcher. Disruptive innovations and disruptive assurance: Assuring machine learning and autonomy. *Computer*, 52(9):82–89, 2019.
- [35] Yi Dong, Wei Huang, Vibhav Bharti, Victoria Cox, Alec Banks, Sen Wang, Xingyu Zhao, Sven Schewe, and Xiaowei Huang. Reliability Assessment and Safety Arguments for Machine Learning Components in System Assurance. *ACM Transactions on Embedded Computing Systems*, 22(3), May 2023. arXiv:2112.00646 [cs].

Uncertainty in Assurance Case Pattern for Machine Learning

Yassir Id Messaoud
IRT SystemX
Palaiseau, France
0009-0000-5673-0844

Jean-Loup Farges
IRT SystemX and ONERA
Palaiseau and Toulouse, France
0000-0002-0737-0640

Eric Jenn
IRT Saint Exupéry
Toulouse, France
0000-0001-9699-3497

Vincent Mussot
IRT SystemX and IRT Saint Exupéry
Palaiseau and Toulouse, France
0009-0001-8819-3163

Anthony Fernandes Pires
IRT SystemX and ONERA
Palaiseau and Toulouse, France
0000-0003-0522-3898

Florent Chenevier
IRT SystemX and Thales AVS
Palaiseau and Toulouse, France
florent.chenevier@irt-systemx.fr

Ramon Conejo Laguna
IRT SystemX and IRT Saint Exupéry
Palaiseau and Toulouse, France
0009-0007-9787-1716

Abstract—Assurance case (AC) patterns are structured arguments in a tree-like form in which certain choices are not frozen. By making these choices a user can determine a design, implementation, integration, verification and validation workflow that will produce artifacts supporting the argument for his/her use case. However, it is difficult to make choices in an AC pattern because of the lack of information on the consequences of these choices and the cost/effort they may require. Based on recently published results, this work proposes an uncertainty assessment that allows the user to be aware of the confidence in the argument induced by those choices. To do so, confidence features are elicited from experts. The elicitation procedure is presented and the propagation of uncertainty through the AC is analyzed. Finally, application of the method on a use case related to robustness of machine learning models demonstrates the validity of the approach.

Index Terms—Assurance case, Dempster-Shafer theory, robustness, machine learning, experts' judgments elicitation

I. INTRODUCTION

Functions designed using Machine Learning (ML) have to comply with standards and nowadays an effort is devoted to the proof of their dependability. Justification of such high-level properties can be done with structured arguments named Assurance Cases (AC). In order to streamline and normalize the design of AC, AC patterns are proposed. The objective of the research presented here is to add uncertainty or confidence to AC patterns. The final objective of uncertainty assessment in instances of AC is to provide to certification authorities an AC presenting a full belief assessment. However, intermediate steps with intermediate objectives are necessary because the product to be certified follows a design, implementation, integration, verification and validation cycle. At the beginning of the cycle, the product owner only relies, for all cycle steps, on an AC pattern that provides choices in a pre-defined tree structure. The difficulty for making decisions among choices is high when the subject of the AC is a new technology with a large number of approaches with different levels of readiness, as it is the case for robust ML. In those cases an uncertainty assessment can be useful for making a judgment about the

opportunity of using a specific approach. Moreover, the uncertainty assessment of each strategy in the tree structure may be performed at no cost and could be directly provided with the AC pattern. At the opposite, the evidence at some leaves of the tree is subject to dynamical uncertainty assessment: The evidence will be provided at no additional cost by the chosen design process but the uncertainty before producing it may be different from the uncertainty after producing it and depending on the choice made in the AC pattern, the evidence must be provided independently from the design process by the verification and validation process with some cost.

The objective of this research raises several issues: Choice of an uncertainty representation, elicitation of uncertainty associated to atomic elements such as relations and evidences, and propagation of the uncertainty of atomic elements through the AC. Working with AC patterns that will become instantiated as actual AC is also quite challenging.

The approach followed here is based on recently published results [1], [2] and brings the following contributions:

- 1) An uncertainty assessment based simultaneously on qualitative and quantitative uncertainty modeling,
- 2) an elicitation method allowing simultaneous capture of qualitative and quantitative uncertainty,
- 3) an analysis of uncertainty modeling and propagation on AC patterns and
- 4) demonstration of the approach with a use case related to robustness of ML models.

The following section is devoted to positioning the approach described above with respect to the state of the art. Then, a section presents the uncertainty assessment. Section IV is devoted to the elicitation process. Modeling and propagation are analyzed in section V. Section VI demonstrates the approach on the use case. Finally the conclusion provides a global assessment of the approach and possible improvements.

II. BACKGROUND AND RELATED WORKS

In this section, we introduce the necessary background information to facilitate a comprehensive understanding of our

work and we highlight the weaknesses of related works.

A. Structured arguments

1) *Formalism*: Goal Structuring Notation (GSN) [3] is a graphical way to describe AC including concepts such as *Goal*, *Solution*, *Strategy*, *Context*, *Assumption*, *Justification* and their relationships such as *Is supported by* and *In the context of*. Figure 1 illustrates some of these elements. Further versions of GSN include an extension allowing the description of argument patterns using the concepts of *Choice* and *Uninstantiated Element* and the description of confidence argument using the concept of *Assurance Claim Point* that refers to another argument for assessing the confidence [4]. The work presented here was conducted in the scope of GSN using another method to assess confidence. An alternative graphical way of describing AC is Claim Argument Evidence (CAE) [5]. More recently, Structured Assurance Case Meta-model (SACM) [6] was built upon GSN and CAE and transformations from these models to SACM were developed. SACM allows arguing the confidence in the arguments provided in the AC by using a meta-claim feature of the Assertion element. Meta-claim as its name suggests, is a Claim about an Assertion to argue the trustworthiness of the Assertion. The approach presented here is quite different from the SACM approach because here confidence is not modeled by additional claims but is grounded on uncertainty measures. Nevertheless, using the transformation GSN to SACM the results obtained here could be used in SACM.

2) *AC for machine learning*: Safety criteria, which if enforced would contribute to justifying the safety of neural networks, were determined and structured in an AC pattern presenting an undeveloped goal “The neural network tolerates faults in its inputs” [7]. The AC pattern for robustness of ML used in our work corresponds to a development of this goal.

Burton and Herd proposed a high level AC pattern for claiming that the ML system satisfy its allocated safety requirements within the defined context [8]. A strategy refines this goal in five sub-goals concerning specification, data sets, design, demonstration and operation. Only the sub-goal concerning specifications is detailed to the level of solutions. The AC pattern used in our work addresses design and demonstration and is detailed to the level of solutions on the design part.

B. Uncertainty modeling

Uncertainty is most of the time modeled using probabilities. Those are most of the time related to frequencies and are more suited for aleatory uncertainty than for epistemic uncertainty. T-norms and T-conorms are binary operations which generalize respectively conjunction and disjunction in valued logic [9]. The probabilistic T-norm corresponds to the product while the Zadeh’s T-norm correspond to the minimum. Their associated T-conorms are the sum minus the product and the maximum. If $T(x, y)$ and $T^*(x, y)$ are a T-norm and its conorm, the distributivity property is characterized by $T(x, T^*(y, z)) = T^*(T(x, y), T(x, z))$ and $T^*(x, T(y, z)) = T(T^*(x, y), T^*(x, z))$ and the absorption

property by $T(T^*(x, y), x) = x$ and $T^*(T(x, y), x) = x$. Finally the idempotency property is characterized by $T(x, x) = x$ and $T^*(x, x) = x$.

The Dempster-Shafer Theory (DST) [10] is a general framework for reasoning with uncertainty. It uses a frame of discernment and may allocate parts of an unitary mass on all non empty subsets of this frame of discernment. DST operations are extension to the cross product of frames of discernment, combination of masses from different sources managing the conflict issue and marginalization. Capacities [11], are set functions which give 0 for the empty set, 1 for the sure event and respect monotonicity with respect to inclusion.

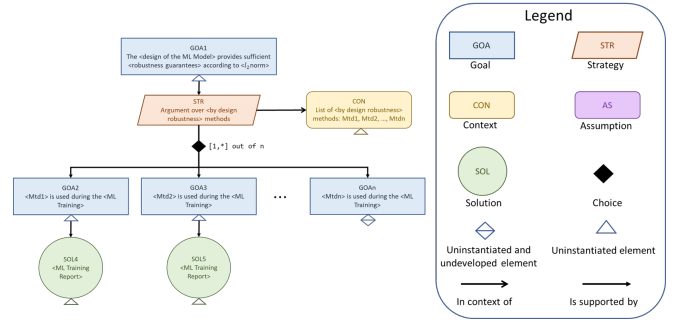


Fig. 1. An example of parts of a GSN pattern (Extract from Robustness AC pattern)

C. Uncertainty assessment in AC

1) *Probabilistic approach*: The question of uncertainty assessment in ACs has been the subject of a number of approaches. Some are based on *probability theory*. They use *Bayesian Networks* (BN) [12]–[15] to propagate probabilities on pieces of evidence provided by the argument up to the top-goal. Probabilities deals well with aleatory uncertainty. However, this is less the case for epistemic uncertainties due to lack of information. For this reason the work presented here is not based on BN.

2) *Approaches based on DST theory*: To address the issue related to BN, other approaches using DST are proposed. In addition to efficiently modeling epistemic uncertainties, these kinds of approach require less data than Bayesian approaches. First of all, those approaches assume that uncertainty is associated on the one hand to goals directly linked to solutions and on the other hand to the support relation between goals, either directly or through an explicit strategy. The other elements of GSN, such as context, assumption, justification and in the context of, provide information about uncertainty but don’t carry this information. For instance, Wang et al. [16] use DST to propose models to elicit confidence values about evidence and propagate them according to the relationships between a goal and its sub-goals. The confidence on these relations is also quantified using DST. To determine their values, Wang et al. proposed to use the non linear least square method. However, this method can lead to values outside the unit interval [0,1]

which makes no sense. Chung-Ling et al. [17] propose to use Vector Space Model (VSM) to identify these values.

Authors in [1], [18], [19] used an approach based on experts judgment to deal with this issue. They assume that *Goals* directly supported by a *Solution* can be believed, disbelieved and epistemically uncertain, rules involved in Strategy can be believed, epistemically uncertain but cannot be disbelieved. Considered rules, $p_i \Rightarrow C$, $\neg p_i \Rightarrow \neg C$, $(\wedge_i p_i) \Rightarrow C$ and $(\wedge_i \neg p_i) \Rightarrow \neg C$, with p_i a child goal and C a father goal, provide a formal and flexible definition of Is supported by. Their corresponding belief are noted here reciprocally B_{\Rightarrow}^i , B_{\Leftarrow}^i , B_{\Rightarrow} and B_{\Leftarrow} . Two approaches to uncertainty assessment of GSN are proposed: the quantitative approach and the qualitative approach. For the quantitative approach elicitation is performed using scales and the rankings are transformed in numbers. For the propagation child Goals with masses B_p^i , D_p^i and $1 - B_p^i - D_p^i$ on respectively itself p_i , its negation $\neg p_i$ and tautology $\top = p_i \vee \neg p_i$ lead to conclusion Goals with masses on C , $\neg C$ and $\top = C \vee \neg C$. If masses on goals linked to solutions are provided, the mass computation can be propagated from the bottom of the tree to the top of the tree and provides belief, B_C and disbelief D_C in top claim. Formulae for numeric propagation are derived from the hypotheses and the DST:

$$B_C = B_{\Rightarrow} \cdot \prod_i B_p^i (1 - B_{\Rightarrow}^i) + 1 - \prod_i (1 - B_p^i \cdot B_{\Rightarrow}^i) - M_C \quad (1)$$

$$D_C = B_{\Leftarrow} \cdot \prod_i D_p^i (1 - B_{\Leftarrow}^i) + 1 - \prod_i (1 - D_p^i \cdot B_{\Leftarrow}^i) - M_C \quad (2)$$

where M_C is the conflict mass on C . For its computation see [19]. For the qualitative approach elicitation is also performed using scales but there is no need to transform rankings in numbers. Formulae for qualitative propagation are derived from the hypotheses, the DST and the properties of capacities:

$$\beta_C = \max\{\min(\beta_{\Rightarrow}, \min_i \beta_p^i), \max_i \min(\beta_p^i, \beta_{\Rightarrow}^i)\} \quad (3)$$

$$\delta_C = \max\{\min(\beta_{\Leftarrow}, \min_i \delta_p^i), \max_i \min(\delta_p^i, \beta_{\Leftarrow}^i)\} \quad (4)$$

where β and δ are reciprocally the qualitative counterparts of B and D .

3) *Criticism to propagation of uncertainty in AC*: Burton and Herd indicate that these approaches depend on the availability of reliable confidence values that can be assigned to elements of the assurance argument and combined into an overall confidence score, they are themselves subject to uncertainty and subjective judgment [8]. In order to avoid this problem they propose to use locally, i.e. for each element of the AC, a first scale of uncertainty including subjective ranking, subjective probabilities, probabilities and variance combined by a second scale including ignorance, imprecise judgment, precise judgment and certainty. Those scales are quite helpful for improving locally an AC but seem inoperative for making choices in an AC pattern.

III. QUALITATIVE AND QUANTITATIVE MODELING SHALL BE CONSIDERED TOGETHER

Requirements are proposed for uncertainty modeling and assessment: (i) The assessment shall be useful for focusing validation effort and for identifying weaknesses of AC structure, (ii) the result of the assessment of an AC tree shall not be driven by its dimension, (iii) the sensitivity of the assessment shall allow discriminating strategies and (iv) methodological choices should not be arbitrary.

A. Usefulness

The uncertainty assessment is useful for focusing validation effort on most sensitive parts of the AC because it is performed at each goal and can indicate its weakness and contradictions between proof elements. For nodes corresponding to conjunctions a procedure to focus on the most sensitive element, i.e., the one with least belief is derived. If this element corresponds to a Solution, consider means for improving its belief, for example, doing a higher number of tests. The uncertainty assessment is also useful for identifying weaknesses of AC structure and applying uncertainty reduction techniques. The proposed procedure is quite like the one for focusing validation. A Strategy associated to a node, whose uncertainty is sensitive but whose uncertainties of the children are not so, is not sufficiently convincing. Then, an alternative strategy can be considered.

B. Dimension

The result of the analysis of this requirement on a large conjunctive argument case indicates that, for the numeric approach, while the number of solutions increase the general trend is the rejection of the property corresponding to the root goal. At the opposite, for the qualitative approach the belief of the root goal cannot be lower than the belief of the solution with the lowest belief. Moreover, the disbelief of the root goal cannot be larger than the disbelief of the solution with the largest disbelief. With the qualitative approach the uncertainty of the root goal is bounded.

C. Sensitivity

Changing a strategy changes the goal supported by this strategy. This goal supports its father goal. Thus, changing a strategy changes a premise of a goal. For the numeric approach, partial derivatives of the belief and disbelief of the father goal with respect to belief and disbelief of a premise are highlighted. Thus there is a sensitivity to each premise. Concerning the qualitative approach, sensitivity of goal belief to belief of premise argmin and sensitivity of goal disbelief to disbelief of premise argmax are highlighted. However, those sensitivities are valid only when argmin respectively argmax are single premise. Finally, there is no sensibility to other premises.

TABLE I
COMPLIANCE OF UNCERTAINTY MODELING WITH REQUIREMENTS

Requirement	Numeric	Qualitative
Usefulness	+	+
Result not dimension driven	-	++
Sensitivity	++	-
Not arbitrary methodological choices	+	+

D. Methodological choices

The T-norm used in the numeric approach can only be applied to numbers and is grounded on: assimilating the uncertainty measure to frequencies, representativeness of frequencies and independence of events. The T-norm used in the qualitative approach can be applied on numbers as well as on ordered linguistic qualifiers and is the unique T-norm complying with idempotence, absorption and distributivity. Concerning the assessment of elementary elements, the consensus on the association of a number with a linguistic qualifier is difficult. The numeric approach highlights slight differences between belief degrees. However, it is unlikely that two experts provide the same value. The scale used by the qualitative approach is associated to linguistic qualifiers, there is consensus on their order and it is likely that two experts associate the same qualifier to the same element. However, there is gaps between the degrees of the scale and results on an extreme case highlight the negative effect of a limited number of linguistic qualifiers on sensitivity. Indeed, improving the AC implies substituting several elements in a single step.

E. Synthesis

Table I, presents a synthesis of the compliance of uncertainty assessment methods with requirements. It indicates that in order to comply with all requirements it is needed to work with both a scale and numbers and use the numeric and qualitative methods together.

IV. ELICITING QUALITATIVE AND QUANTITATIVE UNCERTAINTY IN A SINGLE STEP

Another important result of the work is the definition of a methodology for elicitation of uncertainty associated to rules and Goals directly linked to Solutions. Following this methodology, the full tree is presented without Strategies to experts, i.e. child Goals are directly connected to father Goals by a Is supported by relation. Then a questionnaire with a form for each hidden Strategy has to be filled by experts. Figure 3 presents an extract of the form for a Goal supported by two children Goals. In those forms, the number of questions per hidden Strategy is equal to the number of rules, i.e. two plus twice the number of child Goals.

Answers are given by experts associating a confidence in decision on the scale { very low, low, high, very high }. For positive rules the provided decision is the acceptance of the father Goal. For negative rules it is the rejection of the father Goal. In both cases the strength of the decision is scaled on {no decision, weak, moderate, strong }. Numerical values are

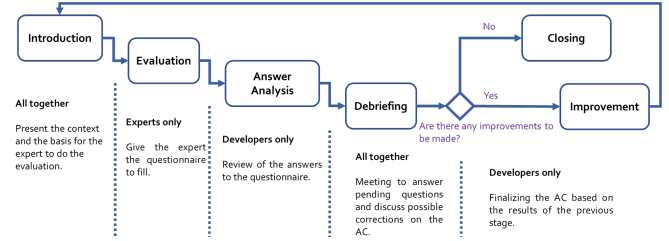


Fig. 2. Assurance case assessment process

captured using a scroll-bar that drives the linguistic qualifier of the corresponding scale.

Answers are converted to masses on belief and tautology for each rule. The quantitative approach considers the values provided by scroll-bars. The qualitative approach uses the semantic qualifiers.

It is important to know that the elicitation phase may require several round of assessment by experts. Normally during its elaboration, an AC is subject to an internal reviewing. However, Rushby et al. [20] explained that this kind of evaluation is not only insufficient, but also not very effective. This is because developers tend to justify their reasoning rather than question it, while external assessors will most likely try to criticise it. Analysis of the elicitation results provided by the external experts allowed us to improve the structure of the argument (i.e., reasoning and evidence). Hence, the necessity of reassessing the argument after each major modification, until we get a structure approved by a reasonable number of experts. Figure 2 shows this process. The answers collected during the closing phase are those that will be used to propagate confidence and uncertainty measures to the top-goal.

As shown Figure 2, the first stage after the selection of external expert(s) is to introduce the GSN standard if required, present the assurance case, and the assessment procedure. I.e., how to interpret and answer the questions in the form. Once the form is filled, answers (i.e., direct responses to questions in Figure 3 for example, and comments left by the expert(s)) are analyzed in order to detect any inconsistency or misunderstanding. A debriefing session is then scheduled to answers pending questions and discuss possible corrections on the AC. If improvements are required, the AC is modified and reassessed by a different set of experts. If this is not the case, the confidence/uncertainty measures resulting from these responses are associated to the AC so that they can be used during the propagation step.

The choice of an expert depends on his/her knowledge and competence in the fields covered by the AC. (e.g., ML, formal proof, V&V processes, etc.). Ideally the expert/assessor needs to have experience from both: (1) industrial domain to judge the use case-dependent arguments, notably for the instantiated assurance cases (i.e., all required artifacts are supplied), and (2) academia to assess relatively new methods from articles used as evidence. However, since such profiles are not easy to identify, one can call a set of experts. Aggregating their answers can be done through discussion by agreeing on a

Questionnaire for node G23 of the argument

GOA23
The <Trained ML Model> is <l₂ locally robust>

GOA24: The <ML model design> ensures that the <Trained ML Model> is <l₂ locally robust>
GOA98: The evaluation of the <Trained ML Model> demonstrates that the <Trained ML Model> is <l₂ locally robust>

	Decision scale	Symbolic scale	Numerical scale
1. Assuming GOA24 is true, what is your Decision/Confidence in the conclusion GOA23?	<----->	Strong acceptance	0.9
2. Assuming GOA24 is false, what is your Decision/Confidence in the conclusion GOA23?	<----->	Very high confidence	0.95
	<----->	No decision	0
	<----->	Very high confidence	1

Fig. 3. Extract from the form used for elicitation

single answer, which can be difficult and time-consuming. It can also be computed using aggregation formulas. This issue is not addressed in this paper since the evaluation of “Robustness AC” was made by a single expert.

V. ANALYSIS OF UNCERTAINTY MODELING AND PROPAGATION IN AC PATTERNS

Results indicate that conflicts, as meant by DST, cannot be detected at single rule level because for rules mass is only on tautology $\top = r \vee \neg r$ and the rule itself r . However variation of mass between experts can be recorded. Moreover, the results indicate that conflicts cannot be detected at node level. Indeed, it is shown that if masses on rules of expert 1 and 2 respect consistency, consistency is respected by masses on rules of the fusion. Finally, conflicts cannot be detected at tree level with an optimistic leaf assignment because the propagation of an optimistic leaf assignment induces for any node of the tree a belief in $[0,1]$ and a null disbelief. Globally those results indicate that conflicts between experts are not detectable without applying the AC to a use case.

For assessing the sensitivity of arguments to disbelief in premises, a parameter ϵ is defined and belief and disbelief in premises are set respectively to $1 - \epsilon$ and ϵ . Results indicate that for the conjunctive argument belief and disbelief of conclusion are highly sensitive to ϵ , for the disjunctive argument belief and disbelief of conclusion are not sensitive to ϵ and that for the hybrid argument belief of conclusion is sensitive to ϵ while disbelief of conclusion is not sensitive to ϵ . Nevertheless, for this argument uncertainty is sensitive to ϵ . Additional sensitivity analysis is performed by varying the mass on individual direct rule. It indicates that the decrease of this mass reduces uncertainty and increases disbelief in conclusion. Finally, it is observed that for those cases the uncertainty is equal to the degree of conflict.

VI. UNCERTAINTY IN THE AC PATTERN FOR ROBUSTNESS OF ML

A. AC pattern for robustness of ML

The root goal of the AC pattern for robustness of ML, i.e. goal 15 in Table II, is “<The Trained ML model> is <robust>”, where “<Trained ML model>” is an artifact resulting from the design and building stages of the life cycle

TABLE II
GOALS SUPPORTED BY STRATEGIES

Goal number	Wording
15	<The Trained ML model> is <robust>
17	<The Trained ML model> satisfies the <global robustness criteria>
18	<The Trained ML model> satisfies the <Global nbsample robustness criteria>
21	<The Trained ML model> is <locally robust>
23	<The Trained ML model> is <l ₂ locally robust>
24	<The ML model design> ensures that <The Trained ML model> is <l ₂ locally robust>
25	<The ML model design> integrates applicable <robustness reinforcement methods> and these methods allows that <The Trained ML model> is <l ₂ locally robust>
99	<The Trained ML model> is <l _∞ locally robust>
100	<The ML model design> ensures that <The Trained ML model> is <l _∞ locally robust>
101	<The ML model design> integrates applicable <robustness reinforcement methods> and these methods allows that <The Trained ML model> is <l _∞ locally robust>

and “<robust>” is a property defined in the AC. This goal is reformulated and then decomposed in three sub-goals, all based on the concept of local robustness. Then a decomposition is performed with respect to the norms involved in the local robustness criterion and then with respect to the way robustness can be obtained, either by design or by validation. The tree further develop the branch dedicated to *by design* methods, splitting in sub-goals corresponding to families of methods of this category. Tables II and III present some stages of this decomposition. Note that goal 19 corresponds to “The <verification set> is relevant for robustness evaluation”. Goals 98 and 178 are respectively “The evaluation of the <Trained ML Model> demonstrates that the <Trained ML Model> is <l₂ locally robust>” and “The evaluation of the <Trained ML Model> demonstrates that the <Trained ML Model> is <l_∞ locally robust>”. Finally, as shown on Figure 4, the goal corresponding to each method is supported by a set of three goals: two which are dependent on artifacts linked to the trained ML model, and one connected to a solution referencing published research articles, cf. Table IV. The goals connected to solutions for goals 30, 42, 55, 76, 103, 126, 139, 150, and 165 are respectively goals 36, 48, 60, 82, 108, 132, 143, 154, and 168.

This structure is a pattern AC and not an AC because artifacts are not present and branches of the tree can be deleted for a specific ML model.

B. Elicitation results

One expert filled forms of the type of the one shown in Figure 3, for goals connected to articles and for nodes upper in the tree. The results are derived by gathering and analyzing the filled forms. It consists in filling the AC pattern from expert’s answers.

1) *Qualitative analysis*: The analysis of answers to open questions and binary questions highlights the following points.

a) *Too demanding expert effort*: The expert indicated that he didn’t analyze articles related to goals 108, 154 and 168,

TABLE III
SUPPORTING GOALS FOR GOALS SUPPORTED BY STRATEGIES

Goal number	Sub-goals
15	17
17	18
18	19, 21
21	23, 99
23	24, 98
24	25
25	30, 42, 55, 76
99	100, 178
100	101
101	103, 126, 139, 150, 165

TABLE IV
GOALS SUPPORTED BY SCIENTIFIC ARTICLES

Goal number	Norm	Solution
36	l_2	Jacobian regularization [21]
48	l_2	Lipschitz training [22]
60	l_2	Certified robust training [23]–[25]
82	l_2	Randomized smoothing [26]–[28]
108	l_∞	Empirical robustness reinforcement [21], [29]–[32]
132	l_∞	Lipschitz training [22]
143	l_∞	Gowal certified robust training [33]
154	l_∞	Certified robust training [34], [35]
168	l_∞	Random Noising [27], [36]

i.e., Empirical robustness reinforcement method, Certified robust training and Random Noising for l_∞ robustness. It seems that the reason is the amount of effort needed to fill seriously the questionnaire is too large. Indeed, this evaluation procedure requires considerable time and effort to complete the questionnaire especially for parts concerning the goal/solution(s) nodes, which require the reading and processing of extensive documentation (e.g., technical reports, scientific articles, etc.).

b) Definitions: Concerning the definition of robustness, the expert indicated that the definition of robust provided by the AC is restrictive. For instance, this definition don’t cover robustness with respect to distribution shift. The expert thinks that in the definition of <Global nbsample robustness criteria>, i.e., “the number of samples of a subset that are <locally robust> is greater than a threshold”, a criterion of representativity of the “subset” is needed. The expert found that the wording of goal 21 is incomplete because <local robustness> is defined for a single input while it supports the goals 18 that is grounded on <Global nbsample robustness criteria> that refers to several inputs. A consistent wording for goal 21 could be “<The Trained ML model> is <locally robust> for a sufficient number of inputs”. The expert considered such wording. The addition of “for a sufficient number of inputs” could also be done for goals 23, 24, 25, 99, 100, 101 and for all goals of table IV. The expert stated that he was unable to assess Goal 19 whose wording is “The <verification set> is relevant for robustness evaluation” and support is “<Verification set>” because the definition of a relevant verification set is not provided. Nevertheless, he indicated values for the answers to the questions.

c) Contexts: For the context associated to goal 101, the expert has some doubts about the applicability for l_∞ robustness of all methods among Double Backpropagation, Jacobian regularization, Saturated Network, Ensemble adversarial training, Lipschitz Training, Wong_Kolter, Universal Random Smoothing, Feature pruning and Random Noising. Moreover, the expert has specific doubt about Lipschitz Training even if he thinks that the method helps obtaining l_∞ robustness

d) Relations between goal and sub-goals: The expert signaled that, for a given perturbation radius, goal 99 implies goal 23 because the l_2 ball is included in the l_∞ ball. This is true from a formal point of view, but the hidden Strategy is “Argument by partitioning of norms”. It seems that the expert has understood goal 21 as “<The Trained ML model> is <locally robust> for any norm with the same radius”. The expert indicated that the conjunction of goals 30, 42, 55 and 76 is impossible because the methods cannot be applied together at learning time. This also applies to goals 103, 126, 139, 150 and 165. Moreover, for the negation of the use of all methods he assumed that those methods are the only available methods.

e) Relations between goal and solutions: The expert pointed out that when multiple solutions are provided for a goal, it is unclear whether the goal shall be assessed as supported by a logical “and” or by a logical “or” of solutions. Some articles are subject to a deep analysis by the expert. For Jacobian regularization [21] the expert concludes that it improves l_2 robustness but doesn’t ensure it. For Lipschitz training [22] he indicates that a specific loss function should be used as done in recent work [37]. For Certified robust training for l_2 robustness, the expert indicates that one article [24] is out of scope

2) Quantitative analysis:

a) Completing the AC for unassessed goals: Goals 30, 42, 55, 76, 103, 126, 139, 150 and 165 are not assessed through the questionnaire. However, they have the l_x and Method of Goals 36, 48, 60, 82, 108, 132, 143, 154 and 168 respectively. In the full AC they are connected through structures like the structure of Figure 4. Without a concrete use case with a specific ML model, it is not possible to assess GOA2 and GOA4 in this figure. Thus goals 30 and 36, 42 and 48... are linked for uncertainty propagation by simple arguments with no uncertainty. Goals 98 and 178 corresponding to robustness by evaluation are not assessed through the questionnaire. At the first order, it is assumed that the evaluation provided a full confidence in robustness and that their assessment is $Bel = (1, \text{Acceptance, with Very High Confidence})$ and $Disb = (0, \text{Rejection, with Very Low Confidence})$. Despite being in the questionnaire goals 19, 108, 154, and 168 were not assessed by the expert. Goals 108, 154 and 168 are dismissed because their branches lead almost directly to a choice node with multiple incompatible alternatives. The case of goal 19, is more complex. Indeed, during the debriefing the expert suggested a quite different property than relevance for data without a clear link with robustness. Thus, the structure of goal 18 is changed to a simple argument with sub goal 21. Uncertainty of rules for

TABLE V
UNCERTAINTY ASSOCIATED TO RULES

Goal	Sub-goal(s)	Direct belief		Inverse belief	
		quantitative	qualitative	quantitative	qualitative
15	17	1.000	VH	1.000	VH
17	18	1.000	VH	0.915	VH
18	21	0.765	VH	0.845	VH
21	23	0.860	VH	1.000	VH
21	99	0.860	VH	1.000	VH
21	all	1.000	VH	1.000	VH
23	24	0.325	H	0.345	H
23	98	0.400	L	0.345	VH
23	all	0.870	VH	1.000	VH
24	25	1.000	VH	0.330	L
25	30	1.000	VH	0.500	VH
25	42	1.000	VH	0.500	VH
25	55	1.000	VH	0.500	VH
25	76	1.000	VH	0.500	VH
99	100	0.310	H	0.320	H
99	178	0.410	L	1.000	VH
99	all	0.650	H	1.000	VH
100	101	0.320	L	0.240	L
101	103	1.000	VH	0.500	VH
101	126	1.000	VH	0.500	VH
101	139	1.000	VH	0.500	VH
101	150	0.995	VH	0.500	VH
101	165	1.000	VH	0.500	VH

this simple argument is derived from the answers in the form to questions concerning goal 21 alone.

b) *Elicitation problems*: Analysis of answer to elementary questions indicate that the expert takes sometime a decision that is excessive with respect to its confidence leading to a disrespect of Josang constraint. Moreover some inconsistency between elementary and conjunctive rules is observed. Finally, some strategies that were considered by the AC developers as pure rewording or as pure logical operators are assessed differently by the expert when the goal and sub-goals are presented without explaining the strategy, indicating that the wording of goals should be revised. This has been particularly critical for the node 21, that is a choice of a norm and that was interpreted by the expert as a competition between norms. All those elicitation problems were solved during the debriefing with the expert.

c) *Elicitation of uncertainty associated to rules*: Table V presents the uncertainty associated to rules after correcting the elicitation problems. In this table VH, H and L stand reciprocally for Very High, High and Low.

d) *Elicitation of uncertainty for goals associated with solutions*: Table IV presents the uncertainty associated to goals directly supported by solutions. In the table VL stands for Very Low. The table indicates that at the leafs of the tree the expert is confident of using Lipschitz training when considering robustness criteria based on l_2 norm, goal 48, and less confident when considering robustness criteria based on l_∞ norm, goal 132. For all other methods the belief is too low and sometime the disbelief is larger than the belief.

TABLE VI
UNCERTAINTY ASSOCIATED TO GOALS LINKED TO SOLUTIONS

Goal	Belief		Disbelief	
	quantitative	qualitative	quantitative	qualitative
36	0.120	L	0.880	VH
48	0.600	VH	0.400	H
60	0.040	VL	0.060	VL
82	0.000	L	0.300	L
132	0.375	L	0.115	L
183	0.270	L	0.110	L

TABLE VII
UNCERTAINTY PROPAGATION FOR LIPSCHITZ TRAINING BASED ON l_2 NORM

Goal	Belief		Disbelief	
	quantitative	qualitative	quantitative	qualitative
42	0.600	VH	0.400	H
25	0.600	VH	0.200	H
24	0.600	VH	0.066	L
23	0.719	VH	0.014	L
21	0.618	VH	0.014	L

C. Propagation results

Results on the use of the AC pattern by a ML model developer are obtained by making uncertainty propagation under different hypotheses for the solution directly linked to artifacts. Assuming that Lipschitz training is applicable to a specific ML model, confidence can be propagated higher in the AC assuming that goals 98 and 178 related to verification will be fulfilled with very high belief and no disbelief. A user of the AC pattern will then make a propagation up to the choice between l_2 and l_∞ norms and use the propagation results to make the choice.

1) Propagation to the choice of a norm:

a) l_2 norm: As shown in Table VII for Lipschitz training considering a robustness criteria based on l_2 , the argumentation improves its initial strength, i.e. belief of 0.6 qualified as very high, because of the confidence brought by the validation, c.f. goal 23.

b) l_∞ norm: As shown by Table VIII, for a robustness criteria based on l_∞ the initial belief of 0.375, qualified as low, is also improved by the hypothesis of successful validation.

Note that for both training methods all depends on the presence of successful validation. Moreover, disbelief is reduced by propagation in the AC and reaches 0.014 and 0.003, both

TABLE VIII
UNCERTAINTY PROPAGATION FOR LIPSCHITZ TRAINING BASED ON l_∞ NORM

Goal	Belief		Disbelief	
	quantitative	qualitative	quantitative	qualitative
126	0.375	L	0.115	L
101	0.375	L	0.057	L
100	0.120	L	0.014	L
99	0.462	L	0.003	L
21	0.430	L	0.003	L

TABLE IX
POST CHOICE PROPAGATION

Goal	Belief		Disbelief	
	quantitative	qualitative	quantitative	qualitative
18	0.473	VH	0.012	L
17	0.473	VH	0.011	L
15	0.473	VH	0.011	L

qualified as low, for reciprocally l_2 and l_∞ Lipschitz training. Finally, all depends on the presence of a successful validation.

2) *Choice of a norm*: The user of the AC pattern has to make a choice on the basis of at least four criteria: quantitative and qualitative belief to maximize, quantitative and qualitative disbelief to minimize and other criteria such as cost of artifact production to minimize. Considering only the four criteria, using a ranking with Leximin the values for l_2 are (0.618, 2/3, 0.986, 1) and the values for l_∞ are (1/3, 0.430, 2/3, 0.997). 0.618 being larger than 1/3, the l_2 norm is chosen.

3) *Post choice propagation*: The Table IX presents the propagation from the choice to the top property of the AC. Quantitatively there is some decrease of belief at goal 18 due to a possible difference between local robustness and global robustness.

D. Lessons learned

1) *Strategies shall be shown*: The choice of methodology is to hide from the expert the strategies and choices. The results show that with the information included in the strategy the expert can make a quite different uncertainty assessment of rules than without this information. Moreover, this difference may lead to a quantitative difference in the assessment of the AC property. The methodology could be revised concerning hiding or not the strategies.

2) *Consistency shall be enforced*: The procedure and associated Excel file type for uncertainty elicitation developed here is based on scrollbars actuated by the expert. Each scrollbar drives at the same time a numerical value and a semantic qualifier. The scrollbar associated with decision is totally independent from the scrollbar associated with confidence. However, the Josang constraint must be respected. The results indicate that, when the Josang constraint is violated, the projection may depend on the context. This limitation could be addressed by asking first the question about confidence and limiting the decision scrollbar by the confidence value. The absence of automatic enforcement of consistency between rules at elicitation time is also a serious limitation. Finally, in case of large choices with incompatible sub goals, the question for all sub goals true and the question for all sub goals false are irrelevant. The possibilities for sub goals combinations should be assessed before making uncertainty assessment.

3) *GSN format shall be adjusted for uncertainty assessment*: So far, there is no systematic method to design an assurance case using GSN formalism. Moreover, uncertainty assessment procedures proposed in the state of the art are not mature enough to consider their features

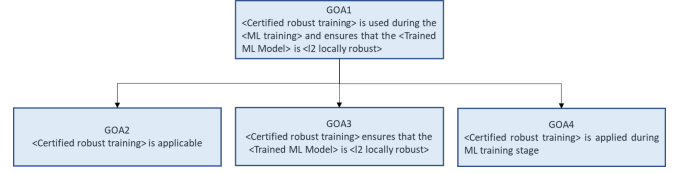


Fig. 4. Example of an argument to be adjusted for uncertainty assessment

during the development of an AC. For example, in the literature, one can find an argument that presents a goal with a method to verify a property (defined as a top goal) and another goal that argues that this method ensures the property. However, “Is supported by” arrows, formally defined by rules, already fulfill this role. I.e., saying that the application of a method m supports a property p means, according to the nature of the chosen strategy, that m ensures, demonstrates, implies, etc. p . Thus, a goal carrying the inference between a method and the property it supports, must not be considered during the uncertainty assessment. For example, questions about goal GOA3 (“<Certified robust training> ensures that the <Trained ML Model> is < l_2 locally robust >”), in Figure 4, will not be included in the form. In addition, solutions are either used as a reference to an artifact (e.g., a formal verification report, test results, etc.) or to a method to be applied. Remember that the assessment approach described in this paper does not assess the inference between the solution and the goal connected to it. However, in the second case the assessment of inference is needed. To keep coherence in the approach all solutions that carry a method are transformed to goals. The artifacts resulting from the application of these methods, such as reports results, can serve as new solutions.

4) *Multi criteria choice methods shall be integrated*: The result on comparison of approaches indicates that uncertainty modeling in AC is useful and that, when considering relevant requirements, the assessment of uncertainty shall be performed at the same time with both qualitative and quantitative approaches. This leads to a valuation of goals by four elements: the quantitative belief, the qualitative belief, the quantitative disbelief and the qualitative disbelief. For most nodes, propagation of those four elements is quite easy and for the case study the conflict mass value is always low indicating that there is no strong contradiction inside the argument. However, at choice node uncertainty propagation relies on building consistent sets of sub goals and on performing a choice among those sets. This would require a better definition of the choice and it is not sure that the propagation could be fully automatized at those nodes. Moreover, there is no total order between goals assessed following different strategies because there are four uncertainty elements and other elements such as, for instance, the cost. Thus, a multi criteria reasoning shall be performed for choosing the best solution.

5) *The AC pattern shall be extended and consolidated*: The case study highlights the benefits and some limitations of the proposed methodology. However, limited effort and time

inducted additional limitations:

- Only one expert has been involved. It is impossible to distinguish between one the one hand the results that are specific to this expert and on the other hand the results that could be consolidated with a large panel of experts.
- Uncertainty has not been assessed on the whole AC for robustness of ML. Some elements, that are not purely logical were not considered, for instance the branches corresponding to two alternative definitions of robustness and the branches corresponding to verification.
- The expert had the possibility to not assess a node or to indicate that something is missing in the argument of a node and used this possibility. This induced some doubts about the structure of the AC.

In consequence new AC patterns are derived: One pattern is developed for each norm and each robustness definition. Figure 5 presents the structure of the pattern devoted to the number of samples robustness criterion.

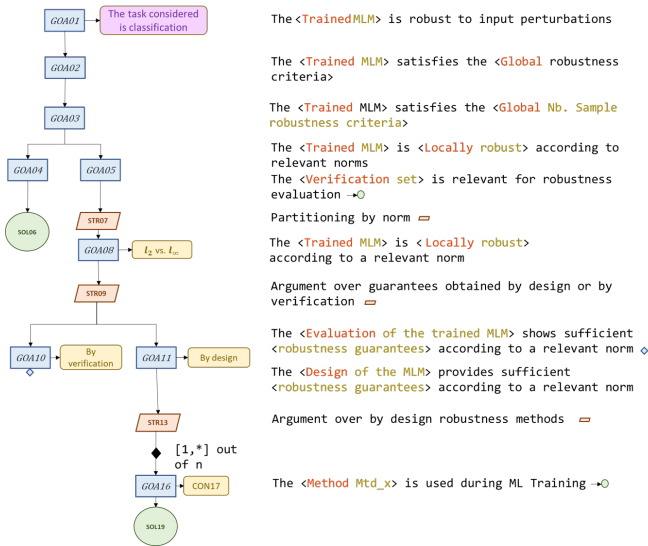


Fig. 5. Updated AC pattern

VII. CONCLUSION

The work presented here shows that recently proposed methods [19] can be applied to large AC patterns. However, the elicitation of masses requires a large number of questions to experts. Fortunately, the results obtained indicate that large AC don't imply large uncertainty on conclusion. The work also shows that it is useful to work with both scale and numbers and that the uncertainty in AC patterns contributes to performing design, implementation, integration, verification and validation choices and improving the AC structure. Finally, the result of this research will be integrated in the Capella system engineering environment¹.

An open question for future researches is the use of uncertainty levels in the context of certification and a possible link

between the qualitative belief and disbelief of the top goal of a final AC with Safety Integrity Levels or Design Assurance Levels (DAL). For instance a belief VH and a disbelief VL could be requested for DAL A and B, a belief VH and a disbelief at most L for DAL C and D and a belief VH and a disbelief at most H for DAL E.

ACKNOWLEDGMENT

This work has been supported by the French government under the "France 2030" program, as part of the SystemX Technological Research Institute.

REFERENCES

- [1] Y. Idmessaoud, D. Dubois, and J. Guiochet, "Uncertainty elicitation and propagation in gsn models of assurance cases," in *International Conference on Computer Safety, Reliability, and Security*. Springer, 2022, pp. 111–125.
- [2] —, "Confidence assessment in safety argument structure - quantitative vs. qualitative approaches," *International Journal of Approximate Reasoning*, vol. 165, p. 109100, 2024.
- [3] T. Kelly and R. Weaver, "The goal structuring notation—a safety argument notation," in *Proceedings of the dependable systems and networks 2004 workshop on assurance cases*, vol. 6. Citeseer Princeton, NJ, 2004.
- [4] Assurance-Case-Working-Group *et al.*, "Goal structuring notation community standard (version 3)," 2021. [Online]. Available: <https://scsc.uk/r141C:1?t=1>
- [5] R. Bloomfield and K. Netkachova, "Building blocks for assurance cases," in *2014 IEEE International Symposium on Software Reliability Engineering Workshops*. IEEE, 2014, pp. 186–191.
- [6] R. Wei, T. P. Kelly, X. Dai, S. Zhao, and R. Hawkins, "Model based system assurance using the structured assurance case metamodel," *Journal of Systems and Software*, vol. 154, pp. 211–233, 2019.
- [7] Z. Kurd and T. Kelly, "Establishing safety criteria for artificial neural networks," in *International Conference on Knowledge-Based and Intelligent Information and Engineering Systems*. Springer, 2003, pp. 163–169.
- [8] S. Burton and B. Herd, "Addressing uncertainty in the safety assurance of machine-learning," *Frontiers in Computer Science*, vol. 5, p. 1132580, 2023.
- [9] M. M. Gupta and J. Qi, "Theory of t-norms and fuzzy inference methods," *Fuzzy sets and systems*, vol. 40, no. 3, pp. 431–450, 1991.
- [10] G. Shafer, *A mathematical theory of evidence*. Princeton university press, 1976, vol. 42.
- [11] D. Dubois, F. Faux, H. Prade, and A. Rico, "Qualitative capacities: Basic notions and potential applications," *International Journal of Approximate Reasoning*, vol. 148, pp. 253–290, 2022.
- [12] E. Denney, G. Pai, and I. Habli, "Towards measurement of confidence in safety cases," in *2011 International Symposium on Empirical Software Engineering and Measurement*. IEEE, 2011, pp. 380–383.
- [13] D. Nešić, M. Nyberg, and B. Gallina, "A probabilistic model of belief in safety cases," *Safety science*, vol. 138, p. 105187, 2021.
- [14] C. Hobbs and M. Lloyd, "The application of bayesian belief networks to assurance case preparation," in *Achieving Systems Safety: Proceedings of the Twentieth Safety-Critical Systems Symposium, Bristol, UK, 7-9th February 2012*. Springer, 2011, pp. 159–176.
- [15] J. Guiochet, Q. A. Do Hoang, and M. Kaaniche, "A model for safety case confidence assessment," in *Computer Safety, Reliability, and Security: 34th International Conference, SAFECOMP 2015, Delft, The Netherlands, September 23-25, 2015, Proceedings 34*. Springer, 2015, pp. 313–327.
- [16] R. Wang, J. Guiochet, G. Motet, and W. Schön, "Modelling confidence in railway safety case," *Safety Science*, vol. 110, pp. 286–299, 2018.
- [17] C.-L. Lin, W. Shen, S. Drager, and B. Cheng, "Measure confidence of assurance cases in safety-critical domains," in *Proceedings of the 40th International Conference on Software Engineering: New Ideas and Emerging Results*, 2018, pp. 13–16.
- [18] Y. Idmessaoud, D. Dubois, and J. Guiochet, "A qualitative counterpart of belief functions with application to uncertainty propagation in safety cases," in *International Conference on Belief Functions*. Springer, 2022, pp. 231–241.

¹<https://eclipse.dev/capella/>

- [19] Y. Idmessaoud, "Uncertainty assessment in safety argument structures—an approach based on dempster-shafer theory," Ph.D. dissertation, UPS Toulouse, 2022.
- [20] J. Rushby, X. Xu, M. Rangarajan, and T. L. Weaver, "Understanding and evaluating assurance cases," Tech. Rep., 2015.
- [21] D. Jakubovitz and R. Giryes, "Improving dnn robustness to adversarial attacks using jacobian regularization," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 514–529.
- [22] C. Anil, J. Lucas, and R. Grosse, "Sorting out lipschitz function approximation," in *International Conference on Machine Learning*. PMLR, 2019, pp. 291–301.
- [23] E. Wong, F. Schmidt, J. H. Metzen, and J. Z. Kolter, "Scaling provable adversarial defenses," *Advances in Neural Information Processing Systems*, vol. 31, 2018.
- [24] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu, "Safety verification of deep neural networks," in *Computer Aided Verification: 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I 30*. Springer, 2017, pp. 3–29.
- [25] M. Hein and M. Andriushchenko, "Formal guarantees on the robustness of a classifier against adversarial manipulation," *Advances in neural information processing systems*, vol. 30, 2017.
- [26] J. Cohen, E. Rosenfeld, and Z. Kolter, "Certified adversarial robustness via randomized smoothing," in *international conference on machine learning*. PMLR, 2019, pp. 1310–1320.
- [27] H. Hong, B. Wang, and Y. Hong, "Unicr: Universally approximated certified robustness via randomized smoothing," in *European Conference on Computer Vision*. Springer, 2022, pp. 86–103.
- [28] M. Lecuyer, V. Atlidakis, R. Geambasu, D. Hsu, and S. Jana, "Certified robustness to adversarial examples with differential privacy," in *2019 IEEE symposium on security and privacy (SP)*. IEEE, 2019, pp. 656–672.
- [29] A. Ross and F. Doshi-Velez, "Improving the adversarial robustness and interpretability of deep neural networks by regularizing their input gradients," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, no. 1, 2018.
- [30] F. Tramèr, A. Kurakin, N. Papernot, I. Goodfellow, D. Boneh, and P. McDaniel, "Ensemble adversarial training: Attacks and defenses," *arXiv preprint arXiv:1705.07204*, 2017.
- [31] A. Nayebi and S. Ganguli, "Biologically inspired protection of deep networks from adversarial attacks," *arXiv preprint arXiv:1703.09202*, 2017.
- [32] J. Gao, B. Wang, Z. Lin, W. Xu, and Y. Qi, "Deepcloak: Masking deep neural network models for robustness against adversarial samples," *arXiv preprint arXiv:1702.06763*, 2017.
- [33] S. Gowal, K. Dvijotham, R. Stanforth, R. Bunel, C. Qin, J. Uesato, R. Arandjelovic, T. Mann, and P. Kohli, "On the effectiveness of interval bound propagation for training verifiably robust models," *arXiv preprint arXiv:1810.12715*, 2018.
- [34] E. Wong and Z. Kolter, "Provable defenses against adversarial examples via the convex outer adversarial polytope," in *International conference on machine learning*. PMLR, 2018, pp. 5286–5295.
- [35] H. Zhang, H. Chen, C. Xiao, S. Gowal, R. Stanforth, B. Li, D. Boning, and C.-J. Hsieh, "Towards stable and efficient training of verifiably robust neural networks," *arXiv preprint arXiv:1906.06316*, 2019.
- [36] M. Lecuyer, V. Atlidakis, R. Geambasu, D. Hsu, and S. Jana, "On the connection between differential privacy and adversarial robustness in machine learning," *stat*, vol. 1050, p. 9, 2018.
- [37] M. Serrurier, F. Mamalet, A. González-Sanz, T. Boissin, J.-M. Loubes, and E. Del Barrio, "Achieving robustness in classification using optimal transport with hinge regularization," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 505–514.

Session Tu.4.B
Logical Execution Time

Tuesday 11th June

17:00

–

Lauragais Room

Separation of functional and time interference concerns for efficient AMC 20-193 compliance

Damien CHABROL¹, Jean GUYOMARC'H¹, Fabien SIRON¹, Guillaume PHAVORIN^{1,3},
Sam THOMPSON², Eric JENN³, François THURIEAU⁴

¹ Asterios Technologies, Massy, France
{firstname.lastname}@asterios-technologies.com

² Rapita Systems Ltd, York, UK
sthompson@rapitasystems.com

³ IRT Saint Exupéry, Toulouse, France
{firstname.lastname}@irt-saintexupery.com

⁴ Safran Electronics & Defense, Massy, France
francois.thurieu@safrangroup.com

Abstract—Safety-critical real-time systems must comply with stringent certification requirements, including temporal ones. Failure to comply with these temporal requirements may contribute to system failure. Therefore, timing considerations, such as response times, are of the foremost importance for such systems. As the use of multi-/many-core hardware platforms is becoming inevitable in the avionics industry due to the increasing computing performance required by modern embedded systems, integration activities are getting more and more complex. Increasing concurrency and parallelism exacerbates integration issues and introduces new challenging problems. To answer those challenges, certification authorities have issued guidelines, referenced as A(M)C 20-193, describing some additional objectives to fulfill for multi-/many-core integration. The present paper describes how a time-aware approach, based on the Synchronous Logical Execution Time paradigm (sLET), makes the design and integration of A(M)C 20-193 compliant safety-critical multi-/many-core systems easier by separating functional and time interference concerns.

Keywords—safety-critical real-time systems, strong determinism, synchronous Logical Execution Time, multicore timing analysis, AMC 20-193.

I. INTRODUCTION

Avionics systems, and more particularly safety-critical ones, are usually subjected to stringent certification constraints to ensure their compliance with safety requirements. Indeed, failure of such systems may result in catastrophic consequences. In particular, for high-criticality real-time systems, functions should be ensured to complete within strict timing constraints. In addition, the avionics industry increasingly relies on modular systems, where multiple applications of possibly different criticality levels can safely share a common hardware platform. Integration activities must ensure that all hosted applications still meet their functional and temporal requirements after their composition in the final system.

Driven by the increasing computing performance required by modern embedded systems and the obsolescence of high-performance single-core processors (SCPs), the avionics industry is moving towards the use of multi-/many-core processor (MCP) hardware platforms. But the increasing parallelism and potential throughput brought by MCPs comes at a cost: integration activities are getting more and more complex, and thus time-consuming and costly [1].

The use of MCPs exacerbates integration issues already present for SCPs. Moreover, it introduces new challenging problems [2]. In particular, when multiple cores are used and different threads of execution simultaneously access the same hardware resource (e.g., a shared memory, a bus, etc.), the hardware must arbitrate these concurrent accesses, effectively introducing additional latency on some of the accesses. This time interference may lead to the violation of the system's temporal requirements. It can also result in new or different data or control coupling paths, and thus functional interference causing the system to behave in a non-deterministic way, or possible data corruption [3]. For example, with MCPs, functional modules can be allocated to different CPU cores, which may create inter-core execution dependencies due to inter-task synchronization (e.g., for communication or to prevent race conditions). Thus, one task could prevent another task, allocated to a different core, from running, and thus forestall any other computation. This may have a significant impact on efficiency and testability for MCP systems.

As per DO-178C/ED-12C, safety-critical systems are usually associated with the highest Design Assurance Levels (i.e., DAL-A or DAL-B). Certification of MCP systems to the highest criticality levels presents the greatest challenge for the avionics industry. As functional and time interference may degrade the system safety, applicants must elaborate an argumentative strategy defending that their systems are indeed robust to such interference. Certification authorities have been working on guidelines to address this specific topic, with the AMC 20-193 document recently issued by EASA [3], and its AC counterpart from the FAA [4]. Few safety-critical MCP systems have been certified until now, and very often at the cost of underusing the additional CPU cores. Thus, new methodologies and tools are needed to support the development and integration process of MCP systems and meet the objectives defined in the A(M)C 20-193.

In this paper, (i) we propose a time-aware strategy based on the Synchronous Logical Execution Time (sLET) paradigm, which encompasses time from system-level design to integration on the final hardware (Section III), (ii) then we show how sLET helps to tackle the MCP functional and temporal interference problem (Section IV), and (iii) eventually, we discuss the application of such time-aware strategy to avionics safety-critical systems (Section V) and illustrate it on an industrial case study, using the combination of **ASTERIOS** and Rapita's on-target analysis tools and **MACH**¹⁷⁸ multicore certification solution (Section VI).

These works have been supported by the French Defense Procurement Agency (DGA) and the French National Research Agency (ANR) in the context of respectively the ASTERLINK and ARCHEOCS projects.

II. POSITIONING

A. Time-aware approaches

Many programming abstractions have been developed to model and reason about real-time systems. For safety-critical systems, they are often coupled with a time-triggered execution model due to its determinism and predictability [5].

The Synchronous-Reactive (SR) model, implemented by synchronous languages [6], totally abstracts execution time to focus on logical instants on which computations are triggered. Each computation should be completed before the next possible instant, and its output must be available before any other computations could use it on the same instant. Thus, SR languages offer both determinism and concurrency. For an implementation on an actual target, logical instants are then mapped to physical time. Multiple logical clocks can exist in SR programs to design multi-rate systems. But due to the causality between computations, induced by instantaneous communications, compilation of SR languages can be quite complex, in particular for multicore platforms (on which computations can be parallelized) [7]. The PRELUDE architecture design language offers a solution to implement multi-periodic synchronous systems, by adding real-time primitives to specify the durations of tasks [8]. Then, the program can be automatically translated into a set of real-time tasks, with periods of tasks computed using clock calculus [9]. Finally, those tasks can be scheduled on-line using policies such as Deadline-Monotonic or Earliest-Deadline-First [10].

The Logical Execution Time (LET) paradigm, implemented for example in the GIOTTO [11] and TDL [12] languages, describes the logical duration taken by computation. Each computation must fit in a logical interval, called LET interval [13]. Furthermore, communications are only made on the boundaries of LET intervals, to ensure determinism. Thus, compared to the SR model, the LET paradigm allows more time variability due to the specified logical duration, which makes concurrent implementations easier. But this comes at the price of a lesser temporal and functional expressiveness, as (i) LET applications in GIOTTO and TDL are limited to strictly periodic tasks and (ii) contrary to SR, LET builds on a delayed communication model.

B. Multicore timing analysis

Multicore timing analysis usually aims to determine safe WCET estimates for software hosted on multicore processors. Different methods can be used: 1) measurement-based analysis, 2) static analysis (deterministic or probabilistic), or 3) hybrid approaches combining both previous points [14]. In every case, the primary challenge that must be overcome is that of multicore interference channels.

1) Interference channels

An interference channel is defined in A(M)C 20-193 as being ‘a platform property that may cause interference between software applications or tasks’. Interference channels can be discovered in many parts of a processor, and are often (but not always) associated with shared hardware resources. Interference channels may be one of the following:

- A bandwidth constraint: e.g., a shared interconnect will typically have a finite bandwidth available which must be shared between any bus masters.
- A space constraint: e.g., shared caches have finite capacity, and tasks that are executing concurrently on

different cores may cause evictions of data and instructions that belong to each other, leading to an increase in cache misses and thus execution time.

- An indirect coupling: e.g., a coherency mechanism, whose purpose is to ensure that all levels of cache maintain a consistent view of the state of the memory.

A resource may contain (i) no interference channels, (ii) just one or two, or (iii) a large number (e.g., some complex interconnects, shared caches, and network accelerators can contain more than 10 independent interference channels).

There exist formulations of multicore interference that instead of treating ‘interference channels’, are built around the concept of ‘interference paths’. The pre-eminent example of the latter is the PHYLOG Model Language (PML) [15], which considers all the possible ‘initiators’ of transactions, all the possible ‘targets’ for transactions, and all the possible routings between initiators and targets. The assertion follows that if all intersections of these paths are exercised, then all multicore interference will have been assessed. While this approach can provide assurance that all bandwidth constraints are likely to have been tested, additional analysis and test specification may be necessary to ensure that the indirect interference channels have also been adequately covered.

2) Static Analysis and Measurement-based Methods

In older single-core avionic systems, static analysis and simulation can prove useful for timing analysis. However, for complex multicore systems, this is no longer the case. Modern, high-performance processors (particularly multicore ones) have many complex features, such as multilevel caches and DMA engines, which frequently (and, in the case of features like random replacement caches, intentionally) sacrifice temporal determinism in favor of average-case performance. These mechanisms can be very hard to model with sufficient accuracy. Moreover, most modern processors incorporate IP cores from a wide range of sources. So, the silicon vendor may either not be in a position to share (or even build) a complete model of the processor. Furthermore, due to this complexity, errors or inconsistencies in implementation and integration of these IP cores are common, resulting in real-world behavior that doesn’t perfectly match the documentation.

Owing to these complexities, A(M)C 20-193 takes a cautious approach to static analysis, and states that ‘simulation of those [interference] mechanisms is, therefore, less likely to be representative in terms of functionality or execution time than testing conducted on the target MCP in the intended final configuration, and thus is less likely to detect errors.’ If an airworthiness authority deems that an analysis method is ‘less likely to detect errors’, then it should generally be avoided.

3) Accounting for pre-emptions

In processors with caches, it is well-documented that pre-emption can be delayed by cache state [16]. In a multicore context, it’s also possible for pre-emption to be delayed by operations from other cores, as typically cache coherency transactions will have a higher priority than local accesses.

From a multicore timing perspective, the determination of the maximal pre-emption latency isn’t significantly different to the single-core case. However, there may be additional scenarios that need testing (e.g., including cases where other cores are generating many coherency transactions with the intent of maximizing the additional impact).

III. SYNCHRONOUS LOGICAL EXECUTION TIME

Hereafter, we introduce the synchronous Logical Execution Time (sLET) paradigm. This is a generalization of the Psy model introduced during the 90's [17].

A. sLET paradigm

The sLET paradigm combines the benefits (but also some shortcomings) of both the Synchronous-Reactive (SR) and Logical Execution Time (LET) models [18]. It bridges the gap between both approaches by incorporating LET intervals into SR. Thus, it combines SR's properties with more time variability, which makes concurrent implementations (in particular multicore scheduling) easier.

As for the SR model, logical and physical times are considered independent under the sLET paradigm and serve different purposes. Logical time is used to specify the system high-level temporal requirements through an abstraction of time, whereas physical time corresponds to the execution time of the system implementation on a specific hardware platform. As for the SR model, time in sLET is purely logical in the sense that physical time is fully replaced by partial or total ordering between computations. In sLET, logical time is expressed through logical clocks. A logical clock, in the sense described by Lamport [19], abstracts time through a series of events called clock ticks. sLET can be seen as a multiform logical time [20] generalization of LET, hence using multiple logical clocks. A set of clocks C can be constrained by a set of precedence and simultaneity relations (e.g., periodicity).

B. Tasks and Elementary Actions

In sLET, all computations are specified by their activation and termination events, expressed using logical clocks. Computation time intervals can be abstracted by the concept of the *Elementary Action* (EA): an EA is a computation that fits in a sLET interval, bounded by the EA's activation and termination dates. Thus, as depicted in Figure 1, it is defined as a sequence of instructions constrained by two logical dates, referred to as *Temporal Synchronization Points* (TSPs):

- An *activation date* (its earliest start date) defined on some event of a logical clock.
- A *termination date* (its deadline) defined on some other event of a possibly different logical clock (with both clocks related together with a total order).

So, unlike the LET paradigm, sLET does not rely on logical durations: instead, as for SR, it uses logical clock instants (of possibly different non-harmonic clocks) to specify interval boundaries. For example, the EA depicted in Figure 1 has its activation date defined on the second tick of Logical Clock $c1$ and its termination date defined on the fifth tick of Logical Clock $c2$. Thus, the sLET interval is defined in terms of clock events and not as a logical duration.

An sLET task is defined by an infinite sequence of EAs. The termination date of an sLET interval corresponds to the activation date of the next interval for the task. Note that it is possible in sLET to have empty logical intervals (i.e., in which no EA from the task can be executed). It allows, for example, to define periodic tasks with constrained deadlines (i.e., with a relative deadline strictly smaller than the period). Moreover, as logical intervals are defined at the EA level (for a same sLET task), this makes it possible to design more complex temporal patterns than strictly periodic tasks.

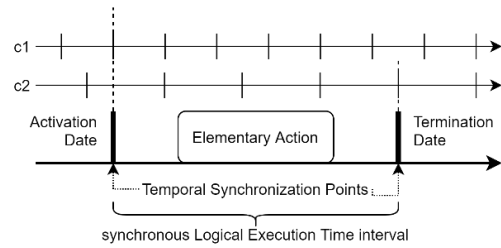


Figure 1: Example of sLET interval (the activation date, respectively termination date, is defined on Clock $c1$, resp. $c2$).

C. Visibility principle

As part of a larger system, a task usually consumes and produces data from and to other tasks during its execution. To ensure determinism, sLET inter-task communication is performed through dedicated channels implementing the *Visibility Principle* [21]. Note that by determinism we mean 'the ability to produce a predictable outcome [...] based on the preceding operations and data [...] in a specific period of time with repeatability' [3]. Under the Visibility Principle:

- Data produced by an EA over an sLET communication channel will only become visible (i.e., available) to the rest of the system from the end of the EA's interval, i.e., for a logical date greater or equal to the EA's termination date.
- Data can be consumed by an EA from an sLET communication channel only if it has become visible prior to the start of the EA's interval, i.e., for a logical date lesser or equal to the EA's activation date.

The logical date from which the data becomes available to some other EA is referred to as the *Visibility Date*. Usually, this corresponds to the termination date of the EA producing the data, but some sLET communication channels may have their own temporal behavior defined on a different logical clock: in that case, the Visibility Date corresponds to the tick of that logical clock which is greater or equal to the EA's termination date. Note that the producer and consumer's clocks, as well as the one used to define the Visibility Date, must be related with a total order. For example, a data produced by the EA depicted in Figure 1 can only become visible after its termination date (defined on the fifth tick of $c2$). If we assume that the Visibility Date corresponds exactly to the EA's termination date, this means that another EA can consume this data only if its own activation date is defined on either the same logical tick (fifth tick of $c2$) or a tick occurring afterwards (e.g., sixth tick of $c2$, eighth tick of $c1$, etc.).

So, as for LET, sLET builds on a delayed communication model, which can somehow limit functional expressiveness, in particular compared to SR. Note that, some kind of instantaneous communication can actually be achieved in sLET (as briefly introduced in Section IV), but at the cost of more complex concurrent implementations.

D. Implementation for safety-critical systems

To implement an sLET design on a specific hardware platform, logical time is mapped to physical time. sLET tasks must then be properly scheduled to ensure design timing constraints (i.e., sLET interval bounds). For a single-core platform, the logical ordering of EAs is sufficient to guarantee the correctness of the execution, whereas for MCPs, inter-core synchronization is required to preserve the logical ordering across CPU cores. Moreover, for the resulting tasks'

scheduling to be valid, all EAs must have enough CPU time to complete before the end of their respective intervals. In the remainder of this document, we focus more specifically on static scheduling, based on Time-Division Multiplexing, as it provides strong guarantees on predictability [22] and is generally favored for safety-critical avionics systems. In this case, for deploying an sLET design on a specific hardware target, the user must provide a Time Budget (TB) for each EA, corresponding to the maximum amount of physical time allocated to the computation of that EA. Based on the relations between logical clocks and the provided TBs, a compiler may generate a time-triggered schedule. In that respect, a given scheduling (and so the corresponding sLET implementation) is valid if no EA actually exceeds its allocated TB at run time.

For safety-critical systems, a TB should be an upper-bound on the worst-case execution time (WCET) of the corresponding EA, to ensure that the resulting schedule will always be valid at run time. Thus, safe TB values are synonymous with safe WCET estimates. How such safe TBs could be obtained is discussed in Section V.

IV. TAMING FUNCTIONAL AND TIME INTERFERENCES

Hereafter, we assume that every computation (i.e., EA) is provided with enough physical time (i.e., safe TB) to complete within its logical time constraints.

A. Functional and time interference

As defined in [23], interference corresponds to an alteration of the processor’s behavior (e.g., longer delay required for a load operation, etc.) experienced by some part of the software executed on one CPU core, and related to the activity of the remaining software running on the other cores.

As per A(M)C 20-193, time interference can be produced, for example, when the MCP arbitrates simultaneous accesses to shared hardware resources, causing contention for those resources and therefore an increase in execution time [3]. Execution of concurrent software on a different CPU core, and in particular the time interference that may be induced, can result in new or different data or control coupling paths leading to functional interference: a communication buffer may be sometimes read before being written (depending on the producer’s and consumer’s actual execution times), shared data could be corrupted if accessed in parallel, etc.

B. Preventing functional interference

The sLET Visibility Principle applies the LET communication model [13] to logical clocks. Provided tasks exchange data exclusively through sLET communication channels (H1), their execution is solely driven by their associated logical clocks. It means that communications between tasks become independent from the underlying real-time scheduling (resulting from the implementation of the system on a specific platform). Any schedule complying with the logical constraints defined by the sLET design results in the same functional behavior, as long as physical timing constraints are fulfilled (H2). Thus, if this later hypothesis holds, sLET allows for transparent distribution as functional determinism is ensured whatever the allocation of tasks to CPU cores. This allows dataflow determinism to be achieved. For example, a longer execution of a third-party EA (not involved in the same functional chain) may delay the execution of a producer EA. This could result in the corresponding consumer EA (allocated to another core) being

executed beforehand. But, thanks to the Visibility Principle, this has no impact on the dataflow determinism: the consumed data does not depend on the actual execution instants but solely on the sLET intervals bounds. Therefore, functional interference can be prevented by design using the sLET model, as long as tasks exclusively communicate through sLET-based communication mechanisms.

It is the responsibility of the user to ensure that their application complies with both H1 and H2. If H1 is a design constraint, H2 is closely related to the design’s implementation on a specific hardware target.

C. Preventing time interference

Time interference between tasks can arise within a single CPU core, e.g., due to cache effects. MCP time interference adds to this ‘traditional’ time interference, making WCET estimation harder. Here, we focus on MCP time interference, as some extensive work has already been conducted regarding mitigation methods for single-core time interference [14].

Multicore-related time interference can originate from deep and intricate hardware implementation details [24]. Preventing contention (or at least, bounding or minimizing contention) for MCPs reduces potential time interference. A wide spectrum of methods and techniques are available to address this objective, many of which can be used in combination. This paper focuses specifically on temporal exclusion, which can be enabled thanks to the sLET model.

With imperative and non-temporal programming models, concurrent accesses to shared resources (hardware peripheral, software buffer, etc.) are usually guarded, e.g., using mutexes or semaphores. Using sLET, temporal exclusion can be enforced by design, and automatically verified, to prevent such concurrent accesses. Thus, sLET can be used to guarantee that simultaneous accesses to a shared resource never happen, while preventing some issues encountered with mutexes, such as deadlocks. This temporal exclusion is provided through *exclusion groups* [25]. An exclusion group provides a safety property: the EAs it contains must not share any physical date in common. More formally, for a set of sLET tasks T , with $E^t = \{e_i | i \in \mathbb{N}\}$ the infinite sequence of EAs that constitute Task t (e_i being the i^{th} EA of t), an exclusion group G is defined as $G \subset E = \bigcup_t E^t : \forall e_i, e_j \in G, e_i \cap e_j = \emptyset$. As a result, a common resource accessed only by EAs from a single exclusion group can only be accessed by at most one EA from that group at a time.

As previously stated, an sLET task consists in an infinite succession of EAs, each bounded by an activation date and a termination date defined on clock events. So, given that all logical clocks can be reduced to a unique global clock, it is possible to define sLET intervals for EAs of a same exclusion group such that they never overlap in time. For example, let us consider the two EAs depicted in Figure 2. Originally (assumably to cope with some high-level timing requirements), EA1’s sLET interval is delimited by the first and fourth ticks of $c1$ and EA2’s interval by the first and third ticks of $c2$. This means that both intervals overlap in time. If EA1 and EA2 need to be part of a same exclusion group, a solution is to modify the sLET design, as depicted in the left sub-figure: EA1’s termination date is now defined on the second tick of $c1$ and EA2’s activation date on the second tick of $c2$. Thus, sLET intervals no longer intersects and temporal exclusion is achieved.

Thus, by constructing exclusion groups, time interferences caused by contention on shared resources may be strongly constrained. However, defining such timing exclusion groups comes at a cost: the user needs to re-design some part(s) of its temporal architecture. This may be arduous work, depending on the temporal patterns of the different tasks. Moreover, this means introducing additional timing constraints (i.e., new or different TSPs) to manage multicore interference (related to a specific integration). This may result in different sLET intervals, which means that overall latencies (derived from high-level requirements) may also change. One solution, described in Section VI, is to deal with timing exclusion only at implementation level: instead of re-designing sLET intervals, scheduling is used to enforce the specified exclusion groups (e.g., by introducing precedence constraints between EAs to avoid concurrent execution). Another solution, using an additional sLET construct, is introduced hereafter.

D. Fractional Temporal Synchronization Points

The sLET paradigm extends ‘classic’ logical clocks with the concept of *fractional logical clocks*. As any logical clock, a fractional clock abstracts time through a series of clocks ticks, referred to, in this case, as fractional logical ticks. The difference is that a fractional clock is defined with regard to a ‘standard’ logical clock, such that there is exactly one fractional tick occurring between any two consecutive ticks of the ‘parent’ logical clock. Note that, as depicted in the right part of Figure 2, this fractional tick can occur anywhere in-between. This means in particular that two fractional clocks defined with regard to the same ‘parent’ clock cannot be compared as their fractional ticks may occur in any order between two consecutive ticks from the ‘parent’ clock.

sLET intervals for Elementary Actions can only be defined using ‘standard’ logical clocks, which means that any EA’s activation and termination dates necessarily correspond to logical clock ticks. But an EA execution in its sLET interval can be over-constrained using fractional logical clocks:

- The activation of an EA can be further ‘delayed’ until after some fractional logical tick, referred to as a *fractional Temporal Synchronization Point*.
- The termination of an EA can be constrained before some other fractional TSP.

For example, as depicted in the right part of Figure 2, EA1 can be constrained to be executed before the second fractional tick of c_{r1} (derived from Logical Clock c_1) and EA2 after it.

As can be seen in that example, when using fractional TSPs there is no modification of the original sLET interval: the EA’s activation and termination dates are left unchanged. As the Visibility Principle is defined with regard to sLET interval boundaries, Visibility Dates for data flows are left unchanged and thus the corresponding latencies. Thus, fractional logical clocks can be used to implement exclusion groups without impeding the original temporal architecture: one EA’s execution can be *constrained before* a fractional TSP while the execution of another EA of the same exclusion group is *delayed after* the same fractional TSP, thus ensuring that they don’t overlap over time. For example, as depicted in the right part of Figure 2, EA1 and EA2 can no longer be executed concurrently thus enforcing the exclusion group without modifying the original sLET intervals.

Note that, fractional clocks can be used to achieve instantaneous communication in sLET. It is possible to define

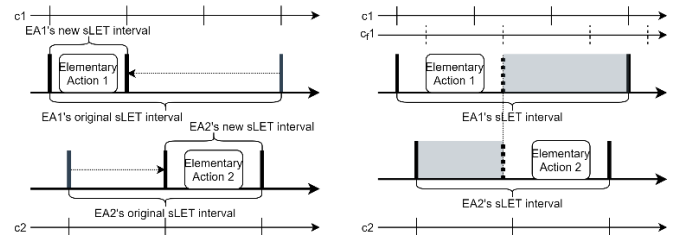


Figure 2: Example of exclusion group between two EAs enforced by: (left) sLET re-design, and (right) using a fractional TSP defined on fractional Logical Clock c_{r1} derived from c_1 (‘classic’ clock ticks and TSPs are in plain lines whereas fractional ones are denoted by dashed lines).

sLET communication channels with regard to fractional clocks, instead of ‘standard’ clocks. In that case, the Visibility Date corresponds to a fractional TSP. On the example from Figure 2, this means that a data produced by EA1 could become visible from the fractional TSP onward, and thus be consumed by EA2.

The additional constraints introduced through fractional TSPs should be ensured by the implementation. Either the corresponding fractional tick is mapped to an actual physical date, as it is the case for ‘standard’ logical dates, or fractional TSPs are used to derive precedence constraints between EAs’ executions that should be ensured by the tasks’ scheduling.

V. TOWARDS A TIME BUDGETS EVALUATION ENCOMPASSING TIME INTERFERENCE

As stated in Section III, an sLET implementation relies on compliance with respect to physical timing requirements. In particular, unaccounted time interference defeats this hypothesis. We discuss here how this issue can be addressed.

A. Basis of the approach

As stated before, we focus on static scheduling. A given schedule is valid if no Elementary Action exceeds its allocated Time Budget at run time. To evaluate safe TBs for all EAs, we propose a measurement-based A(M)C 20-193 compliant approach encompassing time interference. Note that other methodologies, as discussed in Section II, are possible.

1) Time Budgets in isolation

The goal of the approach presented hereafter is to compute Time Budgets *in isolation* (from a scheduling point of view, i.e., non-preemptive WCETs [26]). This means that additional delays due to pre-emptions (e.g., additional cache misses resulting from cache evictions caused by the pre-empting tasks) should be accounted for separately when considering the final integration (i.e., with all the application’s tasks). As the approach targets multicore integration, this TB in isolation should encompass the maximum possible overhead due to MCP interference. Indeed, dealing at the scheduling level with the interaction between tasks executing in parallel on different CPU cores is much harder than accounting for pre-emptions, and might not always be feasible. In the general case, it’s not possible to reason about the test vectors necessary to drive one task to suffer the maximum possible impact from interference caused by another task. Moreover, synchronization is very important for interference impact. Even a single clock cycle of jitter between cores can make a large difference to the interference inflicted on one core by another.

Considering TBs in isolation allows tasks to be handled separately for the timing evaluation, making measurement

campaigns and analyses easier. It also allows for composable approaches and re-usability, and thus possible incremental certification [27], as a single TB could be considered for different multicore integrations of the same task.

2) Incremental approach

The TB evaluation approach presented hereafter is incremental and consists of four main steps:

1. First, an evaluation is performed in single-core to compute a TB upper-bound in isolation, referred to as a *single-core Time Budget*.
2. Then, analyses are conducted to identify possible multicore interference channels and quantify their impact on the different tasks.
3. From those results, multicore interference can be accounted for, either by implementing some mitigation means, or by computing an upper-bound on the maximal overhead to be added to the single-core TB, to derive a *multicore Time Budget*.
4. Finally, multicore TBs for all tasks are verified.

The first three steps are conducted on tasks in isolation. Only the final step is performed on the final configuration.

B. Single-core Time Budget evaluation

First, a TB evaluation is performed in single-core for each task in isolation. As the approach targets TBs in isolation, it means that each task is considered separately, without needing other parts of the application to be present. This is possible as the execution of sLET tasks is solely driven by logical time. So, each task can be executed independently from the others. Of course, inter-task communications, if any, might need to be stubbed. Thanks to the sLET visibility principle, this is easier to achieve as data availability is well-defined.

Measurements are performed using maximizing tests, i.e., exercising the worst-case execution paths for each task at run time, which have to be defined by the applicant on a case-by-case basis. Coverage analyses can be helpful to achieve confidence when building these tests. Moreover, additional metrics might also be collected at this step (e.g., number of memory reads/writes, cache hits/misses, etc.), to (i) construct a profile for the task, which could help understand some software variabilities, and (ii) identify the resources actually used by the task. The high-water mark (HWM), i.e., the highest measured execution time, for each EA can be retrieved from the measurements. Then, a safety margin might be added to get a single-core Time Budget for each EA of the task.

C. Hardware characterization

Hardware characterization deals with the identification and characterization of possible interference channels. To do so, several steps are needed:

1. Hardware resource identification: (i) the resources of the processor need to be identified, and (ii) those that may contain interference channels are singled out.
2. Interference channel identification: any singled-out resource is analyzed in detail, to identify the possible interference channels it contains [23].
3. Interference channel characterization: any non-fully mitigated interference channel is characterized on target to determine its possible effect.

Note that both the hardware resource and interference channel identifications are paper activities and are performed using any available technical documentation and datasheets.

1) Hardware resource identification

Hardware resource identification is required by A(M)C 20-193's MCP_Planning_2 objective. It is important to note at this stage that not all multicore interference channels arise from the explicit sharing of resources. For example, cache coherency mechanisms can cause interference even when only private cache memories are being accessed.

2) Interference channel identification and characterization

Interference channel identification and characterization are partly to satisfy A(M)C 20-193's MCP_Resource_Usage_3 objective. Characterization can also be used to provide evidence that some interference channels can have no practical or measurable timing impact. This activity should be conducted on target. Interference generators can be used for this purpose [28]. For each channel, it is required to:

1. Determine what properties such a benchmark must possess to be sensitive to that interference channel.
2. Execute and perform measurements on the 'sensitive' benchmark on one core, while other cores are idle, to establish a baseline when there is no interference.
3. Identify the properties a benchmark must possess to be aggressive on the interference channel.
4. Execute and perform measurements on the 'sensitive' benchmark on one core, while the 'aggressive' benchmarks are run on the other cores.
5. Compare the timing properties of the 'sensitive' benchmark with and without interference.

3) Mitigation mechanism identification

In parallel with the hardware resource identification, mitigation mechanisms for these interference channels should be identified. Different mitigation levels are possible:

- Hardware configuration. For example, it may be possible to mitigate an interference channel related to cache evictions by configuring cache partitioning on hardware platforms that support it. Alternatively, hardware devices and features may be disabled to remove some interference channels.
- Integration-level configuration. For example, a specific data/code placement in memory could be configured to enforce spatial partitioning for some resources. Time partitioning at scheduling level can also be used to achieve exclusion between some tasks' executions and thus avoid concurrent access to some resource.
- Software architecture. For example, timing exclusion can be ensured by creating exclusion groups, either through sLET intervals re-design or by adding additional timing constraints using fractional clocks.

D. Multicore Time Budget evaluation

Once characterized, the identified interference channels need to be accounted for to derive multicore TBs in isolation. This can be done by mitigating the interference, or upper-bounding its maximal impact to add it to the single-core TB.

1) Multicore interference impact evaluation

Results from the interference channel characterization can be used to assess whether the impact is sufficiently small for the interference channel to be neglected. For interference

channels that cannot be neglected, their actual impact on the different tasks needs to be assessed. Indeed, interference may not have an impact for all interference channels, depending on the actual use of hardware resources by the different tasks.

As for the hardware characterization step, the interference impact evaluation should be conducted on target:

1. First, each sLET task is executed in isolation on one core, with some instrumentation for timing and resource usage, while other cores remain idle.
2. Then, the list of interference channels against which the task should be characterized is refined, removing the ones related to resources the code won't exercise.
3. Finally, the task is executed again on one core, while exercising the remaining interference channels (using the same combinations of 'aggressive' benchmarks as for the hardware characterization).

Comparing for each EA the distributions of execution times measured with and without the 'aggressive' benchmarks provides the applicant with qualitative and quantitative information which allows the identification of: (i) interference channels of concern, i.e., ones that can actually cause time interference due to their use by the application, and (ii) EAs making significant use of each identified interference channel.

Results from the hardware characterization and the interference impact evaluation can be used to discriminate among identified channels, between: (i) those for which the impact is acceptable (in terms of safety but also performance [23] for safety-critical systems), and (ii) those for which mitigation is required. Indeed, full mitigation for all interference channels is impossible in practice, except for very simple applications [29]. Nevertheless, through A(M)C 20-193, the goal is not to reach total freedom from interference, but rather to demonstrate upper bounds on the possible impact of time interference, and that safety is not impacted. A quantitative criterion (e.g., statistical) or an empirical observation may be used, as proposed in [30]. Note that the exact meaning of 'significant' is to be defined by the applicant regarding their needs, as it is an integral part of the argumentation process and highly dependent on the use-case.

2) Multicore time interference mitigation

Mitigation strategies, such as spatial isolation or temporal exclusion using exclusion groups, should be enforced for those interference channels with the most 'significant' impact on processing time, or for high-criticality tasks (e.g., DO-178C/ED-12C DAL-A). For example, for all EAs impacted by the same interference channels, an exclusion group could be constructed by adding additional fractional TSPs so that those EAs can no longer be executed simultaneously. Note that dealing with EAs allows a finer granularity than working at the task level: indeed, a task may not access a given shared resource in all its EAs, and thus some of them could be executed in parallel with other tasks' EAs.

Once implemented, mitigation strategies should be validated. For spatial isolation, a new interference impact evaluation could be conducted; as for temporal exclusion, this may be only tested on the final integrated system.

3) Multicore Time Budget

Finally, the impact of the non-mitigated interference can be bounded for each EA, thanks to the interference impact evaluation step. This upper-bound can then be added to the single-core TB to account for MCP interference. As for the

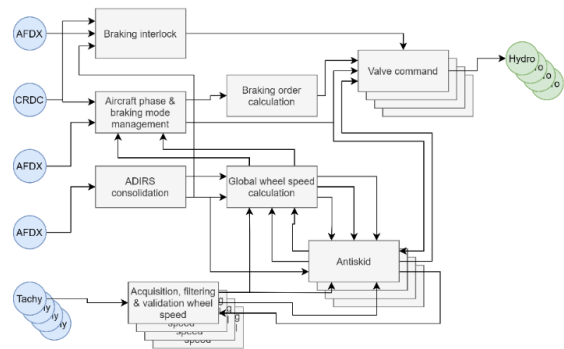


Figure 3: Functional architecture of the LGS application.

single-core case, a safety margin might be added to get the final multicore Time Budget for each EA.

E. Multicore Time Budget validation

Eventually, measurement campaigns on the final configuration should be conducted to validate that the computed multicore Time Budgets are actually upper-bounds (all measurements for an EA in the integrated system should be less than the multicore TB derived from the previous step).

For this step, all the application's tasks must be considered at the same time. In case of pre-emptions, (i) the maximum number of times each task can be pre-empted should be evaluated, and (ii) an upper-bound on the overhead the task might experience due to a pre-emption should be computed.

VI. APPLICATION TO A CASE STUDY

To illustrate the approach, we consider an industrial use case from Safran Electronics & Defense. This application work has been conducted as part of the ARCHEOCS project. For our case study, we focus on a single interference channel.

A. Presentation of the use-case

The use-case consists of a simplified Landing Gear System (LGS), in charge of the aircraft main undercarriage. As depicted in Figure 3, it has 5 functional chains:

- One duplicated acquisition and command chain per side of the undercarriage, to get the wheel speed and apply the braking order on the hydraulic valves.
- The main chain to compute the braking command.

The different functions are all executed periodically, but at different rates: from 1Hz (for the braking order calculation function) up to 10Hz (for the acquisition part).

The LGS is deployed over an NXP T1042 multicore hardware platform consisting of four e5500 PowerPC cores running at 1.4GHz, with private L1 caches, split between instructions and data, and a unified L2 cache per core. An interconnect (CoreNet) is used to access a shared 4GB DDR4 memory, as well as several peripherals and accelerators.

B. Tools to support a full time-aware strategy

To support an application of the TB evaluation approach on the LGS, we use **ASTERIOS** as our integration solution and Rapita's tools to help with the TB evaluation process.

1) ASTERIOS solution

The **ASTERIOS** solution is developed and commercialized by ASTERIOS Technologies (formerly Krono-Safe), based on a technology from the CEA (French

Atomic and Alternative Energies Research Organization). It offers a time-aware methodology, supported by a set of industrials tools, to develop safety-critical embedded systems.

ASTERIOS is centered around an implementation of the sLET model as the PsyC language. It comes with a dedicated toolchain to (i) help with the design and configuration of a PsyC (i.e., sLET) application and (ii) support the compilation for a given hardware target. EA timing constraints (i.e., sLET intervals), extracted from the PsyC design, once mapped to physical time, and Time Budgets, provided by the user for a specific hardware target, can be used as inputs for automatically computing a feasible static schedule (if any) thanks to the **ASTERIOS** toolchain. To support and enforce sLET execution at run time, **ASTERIOS** provides a certified target-specific real-time microkernel which implements time and space partitioning. In particular, it ensures that the schedule generated by the toolchain is not violated at run time (i.e., that no EA exceeds its TB): a run time mechanism is able to detect any violation to prevent the offending task (or the whole application) from continuing its execution, as neither timing nor functional determinism can thereafter be ensured. Finally, **ASTERIOS** offers a qualified tool to verify that the toolchain's outputs are compliant with the user's input (and in particular the specified sLET design) [31].

2) Rapita's solution

Rapita Systems provides a tool suite, called Rapita Verification Suite (**RVS**), to support verification of critical aerospace and automotive systems. From a multicore timing perspective, it allows users to: (i) analyze and verify scheduling behavior on-target using **RapiTask**, (ii) analyze and verify software timing behavior on-target down to the basic block level using **RapiTime**, (iii) automate test harness generation using **RapiTest**, and (iv) perform testing that exercises specific multicore interference channels using **RapiDaemons**. Where applicable, these tools are available with DO-330/ED-215 qualification kits.

RVS can be used as a key part of Rapita's **MACH**¹⁷⁸ solution for certifying multicore aerospace projects in accordance with DO-178C/ED-12C and A(M)C 20-193. **MACH**¹⁷⁸ comprises several components, including software tools with associated qualification kits; procedures, templates, and checklists for generation of multicore certification evidence; an IP library covering interference channels in popular avionic multicore processors; and specialist engineering and consultancy services. The **MACH**¹⁷⁸ procedures both directly address A(M)C 20-193 objectives related to multicore timing, but also intend to provide the required supporting evidence. For example, if debug performance counters are used to provide evidence that: (i) a tool is performing correctly; or (ii) some software is not accessing a particular hardware resource, then these counters also need validation. Therefore, an event monitor validation procedure is incorporated.

3) Tools integration

The LGS software is integrated on the T1042 platform using **ASTERIOS**. Each function is mapped to a PsyC task. Two additional tasks are added for logging. All inter-task communications are performed through sLET communication channels. At this point, the logical and functional behavior of the PsyC application, in particular the data/control coupling, can be verified offline (i.e., without a compilation and execution on the T1042) thanks to the dedicated **ASTERIOS** simulator. Specifically, it allows verification that worst-case

dataflow latencies resulting from the application's timing architecture (according to the visibility principle) are compatible with the high-level end-to-end requirements.

For the LGS application integration on the T1042, a static allocation of the tasks to the CPU cores is used. The main chain tasks, as well as the two logging tasks (one per core), are allocated to Cores 0 and 1. The duplicated chains are allocated to Cores 2 and 3 (2 chains per core). All tasks have access to the shared DDR4 memory. Moreover, all caches are enabled and a write through policy is set for the data cache.

As each task is strictly periodic, we consider one single Time Budget for all EAs of a same task. For TB evaluation, Rapita's tools have to be used with **ASTERIOS**. A connection has been prototyped as part of the ARCHEOCS project: (i) an interfacing layer allows **RVS** to derive **ASTERIOS**-relevant timing results, which means in particular computing timing estimates for each EA, and (ii) **RapiDaemons** can be run against an **ASTERIOS** application on dedicated CPU core(s) without altering scheduling on other core(s).

C. Single-core Time Budget evaluation

As presented in Section V, TB evaluation is conducted on each task in isolation. To stub the communications from and to that task, an additional task is added to act as a 'mock' producer and receiver. It is allocated to the same CPU core as the task under analysis, to avoid creating multicore interference, and its timing behavior is designed to match exactly the one of the task under analysis (i.e., same period, as all LGS tasks are strictly periodic), to avoid any pre-emption.

Moreover, to conduct measurements, a valid schedule is required for the task in isolation. So, an initial TB has to be provided. This presents a cyclic dependency, as the goal of this initial schedule is to perform measurements that will allow an actual TB to be derived. To overcome this issue, and as each task is run in isolation, oversized TBs can be used for the sole purpose of generating a valid schedule. Another solution is to use the concept of ambivalent logical clock, which is implemented in **ASTERIOS**. An ambivalent clock can map logical time to physical time but can also switch to purely logical execution (i.e., regardless of physical time). Since ambivalent clocks are logical clocks, the execution of the scheduled tasks remains correct: logical ordering is preserved, only their physical timing constraints are altered. Thus, ambivalent clocks are definitely not suitable for production systems, but can be used to logically execute a whole system on a hardware target by relaxing the TBs constraints at run time: if a TB is exceeded, then the ambivalent clock allows for the corresponding EA to complete its execution by temporarily delaying any activation of other EAs.

The task's code is instrumented to capture timing and resource usage information on each activation and termination of an EA of the task. The maximal observed Time Budget estimates (i.e., HWMs) for a few tasks are summarized in Table I. There is quite a large variability in execution times among the different tasks, from a few μ s to more than 1ms.

D. Hardware characterization

1) Hardware resource identification

The output of the hardware resource identification step should be a complete list of the hardware components in the platform. This can then be used to check that all relevant hardware resources have been adequately analyzed and

characterized. Additionally, this activity can provide an early indication of whether there is adequate documentation available for the platform to support further analysis.

For our case study, we focus on a single resource, the T1042’s shared DDR memory. As it is used by all tasks for instructions and data (including stacks), this is likely to be a major interference source for the application. In a typical DDR controller, there are several interference channels. We focus only on the one concerned with competition for rowbuffers. A DDR memory device stores data in ‘rows’, which in the T1042 are 8KiB long. When data is requested from a particular row, the DDR controller performs a destructive read on the entire row and buffers it in a rowbuffer. While the row is in the buffer, many reads and writes may be performed on that row. When the row has been finished with, the buffer is needed for another transaction, or a timeout has been reached, it is written back into the DDR device. On many multicore platforms, this interference can cause a significant increase in execution time.

2) Interference channel characterization

To evaluate the maximal possible impact of rowbuffer interference, we use a specifically tailored RapiDaemon, targeting the DDR controller rowbuffers, as our interference benchmark. The T1042’s DDR controller contains 64 rowbuffers: up to 64 rows may be buffered at a time, but accesses to unbuffered rows require that a currently-buffered row is written back to the DDR memory device before the new row can be accessed. So, the RapiDaemon used in this case study is designed to cause a row eviction with every instruction executed; this should be able to demonstrate the worst-case effect of contention for rowbuffer availability.

For the interference channel characterization, we create a specific task, referred to as the unit under test (UUT), to act as a benchmark for the analysis. It executes the RapiDaemon code on Core 0 at a 10Hz frequency. RapiDaemons on other cores (to create interference), are executed continuously in bare metal. For the measurements, we consider 4 scenarios:

1. No RapiDaemon is run in parallel with the UUT.
2. 1 RapiDaemon is run in parallel on Core 1.
3. 2 RapiDaemons are run on Cores 1 and 2.
4. 3 RapiDaemons are run on Cores 1, 2 and 3.

Timing measurements are retrieved for each execution of the UUT under each scenario. From those measurements, TB estimates accounting for rowbuffer interference are computed for each execution of the UUT, using the RVS tools with the dedicated ASTERIOS interfacing layer. As depicted in Figure 4, the impact of interference can be quite high: up to a 43% increase for the HWM when suffering from interference due to RapiDaemons running on all three remaining cores.

3) Mitigation mechanisms identification

Different mechanisms provided by ASTERIOS can be used for mitigation. At design level, we can use fractional

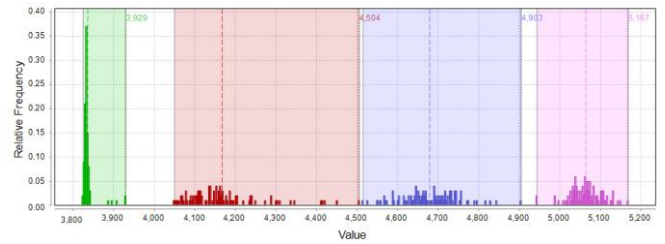


Figure 4: Histograms of execution times (in μs) computed by RVS for the hardware characterization under the different scenarios (green: isolation; red: 1 RapiDaemon; blue: 2 RapiDaemons; purple: 3 RapiDaemons).

TSPs to construct exclusion groups between some EAs. At integration level, we can rely on the ASTERIOS toolchain’s frames exclusion mechanism, which allows the automatic computation (if possible) of a static schedule enforcing a temporal exclusion between some EAs specified by the user. Note that spatial partitioning is not considered as it would be more cumbersome to implement for a shared-memory architecture like the T1042. For other architectures using memory hierarchies (with some shared memory and other local to the CPU cores), this could be a sustainable solution.

E. Multicore Time Budget evaluation

1) Multicore interference impact evaluation

As competition for rowbuffers can be a potentially significant interference channel, its actual maximal impact on the LGS tasks needs to be quantified. This time, we deal with each task as the UUT and we consider the same 4 RapiDaemons configuration scenarios as previously.

HWMs for the worst-case scenario (3 RapiDaemons) are synthesized in Table I. As all tasks access the DDR, there is always some interference when running contender code. But its impact differs a lot: some tasks of the main chain suffer from overhead of several dozen μs (compared to a few μs for the other tasks). As this impact is larger than the HWMs of most tasks, we chose to consider them as part of an exclusion group for which some mitigation should be implemented.

2) Multicore interference mitigation

For the LGS case study, we consider 2 different mitigation means serving different purposes. First, we deal with the 2 logging tasks which both share a common resource (the logging mechanism). As they can be executed in parallel, this could lead to a functional interference. So, a temporal exclusion between their EAs is enforced, using fractional TSPs. Then, we consider the set of tasks that can suffer significantly from interference over the DDR4 memory, identified in the previous step. As those tasks have very different rates, implementing temporal exclusion through fractional TSPs might be quite hard and over constraining when generating the static schedule. Thus, we rely on the ASTERIOS toolchain’s frames exclusion mechanism to generate a static schedule ensuring the temporal exclusion.

Table I: Timing results for LGS tasks.

Task	Single-core evaluation	Multicore interference evaluation (3 RapiDaemons)	Overhead	Multicore final integration (with mitigation)
ADIRS consolidation	1397.0 μs	1455.2 μs	+58.2 μs	1492.0 μs
Aircraft phase & braking mode management	1255.3 μs	1306.2 μs	+50.9 μs	1316.3 μs
Braking order calculation	1264.6 μs	1310.4 μs	+45.8 μs	1279.6 μs
Global wheel speed calculation	1279.0 μs	1325.2 μs	+46.2 μs	1294.6 μs
Other tasks	12.5 μs -115.9 μs	13.5 μs -120.7 μs	+0.9 μs -5.3 μs	21.6 μs -260.0 μs

3) Towards multicore Time Budgets

To derive safe multicore TBs (in the context of a certification project), all possible interference channels should be dealt with. This can be seen from the measurements conducted on the final configuration (i.e., integration of all the LGS tasks on the T1042). As depicted in Table I, HWMs for all tasks are larger than the ones observed for the interference evaluation step. Thus, there are clearly other interference channels that should be characterized and accounted for. Note that, for the mitigated tasks, the increase in execution times remains quite small (less than 10%). So, the impact of those other interference channels could be accounted for as an additional safety margin on the multicore TB.

VII. SUMMARY AND PERSPECTIVES

In this paper, we described a time-aware strategy suitable for safety-critical real-time systems, based on the sLET paradigm. We showed that sLET properties can help a DO-178C/ED-12C applicant build an argumentative strategy for answering A(M)C 20-193 objectives related to functional and time interferences. Thanks to sLET, functional interference is fully prevented by design, and time interference can be restrained through temporal exclusion.

The application of a sLET-based strategy to an industrial use case has been illustrated using the **ASTERIOS** solution, which is already being deployed by Safran Electronics & Defense for single- and multicore commercial systems. To meet the required A(M)C 20-193 objectives, we showed that Rapita's approach and tools can support interference and timing analyses for **ASTERIOS**-based systems.

In future steps, we plan to further develop our multicore Time Budget evaluation methodology to provide a comprehensive solution for implementing and integrating safety-critical real-time systems on MCPs.

References

- [1] S. Gerhold, M. Dunham and B. Sletteland, "Alternative multi-core processor considerations for aviation," 2018.
- [2] C. Maiza *et al.*, "A Survey of Timing Verification Techniques for Multi-Core Real-Time Systems," *ACM Computing Surveys*, 2020.
- [3] European Union Aviation Safety Agency, "AMC 20-193 Use of multi-core processors," 2022.
- [4] Federal Aviation Administration, "AC 20-193 - Use of Multi-Core Processors," 2024.
- [5] S. Baruah and G. Fohler, "Certification-Cognizant Time-Triggered Scheduling of Mixed-Criticality Systems," *RTSS*, 2011.
- [6] A. Benveniste *et al.*, "The synchronous languages 12 years later," *Proceedings of the IEEE*, 2003.
- [7] A. Yip *et al.*, "The ForeC Synchronous Deterministic Parallel Programming Language for Multicores," *MCSOC*, 2016.
- [8] J. Forget *et al.*, "A Multi-Periodic Synchronous Data-Flow Language," *HASE*, 2008.
- [9] J. Forget, "Un Langage Synchrone pour les Systèmes Embarqués Critiques Soumis à des Contraintes Temps Réel Multiples," 2009.
- [10] C. Pagetti *et al.*, "Multi-task Implementation of Multi-periodic Synchronous Programs," *Discrete event dynamic systems*, 2011.
- [11] T. A. Henzinger, B. Horowitz and C. M. Kirsch, "Giotto: a time-triggered language for embedded programming," *LNCS*, 2003.
- [12] E. Farcas *et al.*, "Transparent distribution of real-time components based on logical execution time," *LCTES*, 2005.
- [13] C. M. Kirsch and A. Sokolova, "The Logical Execution Time Paradigm," *Advances in Real-Time Systems*, 2012.
- [14] R. Wilhelm *et al.*, "The worst-case execution-time problem—overview of methods and survey of tools," *ACM Transactions on Embedded Computing Systems*, 2008.
- [15] F. Boniol *et al.*, "Modelling and analyzing multi-core COTS processors," *ERTS*, 2022.
- [16] C.-G. Lee *et al.*, "Analysis of Cache-Related Preemption Delay in Fixed-Priority Preemptive Scheduling," *IEEE Transactions On Computers*, vol. 47, no. 6, pp. 1-14, 1998.
- [17] V. David *et al.*, "Safety properties ensured by the oasis model for safety critical real-time systems," *SAFECOMP*, 1998.
- [18] F. Siron *et al.*, "The synchronous Logical Execution Time paradigm," *ERTS*, 2022.
- [19] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Communications of the ACM*, 1978.
- [20] C. André, F. Mallet and M.-A. Peraldi-Frati, "A multiform time approach to real-time system modeling; Application to an automotive system," *SIES*, 2007.
- [21] M. Lemerre and E. Ohayon, "A Model of Parallel Deterministic Real-Time Computation," *RTSS*, 2012.
- [22] P. Axer *et al.*, "Building timing predictable embedded systems," *ACM Transactions on Embedded Computing Systems*, 2014.
- [23] X. Jean, L. H. Mutuel and R. Soulat, "Assurance of Multicore Processors: Limits on Interference Analysis," *FAA Final report*, 2020.
- [24] J. Bin *et al.*, "Studying co-running avionic real-time applications on multi-core COTS architectures," *ERTS*, 2014.
- [25] J. Guyomarc'h *et al.*, "Non-Simultaneity as a Design Constraint," *TIME*, 2020.
- [26] J. Cavicchio and N. Fisher, "Integrating Preemption Thresholds with Limited Preemption Scheduling," *RTCSA*, 2020.
- [27] S. H. VanderLeest and D. C. Matthews, "Incremental Assurance of Multicore Integrated Modular Avionics (IMA)," *DASC*, 2021.
- [28] P. Radojković *et al.*, "On the evaluation of the impact of shared resources in multithreaded COTS processors in time-critical environments," *ACM Transactions on Architecture and Code Optimization*, 2012.
- [29] F. J. Cazorla, J. Abella and E. Mezzetti, "Dissecting Robust Resource Partitioning, Robust Time Partitioning, and Robust Partitioning in CAST-32A," *SAE*, 2021.
- [30] A. Ferlin, E. Jenn and M. Kaufmann, "Accounting for interferences in the design of Time-Triggered Applications," *ERTS*, 2020.
- [31] A. Methni, E. Ohayon and F. Thuricau, "ASTERIOS Checker: A Verification Tool for Certifying Airborne Software," *ERTS*, 2020.

Reducing End-to-End Latencies of Multi-Rate Cause-Effect Chains in Safety Critical Embedded Systems

Luiz Maia, Gerhard Fohler
University of Kaiserslautern-Landau, Germany
{maianeto,fohler}@rptu.de

Abstract—The *Logical Execution Time (LET)* model has deterministic timing and data-flow properties, which simplify the computation of end-to-end latencies of multi-rate cause-effect chains. However, the LET model results in pessimistic end-to-end latencies since it abstracts the underlying platform and scheduling choices. In this paper, we propose a method to reduce end-to-end latencies of multi-rate cause-effect chains applying the LET model, by considering knowledge of the schedule in later design phases of safety critical embedded systems. Our method shortens and shifts the communication intervals of the LET model. If needed, e.g., for legacy reasons, our method can be applied to a subset of tasks only. We evaluate our work based on automotive benchmarks and synthetic task sets. We compare our results with previous work and the LET model. The experiments show significant reductions of maximum *reaction time* and *data age* values.

Index Terms—Safety Critical Embedded Systems, Real-Time Systems, End-to-End Timing Analysis, LET

I. INTRODUCTION

Designing safety critical applications in embedded systems, such as in AUTOSAR, requires complex analysis for temporal properties, such as end-to-end latencies. During early design phases, designers abstract system semantics, e.g., scheduling algorithms, in order to reduce the complexity. However, abstracting system semantics in this manner results in pessimistic end-to-end latencies [1].

A *cause-effect chain (CEC)* represents a sequence of tasks, which are executed to achieve a given functionality. A typical example is a sensor to actuator CEC, which consists of a task that reads the sensor (*cause*), a task that processes the read value, and a task that writes the output to an actuator (*effect*).

The analysis of whether or not the end-to-end (E2E) latency between the cause and the effect respects system's timing requirements is not trivial [2]. Especially when the CEC contains tasks with different periods and multiple data dependencies. The complexity of computing E2E latencies of multi-rate CECs increases even further when considering multi-core systems as tasks can be mapped to different cores and can execute in parallel [3].

Depending on the adopted communication model, the points in time when inter-task communications (accesses to shared resources) occur can be non-deterministic. They depend on when tasks start and finish their execution.

The Logical Execution Time (LET) model [4] emerged as a solution which significantly reduces timing analysis complexity of multi-rate CECs. By having fixed inter-task communication points that are independent from the actual task execution, the LET model brings timing and data-flow determinism to the analysis of multi-rate CECs. In LET, inter-task communication only occurs at the boundaries of the so-called *communication interval* [5], which is considered equal to the period interval of the task. As a result, the LET model helps abstracting from the actual system implementation (scheduling choices), and consequently reduces complexity of analysis, but at the cost of increased pessimism, i.e., larger E2E latency values.

In this paper we propose a method to reduce the pessimism present in the LET model by taking scheduling choices into consideration. The method shortens and shifts communication intervals based on a chosen scheduling algorithm. Therefore, our method is applied later in the design process (after scheduling choice). By analyzing a feasible schedule, it derives new boundaries for the communication intervals.

Our method is built on the ideas of adding phase to specific tasks [6] and shortening the communication interval equal to task's worst-case response time [7].

As design phases progress and a schedule has to be determined, our method can be applied during later design phases to optimize the E2E latencies of multi-rate CECs applying the LET model. Without losing the timing and data-flow determinism of LET, our method keeps tasks periodic and with well-defined communication points. If needed, it can be applied individually to selected tasks and/or CECs during later design phases, e.g., for legacy reasons. Results from evaluation based on an automotive benchmark presented by BOSCH [8], as well as synthetic task sets show a reduction of $\approx 65\%$ for the E2E latencies, on average.

Summary of contributions: Our method

- significantly reduces the E2E latencies of multi-rate CECs applying the LET model
- shortens and shifts tasks' communication intervals by taking scheduling decisions into consideration
- keeps tasks periodic and with well-defined inter-task communication points
- can be applied individually to selected tasks and/or CECs for legacy reasons

II. RELATED WORK

The two most commonly considered latencies when analyzing a multi-rate CEC are: *reaction time* (*First to First* semantic) and *data age* (*Last to Last* semantic) [9]. The reaction time measures the *reactivity* of the system. It is the time interval between the occurrence of an external event until the *first* output based on that event. Data age measures the *freshness* of data in the CEC. It is the time interval between a data sampled (read) by the first task in the CEC until the *last* output (actuation) based on such data is produced by the last task in the CEC. Recently Günzel et al. [10] showed that the values for the maximum reaction time and maximum data age are equivalent.

In order to take data propagation delays into account, Klaus et al. [11] proposed an extension of the Real-Time Systems Compiler (RTSC), while Forget et al. [12] proposed a language to do a formal verification of E2E constraints at the model level. Becker et al. [13] proposed to use job-level dependency as a way to control data propagation and E2E latencies in multi-rate CECs. In [14], Dürr et al. introduced the concept of *job chains* and provided an analysis of the maximum reaction time and maximum data age. Schlatow et al. presented in [15] an analysis of the data age for periodic offset-synchronized tasks. In [16], Günzel et al. presented a timing analysis of asynchronous distributed CECs.

The LET model [4] was first introduced as part of the Giotto programming language in the context of time-triggered tasks. In [17], Biondi et al. presented a method to implement the LET model using additional dedicated tasks to realize the logical behavior of LET. In [18], Pazzaglia et al. used LET to enforce causality and determinism as a way to control accesses to shared memory and optimize the functional deployment on multi-core platforms.

In [19], Kloda et al. proposed to decouple the communication interval from the periods of tasks as an extension for the Timing Definition Language (TDL) [20], a successor of Giotto. However, Kloda et al. [19] did not present a method to formally compute the additional offsets or how to actually decouple the communication intervals.

Techniques have been proposed to compute the E2E latencies of multi-rate CECs applying the LET model. Becker et al. [21] presented a method to compute the maximum data age considering different communication models. Kordon and Tang [22] proposed a method to determine the maximum data age based on a task dependency graph. In [6], Martinez et al. presented a phase-aware LET analysis to improve the E2E latencies of multi-rate CECs. In [7], Bradatsch et al. proposed a method to reduce data age by setting the communication intervals equal to tasks' worst-case response time.

We build our work on top of the ideas proposed by Martinez et al. [6] and Bradatsch et al. [7]. By taking scheduling decisions into consideration, our method shortens and shifts communication intervals while keeping tasks periodic and with well-defined inter-task communication points.

III. SYSTEM MODEL

We consider a multi-core system composed of identical cores and a task set Γ containing periodic and independent real-time tasks.

A. Tasks and Jobs

A task τ is a tuple $(C_\tau, T_\tau, D_\tau, \phi_\tau)$, where C_τ represents the worst-case execution time (WCET), T_τ is the period, D_τ is the deadline, and ϕ_τ is the phase. We assume tasks have implicit deadlines, i.e., deadline is equal to period. A *job* J represents an instance of τ , where $J(i)$ is the i^{th} instance of τ , $i \in \mathbb{N}^+$. $J(i)$ has a release time at $\phi_\tau + (i - 1)T_\tau$ and an absolute deadline D_τ time units later. A schedule \mathcal{S} specifies the execution behavior of all jobs of τ according to a scheduling policy. The *start time* of $J(i)$ according to \mathcal{S} is $s_{J(i)}^{\mathcal{S}}$, while the *finishing time* is $f_{J(i)}^{\mathcal{S}}$. If the choice of a schedule is clear, we omit the index \mathcal{S} for all definitions.

B. Communication Model

We assume communication between tasks happens through the use of shared resources and to be based on the LET model. Each task τ has a fixed and well defined communication interval L_τ (Figure 1). The inputs and outputs of τ are logically updated at the boundaries of L_τ . $begin(L_\tau)$ and $end(L_\tau)$ are relative points in time representing the boundaries of L_τ , i.e., $L_\tau = [begin(L_\tau), end(L_\tau)]$. $|L_\tau|$ represents the length of interval L_τ .

Each job J of τ has a communication interval L_J . The boundaries of L_J define when a job J logically receives (read) input from a shared resource, as well as when it logically transmits (write) output to a shared resource. At $begin(L_J)$, the logical *read-event* of J from a shared resource occurs. For instance, if $begin(L_\tau) = 0$, that means the logical read-event of each $J \in \tau$ happens during its release. At $end(L_J)$, the logical *write-event* of J to a shared resource occurs.

Figure 1 shows the communication boundaries of interval $L_{J(i)}$ for a given job $J(i)$ of task τ assuming $|L_\tau| = T_\tau$, i.e., $L_\tau = [0, T_\tau]$.

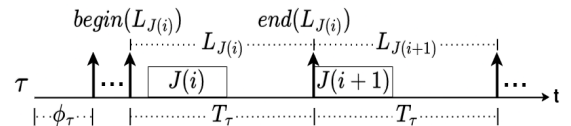


Fig. 1: Communication boundaries of interval $L_{J(i)}$ for a given job $J(i)$ of task τ assuming $L_\tau = [0, T_\tau]$

C. Cause-Effect Chain

A Cause-Effect Chain (CEC) represents an ordered sequence of communications carried out between a finite set of tasks. We represent a CEC by $E = (\tau_1 \rightarrow \tau_2 \rightarrow \dots \rightarrow \tau_{|E|})$, $|E|$ being the number of tasks in E . The function $E(i)$ returns the i^{th} task in E , $i \in \{1, 2, \dots, |E|\}$. The \rightarrow operator indicates that τ_{i+1} acts as a consumer/reader task, while τ_i as a producer/writer task.

We assume that E samples (acquires data) at every $begin(J_1)$, J_1 being a job of task $E(1)$. We use z to represent the time interval between the occurrence of an external event (input) and its sampling by J_1 . Likewise, z' represents the time interval between $end(J_{|E|})$ and the actuation (output).

D. Job Chain

Given a CEC E , a job chain $c^E = (J_1 \rightarrow \dots \rightarrow J_{|E|})$ is a finite sequence of jobs representing one of the possible data propagation paths of E . In a job chain, the following requirements are respected:

- J_i is a job of $E(i)$, $i \in \{1, 2, \dots, |E|\}$.
- The data written by J_i is read by J_{i+1} . That is, $end(L_{J_i}) \leq begin(L_{J_{i+1}})$ for all $i \in \{1, 2, \dots, |E| - 1\}$

We use c_i^E to represent the i^{th} job chain of E , $i \in \mathbb{N}^+$, while function $l(c^E)$ returns the time interval between $end(L_{J_{|E|}})$ and $begin(L_{J_1})$.

IV. MANIPULATING LET COMMUNICATION INTERVALS TO REDUCE END-TO-END LATENCIES

As discussed in Section I, multi-rate CECs applying the LET model have timing and data-flow determinism, but pessimistic E2E latencies. By exploiting information from a feasible schedule, our method reduces the pessimism present in the LET model while maintaining its deterministic characteristics and the periodicity of tasks.

Instead of setting $|L_\tau| = T_\tau$, i.e., $L_\tau = [0, T_\tau] \forall \tau \in \Gamma$, our method derives new relative points in time for $begin(L_\tau)$ and $end(L_\tau)$. By repositioning the boundaries of L_τ and therefore the boundaries of L_J , it can postpone the logical read-event of J and prepone the logical write-event of J .

A. Defining Schedule-Aware Intervals

In order to make L_τ schedule-aware, $\forall \tau \in \Gamma$, our method sets L_τ 's length and position equal to a new time interval I_τ , where the length of I_τ is $C_\tau \leq |I_\tau| \leq T_\tau$. $begin(I_\tau)$ and $end(I_\tau)$ delimit the boundaries of I_τ , i.e., $I_\tau = [begin(I_\tau), end(I_\tau)]$.

The length and position of I_τ are defined according to schedule \mathcal{S} . As explained in Section III-A, \mathcal{S} specifies the start time s_J and the finishing time f_J for all $J \in \tau$.

Below we define the terms *relative start time* (S_J) and *relative finishing time* (F_J) to derive the communication boundaries for I_τ .

Definition 1: Relative Start Time (of a Job). Let $J(i)$ be the i^{th} job of task τ in schedule \mathcal{S} . The relative start time ($S_{J(i)}$) of a job is the start time of $J(i)$ minus its release time.

$$S_{J(i)} = s_{J(i)} - (\phi_\tau + (i-1)T_\tau) \quad (1)$$

Definition 2: Relative Finishing Time (of a Job). Let $J(i)$ be the i^{th} job of task τ in schedule \mathcal{S} . The relative finishing time ($F_{J(i)}$) of a job is the finishing time of $J(i)$ minus its release time.

$$F_{J(i)} = f_{J(i)} - (\phi_\tau + (i-1)T_\tau) \quad (2)$$

Depending on when each J executes between its release and deadline, the values for S_J and F_J can change for each $J \in \tau$ (one job may execute early during its period, while another job may execute later). In order to keep the timing and data-flow determinism of LET when setting $L_\tau = I_\tau$, it is necessary to ensure that all $J \in \tau$ have a common periodic communication interval, i.e., respects S_J and F_J , for all $J \in \tau$.

Our method sets communication boundaries for I_τ by computing the *earliest relative start time* (ES_τ) and the *latest relative finishing time* (LF_τ) of a task τ based on \mathcal{S} .

Definition 3: Earliest Relative Start Time (of a Task). Let τ be a task in schedule \mathcal{S} . The earliest relative start time (ES_τ) of τ is the minimum relative start time among all jobs of τ in \mathcal{S} .

$$ES_\tau = \min_{\forall J \in \tau} S_J \quad (3)$$

Definition 4: Latest Relative Finishing Time (of a Task). Let τ be a task in schedule \mathcal{S} . The latest relative finishing time (LF_τ) of τ is the maximum relative finishing time among all jobs of τ in \mathcal{S} .

$$LF_\tau = \max_{\forall J \in \tau} F_J \quad (4)$$

In order to exemplify definitions 1 to 4, we show in Figure 2 a schedule \mathcal{S} for three tasks that are part of a CEC E , $E = (\tau_1 \rightarrow \tau_2 \rightarrow \tau_3)$. In this example, we analyze task τ_2 , which has three jobs in \mathcal{S} . Following definitions 1 and 2, the first job of τ_2 , $J(1)$, has $S_{J(1)} = 2$ and $F_{J(1)} = 3$, while $J(2)$ has $S_{J(2)} = 0$ and $F_{J(2)} = 1$. $J(3)$ has $S_{J(3)} = 1$ and $F_{J(3)} = 2$. Following definitions 3 and 4, $ES_{\tau_2} = 0$ and $LF_{\tau_2} = 3$. By using definitions 1 to 4, ES_τ and LF_τ for the other tasks in E , $ES_{\tau_1} = 0$ and $LF_{\tau_1} = 1$ for τ_1 , while $ES_{\tau_3} = 1$ and $LF_{\tau_3} = 2$ for τ_3 .

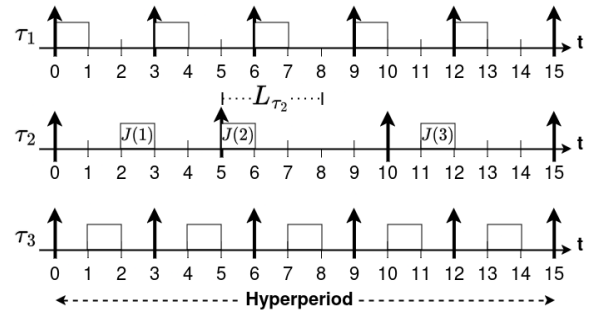


Fig. 2: Schedule \mathcal{S} for a CEC $E = (\tau_1 \rightarrow \tau_2 \rightarrow \tau_3)$

Based on the values of ES_τ and LF_τ , our proposed method shifts and shortens the communication intervals of a given task τ . It repositions the communication intervals of τ by shifting them according to ES_τ . For instance, it sets the new phase of τ to be: $\phi_\tau = \phi'_\tau + ES_\tau$, ϕ'_τ is τ 's initial phase. Our method shortens the length of τ 's communication intervals according to LF_τ . Since the boundaries of I_τ are relative points in time with respect to τ and our proposed method shifted τ

according to ES_τ , $begin(I_\tau) = 0$ and $end(I_\tau) = LF_\tau - ES_\tau$. Therefore, by setting $I_\tau = [begin(I_\tau), end(I_\tau)]$, and $L_\tau = I_\tau$, our method shortens and shifts the communication intervals of τ . For instance, for task τ_2 shown in Figure 2, instead of setting $L_{\tau_2} = [0, T_{\tau_2}]$, i.e., $[0, 5]$, our method sets $L_{\tau_2} = I_{\tau_2} = [0, 3]$. Note that in this example τ_2 is not shifted because $ES_{\tau_2} = 0$.

In Figure 3, we show the communication boundaries of interval $L_{J(i)}$ for a given job $J(i)$ of task τ assuming $L_\tau = I_\tau$.

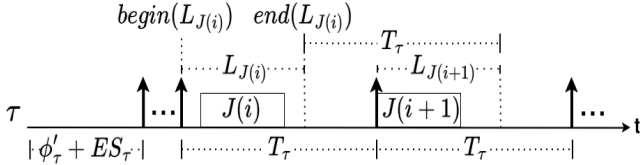


Fig. 3: Communication boundaries of interval $L_{J(i)}$ for a given job $J(i)$ of task τ assuming $L_\tau = I_\tau$

Note that although our method adds a phase ES_τ to a task τ , neither the schedulability of the task set nor jobs' execution order in schedule \mathcal{S} are affected: for any $J(i) \in \tau$, $i \in \mathbb{N}^+$, there is no $J(i)$ that executes before $(i-1)T_\tau + ES_\tau$ according to \mathcal{S} . All J of τ have to wait at least ES_τ time units after its release in order to execute. Therefore, as long as our method postpones the release of τ by ES_τ time units, $\forall \tau \in \Gamma$, our method does not affect the schedulability of the task set and preserve the execution order of jobs in schedule \mathcal{S} .

B. Computing Communication Points

Since the logical read and write-events of J happen at well defined points in time, it is possible to identify the communication points where data propagates from one task to the other. Martinez et al. [6] presented an analysis to compute the communication points between two tasks applying the LET model assuming that $|L_\tau| = T_\tau$, $\forall \tau \in \Gamma$. As our method shortens and shifts communication intervals, the assumption does not hold anymore and the analysis is no longer applicable.

Inspired by the work done by Martinez et al. [6], we present a new analysis to compute the communication points of tasks applying the LET model assuming that $L_\tau = I_\tau$ in theorems 1 and 2. Below we define the terms *publishing point* and *reading point*, which are later used in theorems 1 and 2.

Definition 5: Publishing Point. Given a pair of tasks in a CEC E , where $\tau_i \rightarrow \tau_{i+1}$, $i \in \{1, 2, \dots, |E| - 1\}$. Let a publishing point ($P_{\tau_i, \tau_{i+1}}^n$) be the n^{th} point in time where the resource shared by τ_i and τ_{i+1} is updated by τ_i . After $P_{\tau_i, \tau_{i+1}}^n$, no other logical write-event of τ_i will take place before the next logical read-event of τ_{i+1} .

Definition 6: Reading Point. Given a pair of tasks in a CEC E , where $\tau_i \rightarrow \tau_{i+1}$, $i \in \{1, 2, \dots, |E| - 1\}$. Let a reading point ($Q_{\tau_i, \tau_{i+1}}^n$) be the n^{th} point in time where the resource shared by τ_i and τ_{i+1} is read by τ_{i+1} . The logical read-event of τ_{i+1} after the n^{th} publishing point of τ_i is the reading point $Q_{\tau_i, \tau_{i+1}}^n$.

Theorem 1: Let $\tau_i \rightarrow \tau_{i+1}$ be a pair of tasks applying the LET model with schedule-aware intervals in a CEC E , where $T_{\tau_i} \leq T_{\tau_{i+1}}$, $i \in \{1, 2, \dots, |E| - 1\}$. Then the reading and publishing points between τ_i and τ_{i+1} can be computed as:

$$Q_{\tau_i, \tau_{i+1}}^n = nT_{\tau_{i+1}} + \phi_{\tau_{i+1}} \quad (5)$$

$$P_{\tau_i, \tau_{i+1}}^n = \left\lfloor \frac{Q_{\tau_i, \tau_{i+1}}^n - \phi_{\tau_i} - end(L_{\tau_i})}{T_{\tau_i}} \right\rfloor T_{\tau_i} + \phi_{\tau_i} + end(L_{\tau_i}) \quad (6)$$

$$n \geq \begin{cases} 0, & \text{if } \phi_{\tau_{i+1}} \geq \phi_{\tau_i} + end(L_{\tau_i}) \\ \left\lceil \frac{\phi_{\tau_i} + end(L_{\tau_i}) - \phi_{\tau_{i+1}}}{T_{\tau_{i+1}}} \right\rceil, & \text{otherwise} \end{cases}$$

Proof: We prove this theorem in two steps.

Step 1 (Reading point): Since $T_{\tau_i} \leq T_{\tau_{i+1}}$, there is always one job of τ_i being released between two job releases of τ_{i+1} . That means, τ_i always updates the resource shared with τ_{i+1} before each logical read-event of τ_{i+1} . By definition (Section III-B), the inputs of τ_{i+1} are logically updated at $begin(L_{\tau_{i+1}})$, which occurs every $T_{\tau_{i+1}}$ time units after $\phi_{\tau_{i+1}}$. Therefore, a reading point between τ_i and τ_{i+1} occurs every $nT_{\tau_{i+1}} + \phi_{\tau_{i+1}}$.

Step 2 (Publishing point): By definition (Section III-B), the logical write-event of the *first* job of τ_i logically occurs at $\phi_{\tau_i} + end(L_{\tau_i})$. That means, any reading point $Q_{\tau_i, \tau_{i+1}}^n$ has to be \geq than $\phi_{\tau_i} + end(L_{\tau_i})$. If $Q_{\tau_i, \tau_{i+1}}^n = \phi_{\tau_i} + end(L_{\tau_i})$, then $P_{\tau_i, \tau_{i+1}}^n = Q_{\tau_i, \tau_{i+1}}^n$. If $Q_{\tau_i, \tau_{i+1}}^n > \phi_{\tau_i} + end(L_{\tau_i})$, the publishing point that immediately precedes $Q_{\tau_i, \tau_{i+1}}^n$ depends on how many logical write-events of τ_i happened within the interval $[\phi_{\tau_i} + end(L_{\tau_i}), Q_{\tau_i, \tau_{i+1}}^n]$. Since logical write-events of τ_i occur periodically according to T_{τ_i} , the number of logical write-events within the considered interval is $\left\lfloor \frac{Q_{\tau_i, \tau_{i+1}}^n - \phi_{\tau_i} - end(L_{\tau_i})}{T_{\tau_i}} \right\rfloor$. Hence, the publishing point that immediately precedes $Q_{\tau_i, \tau_{i+1}}^n$ is at $\left\lfloor \frac{Q_{\tau_i, \tau_{i+1}}^n - \phi_{\tau_i} - end(L_{\tau_i})}{T_{\tau_i}} \right\rfloor T_{\tau_i} + \phi_{\tau_i} + end(L_{\tau_i})$. ■

Theorem 2: Let $\tau_i \rightarrow \tau_{i+1}$ be a pair of tasks applying the LET model with schedule-aware intervals in a CEC E , where $T_{\tau_i} > T_{\tau_{i+1}}$ and $i \in \{1, 2, \dots, |E| - 1\}$. Then the publishing and reading points between τ_i and τ_{i+1} can be computed as:

$$P_{\tau_i, \tau_{i+1}}^n = nT_{\tau_i} + \phi_{\tau_i} + end(L_{\tau_i}) \quad (7)$$

$$Q_{\tau_i, \tau_{i+1}}^n = \left\lfloor \frac{P_{\tau_i, \tau_{i+1}}^n - \phi_{\tau_{i+1}}}{T_{\tau_{i+1}}} \right\rfloor T_{\tau_{i+1}} + \phi_{\tau_{i+1}} \quad (8)$$

$$n \geq \begin{cases} 0, & \text{if } \phi_{\tau_{i+1}} \leq \phi_{\tau_i} + end(L_{\tau_i}) \\ \left\lceil \frac{\phi_{\tau_i} + end(L_{\tau_i}) - \phi_{\tau_{i+1}}}{T_{\tau_{i+1}}} \right\rceil, & \text{otherwise} \end{cases}$$

Proof: We prove this theorem in two steps.

Step 1 (Publishing point): Since $T_{\tau_i} > T_{\tau_{i+1}}$, there is always one job of τ_{i+1} being released between two job releases of τ_i . That means, τ_{i+1} always reads the resource shared with τ_i after each logical write-event of τ_i . By definition (Section III-B), the outputs of τ_i are logically updated at $end(L_{\tau_i})$, which occurs

every T_{τ_i} after $\phi_{\tau_i} + \text{end}(L_{\tau_i})$. Therefore, a publishing point between τ_i and τ_{i+1} occurs every $nT_{\tau_i} + \phi_{\tau_i} + \text{end}(L_{\tau_i})$.

Step 2 (Reading point): By definition (Section III-B), the logical read-event of the *first* job of τ_{i+1} logically occurs at $\text{begin}(L_{\tau_{i+1}})$, i.e., $\phi_{\tau_{i+1}}$. That means, any reading point $Q_{\tau_i, \tau_{i+1}}^n$ has to be \geq than $\phi_{\tau_{i+1}}$. By intuition, if $P_{\tau_i, \tau_{i+1}}^n \leq \phi_{\tau_{i+1}}$, then $Q_{\tau_i, \tau_{i+1}}^n = \phi_{\tau_{i+1}}$. If $P_{\tau_i, \tau_{i+1}}^n > \phi_{\tau_{i+1}}$, the reading point that immediately succeeds $P_{\tau_i, \tau_{i+1}}^n$ depends on how many logical read-events of τ_{i+1} happened within the interval $[\phi_{\tau_{i+1}}, P_{\tau_i, \tau_{i+1}}^n]$. Since logical read-events of τ_{i+1} occur periodically according to $T_{\tau_{i+1}}$, the number of logical read-events within the considered interval is $\left\lceil \frac{P_{\tau_i, \tau_{i+1}}^n - \phi_{\tau_{i+1}}}{T_{\tau_{i+1}}} \right\rceil$. Hence, intuitively, the read point of $\tau_i \rightarrow \tau_{i+1}$ that immediately succeeds $P_{\tau_i, \tau_{i+1}}^n$ is at $\left\lceil \frac{P_{\tau_i, \tau_{i+1}}^n - \phi_{\tau_{i+1}}}{T_{\tau_{i+1}}} \right\rceil T_{\tau_{i+1}} + \phi_{\tau_{i+1}}$. ■

In Section V, we show how to compute the E2E latencies of a given CEC using theorems 1 and 2.

V. COMPUTING END-TO-END LATENCIES

In Section IV we presented a method that exploits information from a schedule and define new communication intervals for tasks applying the LET model. Since our method shortens and shifts the communication intervals, we derived theorems 1 and 2 to identify the points in time when data propagates from one task to another in a CEC. In the following, we demonstrate how to identify which job chains have to be investigated during the E2E analysis of a multi-rate CEC. We also show how to compute the maximum reaction time and data age latencies related to those job chains.

A. Identifying Job Chains

Due to the under/oversampling nature of multi-rate CECs, some job chains might have jobs in common. For instance, when multiple actuators (outputs) of a CEC E are based on the same sampled (input) data, multiple job chains have the same $J(i)$ as their J_1 in E . As mentioned in Section II, when analyzing the data age of a multi-rate CEC, it is necessary to know for how long a sampled value affects the actuation of the CEC, i.e., know the time interval between two consecutive job chains with different $J(i)$ as their J_1 . In order to distinguish job chains that have different $J(i)$ as their J_1 , below we define the term *primary job chain*.

Definition 7: Primary Job Chain. Given a set of job chains that have the same $J(i)$ as their J_1 . We call *primary job chain* (pc_i^E), the job chain with the *earliest* $\text{end}(L_{J_{|E|}})$ in the set. $pc_i^E = (J_1 \rightarrow J_2 \rightarrow \dots \rightarrow J_{|E|})$ represents the i^{th} primary job chain of E , $i \in \mathbb{N}^+$. Given a prime job chain pc_i^E , we represent the next prime job chain of E after pc_i^E as pc_{i+1}^E . That is, pc_{i+1}^E is the first job chain after pc_i^E that has a different $J(i)$ as J_1 . Function $pc_i^E(k)$ returns the communication interval $L_{J(k)}$ of the k^{th} job in the given primary job chain, $k \in \{1, 2, \dots, |E|\}$. Function $l(pc_i^E)$ return the time interval between $\text{end}(pc_i^E(|E|))$ and $\text{begin}(pc_i^E(1))$.

In order to identify the primary job chains of a given CEC E , our method first identifies the job chains of E using

algorithms 1 and 2. As shown by Leung et al. [23], for a task set with periodic tasks and offsets, a schedule \mathcal{S} repeats itself every $LCM(T_{\tau_1}, \dots, T_{\tau_{|E|}})$ units of time. The repetition starts at: $\Phi(E) + H(E)$, $\Phi(E) = \max(\phi_{\tau_1}, \dots, \phi_{\tau_{|E|}})$ and $H(E) = LCM(T_{\tau_1}, \dots, T_{\tau_{|E|}})$. Günzel et al. [10] showed that for the sake of computing the maximum reaction time and data age latencies of a given CEC E applying the LET model, it is enough to analyze the job chains within one of the repetition intervals after the *warm-up* period. The warm-up period covers the time interval required for an input to fully traverse E for the first time [10].

In order to identify which jobs are part of the job chains within one of the repetition intervals, our method applies Algorithm 1 to all jobs of $E(|E|)$ released within the repetition interval. The process of identifying jobs chains starts with the $J_{|E|}$ that has the *earliest* $\text{begin}(L_{J_{|E|}})$ within the repetition interval. Our method starts the analysis with the last communication task pair in E , i.e., $\tau_{|E|-1} \rightarrow \tau_{|E|}$. By applying Algorithm 1 to a given $J_{|E|}$ of $E(|E|)$, our method obtains the publishing point related to a $J_{|E|-1}$.

Algorithm 1 Compute publishing points

Input: $\text{begin}(L_{J_i}), \tau_{i-1}, \tau_i$

- 1: **if** $T_{\tau_{i-1}} \leq T_{\tau_i}$ **then**
- 2: According to Theorem 1
- 3: **else**
- 4: According to Theorem 2
- 5: **end if**
- 6: Find m_{\max} , the largest value of m such that $P_{\tau_{i-1}, \tau_i}^m \leq \text{begin}(L_{J_i})$

Output: P_{τ_{i-1}, τ_i}^m

Using the output of Algorithm 1 as an input to Algorithm 2, our method computes the publishing and reading points of the previous communication task pair in E , i.e., $\tau_{|E|-2} \rightarrow \tau_{|E|-1}$. Note that the process of identifying a job chain‘ starts from its tail ($J_{|E|}$) and stops when its head (J_1) is found. Our method identifies the remaining part of the job chain by applying Algorithm 2 recursively to all the other communication task pairs in E until our method obtains the reading and publishing point related to a job J_1 of $E(1)$.

Algorithm 2 Compute reading points

Input: $P_{\tau_{i-1}, \tau_i}^n, \tau_{i-2}, \tau_{i-1}$

- 1: **if** $T_{\tau_{i-2}} \leq T_{\tau_{i-1}}$ **then**
- 2: According to Theorem 1
- 3: **else**
- 4: According to Theorem 2
- 5: **end if**
- 6: Find m_{\max} , the largest value of m such that $Q_{\tau_{i-2}, \tau_{i-1}}^m < P_{\tau_{i-1}, \tau_i}^n$
- 7: $n = m_{\max}$
- 8: Compute $Q_{\tau_{i-2}, \tau_{i-1}}^n$ and $P_{\tau_{i-2}, \tau_{i-1}}^n$
- 9: $i = i - 1$

Output: Q_{τ_{i-1}, τ_i}^n and P_{τ_{i-1}, τ_i}^n

Our method uses Equation 9 on each identified publishing point $(P_{\tau_k, \tau_{k+1}}^n, k \in \{1, 2, \dots, |E|-1\})$, in order to determine the i^{th} job of each task τ_k that is part of the communication task pairs in the CEC E . Note that reading points could also be used to determine the i^{th} job of each task τ_k , i.e., $i = 1 + \frac{Q_{\tau_k, \tau_{k+1}}^n - \phi_{\tau_k}}{T_{\tau_k}}$.

$$i = 1 + \frac{P_{\tau_k, \tau_{k+1}}^n - \text{end}(L_{\tau_k})}{T_{\tau_k}} \quad (9)$$

Following Definition 7, our method selects, within the repetition interval, the primary job chains among the set of job chains identified using algorithms 1 and 2. Our method adds the selected primary job chains to a set ζ and use it to compute the maximum reaction time and data age latencies.

B. Computing the Maximum Reaction Time and Data Age Latencies

As discussed in Section V-A, in order to compute de maximum reaction time and data age latencies of a given CEC E , our method first needs to identify all primary job chains of E within one of the repetition intervals after the warm-up period. In this section we show how to compute the E2E latencies of a CEC E given the set of primary job chains within a repetition interval.

We follow the definitions of maximum reaction time and data age latencies used by Günzel et al. [10]. As mentioned in Section III, we consider an additional delay z between the occurrence of an external event (input) and its sampling by J_1 . In the same manner, we consider that z' represents an additional delay between $\text{end}(L_{J_{|E|}})$ and the actuation (output). Therefore, in order to compute the maximum reaction time and data age latencies as done by Günzel et al. [10], we append z to the beginning of a primary job chain pc^E and z' to its end, i.e., $pc^E = (z, J_1, \dots, J_{|E|}, z')$. Despite the fact that the maximum latency values of the reaction time and data age are equivalent [10], in the following we also demonstrate how to calculate individually the intermediate values of those two latency metrics.

1) **Maximum Reaction Time:** For a given CEC E and its set of primary job chains ζ , our method computes the reaction time for all $pc_i^E \in \zeta$ considering the maximum delay z that an incoming input could suffer. In the worst case, for a primary job chain pc_i^E , an input arrives right *after* the logical read-event of the first job of pc_i^E . As a result, the incoming input has to wait until the logical read-event of the next primary job chain (pc_{i+1}^E) in order to be recognized and propagated through E . We consider, as Günzel et al. [10], that actuation takes place at the logical write-event of $pc_{i+1}^E(|E|)$, i.e., $z' = 0$.

Equation 10 shows how to compute the reaction time of a given primary job chain pc_i^E assuming the maximum input delay.

$$RT(pc_i^E) = z + l(pc_{i+1}^E) + z' \quad (10)$$

where:

$$z = \text{begin}(pc_{i+1}^E(1)) - \text{begin}(pc_i^E(1)) \text{ and } z' = 0$$

The maximum reaction time of a CEC E is:

$$MRT(E) = \max_{\forall pc_i^E \in \zeta} RT(pc_i^E) \quad (11)$$

2) **Maximum Data Age:** For a given CEC E and its set of primary job chains ζ , our method computes the data age for all $pc_i^E \in \zeta$ considering the maximum delay z' that an actuation output could suffer. In the worst case, for a primary job chain pc_i^E , the last actuation based on a given input happens right *before* the logical write-event of the last job of pc_{i+1}^E . We consider, as Günzel et al. [10], that the input data on which the outputs are based is read during the logical read-event of $pc_i^E(1)$, i.e., $z = 0$.

Equation 12 shows how to compute the data age of a given primary job chain pc_i^E assuming the maximum output delay.

$$DA(pc_i^E) = z + l(pc_i^E) + z' \quad (12)$$

where: $z = 0$ and $z' = \text{end}(pc_{i+1}^E(|E|)) - \text{end}(pc_i^E(|E|))$

The maximum data age of a CEC E is:

$$MDA(E) = \max_{\forall pc_i^E \in \zeta} DA(pc_i^E) \quad (13)$$

VI. PRACTICALLY ENFORCING DETERMINISTIC COMMUNICATION POINTS

The LET model assumes that tasks' inputs and outputs are logically updated at the beginning and end of their communication intervals. However, platforms are not infinitely fast to realize such behavior. Therefore, the deterministic ordering and atomic execution of these updates must be enforced to preserve the desired logical behavior. At the implementation level, the LET model is enforced by using hardware/software mechanisms [24] [17]. In the literature, different ways to implement the logical behavior of the LET model have been proposed [2] [5] [17] [18] [25].

One way to implement the logical behavior of the LET model is by implementing auxiliary tasks, which are responsible for updating the inputs and outputs of the tasks [2] [17]. For instance, the implementation of a task τ would consist of three tasks: (i) a reader (*copy-in*) task τ^R ; (ii) an execution task τ^E ; (iii) a writer (*copy-out*) task τ^W . Hereafter, we discuss one of the possibilities of implementing the logical behavior of the LET model when shortening and shifting communication intervals. Note that other implementations respecting the logical behavior and the deterministic ordering of execution of the tasks could be used.

At the beginning of L_τ , the auxiliary reader task (τ^R) copies all the input data necessary for τ^E 's execution to a local variable. At the end of L_τ , the auxiliary writer task (τ^W) copies τ^E 's output data to the shared variable that will be accessed by the next task in the CEC. Therefore, tasks τ^R and τ^W are responsible for updating the inputs and outputs of τ at the boundaries of L_τ .

Since the auxiliary reader (resp. writer) task has to be executed as close as possible to the start (resp. end) of L_τ , τ^R and τ^W have to be characterized by a very short WCET and a

very high priority level [17]. Therefore, the correct positioning of τ^R and τ^W is important in order to achieve the expected logical behavior of τ [2].

We use the parameters of τ and its communication interval L_τ to set the parameters of τ^R , τ^E and τ^W . The auxiliary tasks τ^R and τ^W are periodic task with period T_{τ^R} (resp. T_{τ^W}) equal to T_τ . The correct positioning of τ^R and τ^W is done by means of an additional phase. Therefore, τ^R has a phase $\phi_{\tau^R} = \phi_{\tau^E} = \phi'_\tau + ES_\tau$, while τ^W has a phase $\phi_{\tau^W} = \phi'_\tau + LF_\tau$. We consider that the execution times of the copy operations done by τ^R and τ^W are very short and their overhead included in C_τ . The parameters $(C_{\tau^E}, T_{\tau^E}, D_{\tau^E})$ of τ^E are equal to the parameters of τ .

Figure 4 shows how the auxiliary tasks τ^R and τ^W of a given task τ can be modeled. Note that in Figure 4 $\tau^E = \tau$.

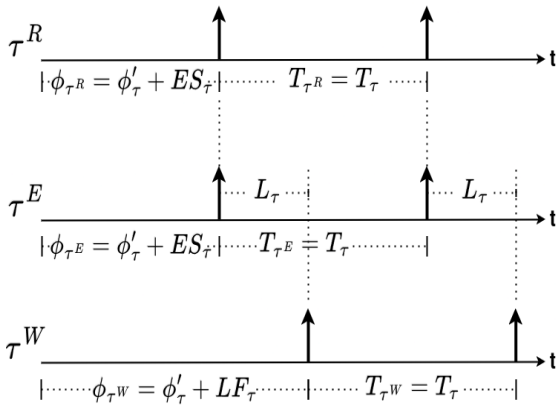


Fig. 4: Enforcing deterministic communication points by means of auxiliary tasks

The copying operations carried out by the auxiliary tasks (τ^R and τ^W) are facilitated via highest priority interrupts, which are added for each periodic auxiliary task. These interrupts are activated with the same period as the tasks they correspond to, allowing them to execute immediately. In order to obtain *fresh* data values, write output operations are favored over read input operations. Therefore, the write output operations are given the highest priority in the system, whereas read input operations are given the second highest priority.

VII. EXPERIMENTAL RESULTS

We evaluate our work based on the Real World Automotive Benchmarks presented by BOSCH [8] and synthetic task sets. We compare our method with the approach presented by Bradatsch et al. [7] and the LET model. The method proposed by Bradatsch et al. [7] will be referenced as the WCRT-LET model. We consider that the task sets run on a system comprising four cores and that tasks are scheduled according to rate monotonic.

A. Real World Automotive Benchmarks

We generated and tested 500 schedulable task sets based on the parameters of the Real World Automotive Benchmarks [8]. We assign periods to tasks following the definitions of Table III in [8]. The range of possible periods is: [1, 2, 5, 10, 20, 50, 100, 200, 1000]ms. Note that the sum of the probabilities for possible periods in [8] is 85%. The remaining 15% is for angle-asynchronous tasks. Since, we do not consider angle-asynchronous tasks, we divided all probability values by 0.85. Inter-task communications follow the definitions of Table II in [8]. For each task we generated a WCET following the definitions of Tables IV and V in [8].

As stated in [8], in typical engine control applications there are between 30 and 60 CECs. In our experiments, on average, there are 38 CECs per task set. The number of periods per CEC is randomly chosen between the interval [1,3] following the definitions of Table VI in [8]. For each period that composes the CEC, there are 2 to 5 tasks with that same period (Table VII in [8]). Each CEC is composed of 2 to 15 tasks. The total utilization of cores is $\approx 71\%$ (per core), on average.

In figures 5 and 6, we show the maximum reaction time and data age results respectively. The box plots show the 25th percentile, average, 75th percentile and the maximum & minimum-case values. Note that in Figure 5, we normalized the results with respect to the maximum reaction time obtained by the LET model, while in Figure 6 we normalized the results with respect to the maximum data age.

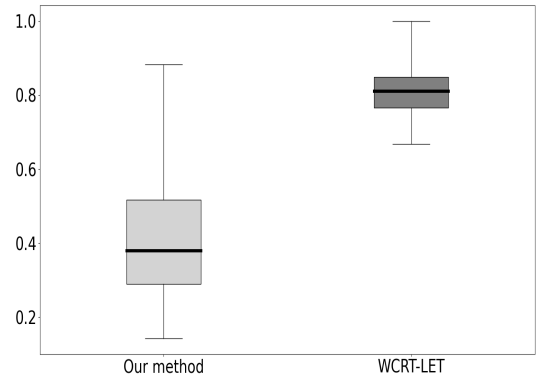


Fig. 5: Normalized maximum reaction time w.r.t LET

As shown in Figure 5, our model managed to obtain maximum reaction time values that are on average, $\approx 63\%$ lower than the values obtained by LET. Similarly, the WCRT-LET model managed to improve the maximum reaction time values by $\approx 20\%$.

As recently shown by Günzel et al. [10], values for the maximum reaction time and data age are equivalent. Figure 6 shows the results for the maximum data age. On average, our method obtained maximum data age values that are $\approx 63\%$ lower than the values obtained by LET. The WCRT-LET model improved the maximum reaction time values by $\approx 20\%$.

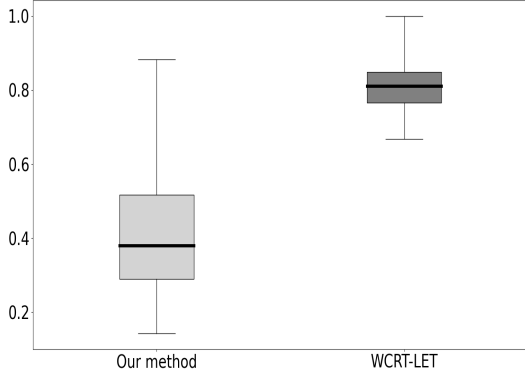


Fig. 6: Normalized maximum data age w.r.t LET

Table VI in [8] shows that 70% of the CECs present in the benchmarks are single-rate. Since our method shortens and shifts the communication intervals of the tasks, it makes possible for incoming inputs to propagate through the CEC within one repetition interval rather than in multiple as with $|L_i| = T_i$. As a result, in some cases our method resulted in an improvement exceeding 80% of that achieved using the LET model as shown in figures 5 and 6.

B. Synthetically Generated Workloads

We randomly generated 500 schedulable task sets, where we chose task periods and inter-task communication as in the previous experiment. However, this time we allowed tasks to have higher WCET values, increased the number of possible periods per CEC from 3 to 5 and reduced the probability of single-rate CECs from 70% to 7%. The 63% difference was split equally among the other possible number of periods. The new probabilities for possible number of periods per CEC are {1: 7%, 2: 35.75%, 3: 25.75%, 4: 15.75%, 5: 15.75%}.

As a result of increasing the number of possible periods in a CEC, we increased the total number of tasks composing it from 15 to 25. We chose randomly the amount of CECs per task set from interval [10, 20], with an average of 13 CECs per task set. The total utilization of cores is $\approx 80\%$. The obtained results are summarized in figures 7 and 8.

As shown in figure 7 and 8, our method outperformed WCRT-LET and the LET model once again. For synthetic task sets, our model managed to obtain maximum reaction time values that are on average, $\approx 67\%$ lower than the values obtained by LET, while the WCRT-LET model managed to improve the maximum reaction time values by $\approx 20\%$.

In Figure 9, we analyze a single task set to further show how much improvement our method achieves over the LET model with respect to the maximum data age. We randomly selected one task set from the 500 schedulable task sets to analyze. The selected task set has 12 CECs and a total of 107 tasks distributed on the 4 cores. For example, the CEC with ID=9 has 18 tasks and 5 periods [20, 50, 100, 200, 1000]ms. Figure

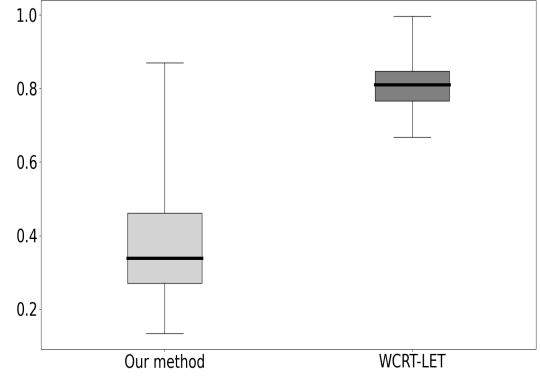


Fig. 7: Normalized maximum reaction time w.r.t LET

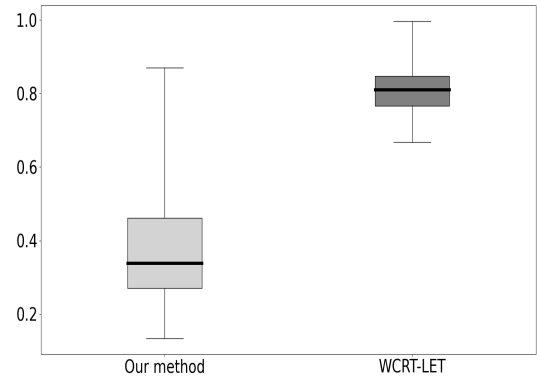


Fig. 8: Normalized maximum data age w.r.t LET

9 shows that for the CEC with ID=9, our method improved the maximum data age value by $\approx 76\%$, while the WCRT-LET model improved it by $\approx 20\%$.

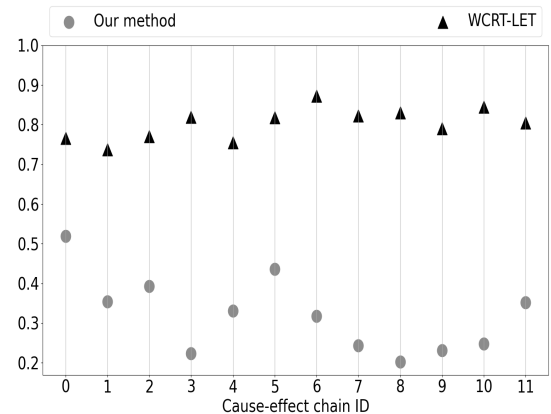


Fig. 9: Comparison of the normalized maximum data age values of the CECs present in a randomly selected task set

VIII. CONCLUSION

In this paper, we proposed a method to obtain less pessimistic end-to-end latencies of multi-rate cause-effect chains applying the LET model, by considering knowledge of the schedule in later design phases of safety critical applications.

Our method shortens and shifts communication intervals by exploiting schedule information, while maintaining the deterministic characteristics of the LET model and tasks' periodicity.

Experiments showed that for task sets based on the Real World Automotive Benchmarks presented by BOSCH [8] or randomly generated, our method produced less pessimistic end-to-end latencies than previous works. On both test case scenarios, it obtained end-to-end latency values that are, on average, $\approx 65\%$ lower when compared to the LET model.

If needed, e.g., for legacy reasons, our method does not have to be applied to the entire task set, but also to only a subset.

In future work, we plan to investigate the impact of our method in cause-effect chains containing different execution models and how it can influence the end-to-end latencies.

ACKNOWLEDGMENTS

We would like to thank Denis Claraz, Vitesco Technologies France S.A.S, and the anonymous reviewers for their feedback and thorough comments. We are also grateful for all the feedback provided by the members of the Real Time Systems Chair at the University of Kaiserslautern-Landau.

REFERENCES

- [1] S. Matic and T. A. Henzinger, "Trading end-to-end latency for composability," in *26th IEEE International Real-Time Systems Symposium (RTSS'05)*. IEEE, 2005, pp. 12–22.
- [2] A. Hamann, D. Dasari, S. Kramer, M. Pressler, and F. Wurst, "Communication centric design in complex automotive embedded systems," in *29th Euromicro Conference on Real-Time Systems (ECRTS 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.
- [3] M. Verucchi, M. Theile, M. Caccamo, and M. Bertogna, "Latency-aware generation of single-rate dags from multi-rate task sets," in *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2020, pp. 226–238.
- [4] T. A. Henzinger, B. Horowitz, and C. M. Kirsch, "Giotto: A time-triggered language for embedded programming," in *International Workshop on Embedded Software*. Springer, 2001, pp. 166–184.
- [5] J. Martinez, I. Sanudo, P. Burgio, and M. Bertogna, "End-to-end latency characterization of implicit and let communication models," in *International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, Dubrovnik, Croatia, 2017.
- [6] J. Martinez, I. Sañudo, and M. Bertogna, "Analytical characterization of end-to-end communication delays with logical execution time," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2244–2254, 2018.
- [7] C. Bradatsch, F. Kluge, and T. Ungerer, "Data age diminution in the logical execution time model," in *International conference on Architecture of computing systems*. Springer, 2016, pp. 173–184.
- [8] S. Kramer, D. Ziegenbein, and A. Hamann, "Real world automotive benchmarks for free," in *6th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, vol. 130, 2015.
- [9] N. Feiertag, K. Richter, J. Nordlander, and J. Jonsson, "A compositional framework for end-to-end path delay calculation of automotive systems under different path semantics," in *IEEE Real-Time Systems Symposium: 30/11/2009-03/12/2009*. IEEE Communications Society, 2009.
- [10] M. Günzel, H. Teper, K.-H. Chen, G. von der Brüggen, and J.-J. Chen, "On the equivalence of maximum reaction time and maximum data age for cause-effect chains," in *35th Euromicro Conference on Real-Time Systems (ECRTS 2023)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2023.
- [11] T. Klaus, F. Franzmann, M. Becker, and P. Ulbrich, "Data propagation delay constraints in multi-rate systems: Deadlines vs. job-level dependencies," in *Proceedings of the 26th International Conference on Real-Time Networks and Systems*, 2018, pp. 93–103.
- [12] J. Forget, F. Boniol, and C. Pagetti, "Verifying end-to-end real-time constraints on multi-periodic models," in *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE, 2017, pp. 1–8.
- [13] M. Becker, D. Dasari, S. Mubeen, M. Behnam, and T. Nolte, "Synthesizing job-level dependencies for automotive multi-rate effect chains," in *2016 IEEE 22nd International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE, 2016, pp. 159–169.
- [14] M. Dürr, G. V. D. Brüggen, K.-H. Chen, and J.-J. Chen, "End-to-end timing analysis of sporadic cause-effect chains in distributed systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 18, no. 5s, pp. 1–24, 2019.
- [15] J. Schlatow, M. Mostl, S. Tobuschat, T. Ishigooka, and R. Ernst, "Data-age analysis and optimisation for cause-effect chains in automotive control systems," in *2018 IEEE 13th international symposium on industrial embedded systems (SIES)*. IEEE, 2018, pp. 1–9.
- [16] M. Günzel, K.-H. Chen, N. Ueter, G. von der Brüggen, M. Dürr, and J.-J. Chen, "Timing analysis of asynchronized distributed cause-effect chains," in *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2021, pp. 40–52.
- [17] A. Biondi, P. Pazzaglia, A. Balsini, and M. Di Natale, "Logical execution time implementation and memory optimization issues in autosar applications for multicores," in *International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, 2017.
- [18] P. Pazzaglia, A. Biondi, and M. Di Natale, "Optimizing the functional deployment on multicore platforms with logical execution time," in *2019 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2019, pp. 207–219.
- [19] T. Kloda, B. d'Ausbourg, and L. Santinelli, "Towards a more flexible timing definition language," in *12th International Workshop Quantitative Aspects of Programming Languages and Systems-at ETAPS 2014*, 2014.
- [20] W. Pree, J. Templ, P. Hintenaus, A. Naderlinger, and J. Pletzer, "Tdl-steps beyond giotto: A case for automated software construction," *Int. J. Softw. Informatics*, vol. 5, no. 1-2, pp. 335–354, 2011.
- [21] M. Becker, D. Dasari, S. Mubeen, M. Behnam, and T. Nolte, "End-to-end timing analysis of cause-effect chains in automotive embedded systems," *Journal of Systems Architecture*, vol. 80, pp. 104–113, 2017.
- [22] A. Kordon and N. Tang, "Evaluation of the age latency of a real-time communicating system using the let paradigm," in *ECRTS 2020*, vol. 165. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2020.
- [23] J. Y.-T. Leung, Merrill, and ML, "A note on preemptive scheduling of periodic, real-time tasks," *Information processing letters*, vol. 11, no. 3, pp. 115–118, 1980.
- [24] K.-B. Gemlau, L. Köhler, R. Ernst, and S. Quinton, "System-level logical execution time: Augmenting the logical execution time paradigm for distributed real-time automotive software," *ACM Transactions on Cyber-Physical Systems*, vol. 5, no. 2, pp. 1–27, 2021.
- [25] A. Biondi and M. Di Natale, "Achieving predictable multicore execution of automotive applications using the let paradigm," in *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2018, pp. 240–250.

Session Tu.4.C

Security

Tuesday 11th June

17:00

–

Pastel Room

Security by Default - CHERI ISA Extensions Coupled with a Security-Enhanced Ada Runtime

P. Butcher | D. King | J. Kliemann

Abstract

In an age where security breaches and cyberattacks have become increasingly prevalent, the need for robust and comprehensive security mechanisms within embedded real-time systems is paramount. We propose a novel solution to enforce fault-detection and increase security assurance: "Security by Default", specifically combining Capability Hardware Enhanced RISC Instructions (CHERI) ISA microprocessor extensions with a CHERI pure-capability compliant Ada runtime. We present case studies showing how combining memory-safe hardware with memory-safe software results in a mutualistic layered approach to security and increases assurance of embedded real-time systems. We argue that this satisfies regulatory security verification objectives outlined in standards like the "Airworthiness Security Process Specification" (DO-326A/ED-202A [1] [2]).

Keywords: Cyber-security, CHERI, Ada, Airworthiness Security, Memory-safety, embedded real-time systems

1 Introduction

As the UK's National Cyber Security Centre (NCSC) aptly states, "Secure by Default" is defined as "technology which has the best security it can without you even knowing it's there or having to turn it on." [3]. This principle served as the guiding philosophy of our research as we set out to evaluate the security assurance claims being made over the adoption of a CHERI compliant microprocessor and a CHERI pure capability runtime environment that understands how best to benefit from the CHERI extended instruction set architecture (ISA). More specifically, this paper describes the development steps and subsequent evaluation of a security-hardened Ada runtime executing on Arm's Morello CHERI extended ISA microprocessor [4]. The goal of the research was to demonstrate and evaluate a layered approach to security that avoids common failure modes and provides security with significantly reduced effort.

Whilst at first glance it may seem unnecessary to implement a CHERI pure-capability compliant runtime for a memory-safe programming language [5], our research shows that the two technologies are complementary, and although there are overlaps in memory safety checks, the limitations of one approach are overcome by the feature set of the other. This paper details why combining a memory-safe programming language and runtime with a memory-safe microprocessor results in a security framework upon which developed embedded real-time systems are resilient to attack and capable of attack recovery.

This paper assesses security claims about CHERI regarding the benefits of fine-grained memory protection, and the enabling of granularity in memory access controls that, while not entirely novel (other historical solutions have come and gone), are claimed to elevate security assurance levels of software executing on CHERI hardware. CHERI offers execution security through dynamic fine-grained memory protection checks, which offer a different approach than other microprocessor security features like trust zones and secure boot [6]. This paper documents our research findings around the security assurance impact when systems utilize CHERI to precisely define which portions of memory are accessible and which are off-limits. This evaluation includes measuring the potential reduction of a system's attack surface, CHERI's overall ability to minimize memory-based vulnerabilities (for example, buffer overflows and data injection attacks), and the impact reduction following unauthorized electronic interaction.

Moreover, this paper quantifies the reduction in risk of privilege escalation and unauthorized data access. CHERI's object capabilities grant programs the ability to manage and control access to their data structures and resources with precision. This level of control can be utilized by systems to enact robust isolation between components.

Our research shows that CHERI adoption can not only provide a security layer to applications written in a memory-unsafe language like C but also provide a benefit to the adoption of applications written in a memory-safe language like Ada or Rust [5]. Thus CHERI can enhance critical systems, offering high resilience against security threats. This paper explores the integration of CHERI ISA extensions into high-integrity embedded systems and answers the question: Can CHERI bring extended memory protection and isolation, thereby enhancing the security posture of high-integrity, real-time embedded systems?

Ada, as a high-integrity programming language, is focused on supporting high-assurance, safety-critical, and embedded real-time systems. Over the years, Ada has demonstrated its value in various domains, including aerospace [7], defense [8], rail [9], space [10] and multiple other safety-critical applications. Its success is attributable to several key attributes, including its robust type system, rigorous runtime checking, a history of reliability and the availability of qualifiable freely licensed open source tooling. Historically, these characteristics have made Ada a valuable language for critical systems where maintainability and safety are paramount.

This paper introduces and evaluates a security-enhanced Ada runtime that extends the freely licensed open source GNAT Pro Ada runtime environment [11] with a tailored set of security features that align with the CHERI architecture. To assess the security benefits of coupling CHERI hardware with the Ada programming language, we have developed spatially safe and CHERI pure-capability compliant memory allocators within the GNAT Pro Ada runtime. Spatial safety ensures that out-of-bounds memory accesses beyond the bounds of the allocated memory are detected. In addition, by leveraging Ada’s runtime exception handling, we have implemented a mechanism to propagate CHERI-hardware-detected memory vulnerabilities into software exception handlers. We have also assessed the different approaches the Ada language and CHERI have taken to bounds checking, and we argue that joint adoption provides a defense-in-depth approach.

1.1 Introduction to CHERI

CHERI is an extension of the RISC (Reduced Instruction Set Computer) architecture to enhance memory safety and security in computing systems. CHERI is a joint research project of SRI International and the University of Cambridge [12], CHERI introduces new instructions and architectural features to enable fine-grained memory protection and mitigate common security vulnerabilities.

CHERI aims to improve memory safety by providing fine-grained protection mechanisms, reducing the risk of memory-related vulnerabilities such as buffer overflows and dangling pointers. CHERI is a microprocessor ISA hardware security toolkit for developing high-assurance software runtime environments to secure application execution.

The CHERI instruction set architecture introduces security extensions to standard memory addresses (i.e. pointers) via the concept of capabilities. More specifically, CHERI capabilities extend standard pointers with: a capability tag used to define the validity of the capability, a specification of the bounds of the accessible memory region within the address space, a set of permissible actions related to the memory and an object type field used for *sealing* capabilities (making them immutable and non-dereferenceable).

CHERI enforces access control policies that could be missed at the software level by directly incorporating processing checks of capabilities into the hardware architecture, reducing the attack surface for malicious exploits. In addition, CHERI was designed to be compatible with existing software, allowing for gradual adoption and integration into existing computing systems without requiring significant changes to software development practices [12].

The CHERI Tag extension is a 1-bit value that defines the validity of the memory address. Any attempt to dereference invalid capabilities will result in a CHERI processor exception.

The hardware will raise a capability bounds fault hardware exception if an instruction attempts to dereference a capability when the associated virtual address is outside the configured capability bounds. In addition, the capability tag is automatically cleared by the CPU architecture when instructions attempt to change the configuration of a capability in

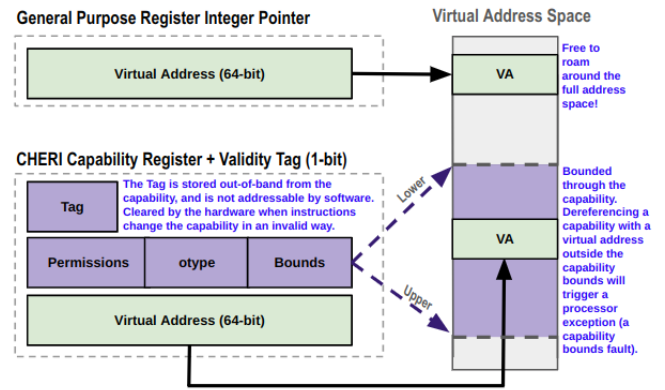


Figure 1: General Purpose Register Integer Pointer vs CHERI Capability

an invalid way (for example, when trying to *increase* a capability’s bounds). In addition, CHERI supports the concept of fine-grained memory protection by enforcing capability inheritance. Capability inheritance ensures that capabilities cannot be created out of thin air; instead, capabilities are derived from other valid capabilities and inherit the parent capability’s bounds and permissions. Capability inheritance is also monotonic; capabilities cannot increase their bounds or permissions beyond those inherited from the parent capability. This policy allowed us to build an Ada runtime fine-grained memory protection model that adheres to the principle of least privilege. Figure 1 shows the memory layout and some of the enforcement policies made available via capabilities.

CHERI introduces new instructions for working with capabilities, such as loading and storing capabilities in memory, restricting the bounds and permissions of capabilities, and sealing and unsealing capabilities. These instructions allow for the creation, manipulation, and enforcement of capabilities within the hardware architecture. Additionally, CHERI includes instructions for performing capability-based memory accesses, bounds checking, and permission checks to ensure secure and authorized access to memory regions.

Furthermore, CHERI introduces new registers dedicated to storing capabilities, such as the capability register file. These registers hold capabilities that represent specific memory regions and include metadata such as base address, bounds, and permissions. The capability register file provides a means for managing and manipulating capabilities within the hardware, enabling efficient enforcement of memory protection policies and access control mechanisms.

1.2 Introduction to the Ada Runtime

The Ada language provides a significant set of features including multitasking, exception handling and memory management. To provide these features it requires a runtime library. The runtime library is similar but not identical to a standard library seen in other languages. It provides both interfaces for the compiler and the programmer to use. Some parts of the library are intended to be used by programmers, such as I/O interfaces while others are used indirectly like the multitasking interface.

The runtime library and the compiler work together to provide the complete set of Ada features to the programmer. Features like tasking or returning variable sized objects from functions are defined in the language itself but require runtime support. If one of these features is used, the source code itself will not contain any direct reference to the runtime. Instead the compiler will generate the required code to call the runtime library transparently to the user. The following paragraphs will go through the major features in more detail.

The Ada language and its runtime, support a feature called elaboration. It is used to ensure the initialization of all global objects in the proper order. It also takes care of initializing the runtime itself before starting the program execution. This feature is implemented as an additional step in the compilation process. After compilation, the compiler will call the binder. The binder evaluates the dependencies of both the program and the runtime features it uses and creates the proper initialization code. It will also detect dependency issues in the initialization such as circular dependencies. After the code generation the compilation process will continue similar to other languages.

The runtime also handles a part of the memory management that is typically not present in other languages. Ada supports returning variable sized objects from functions without requiring a heap. In a compiled language the compiler in part takes care of managing the stack. While code can use the stack for objects whose size is known only at runtime it can only do so on the top of the stack. When a function is called the memory for its return value is stored on the stack before the new stack frame is created. This requires knowledge about the size of the returned data. Otherwise the return value may overflow its allocated memory. Ada solves that by having a secondary stack. This secondary stack is not used for stack frames and therefore can be used for dynamically sized objects at runtime. If a function returns a variable sized object the compiler will emit code that uses the secondary stack to allocate memory for this object when returning the object. The stack implementation itself is part of the runtime library as this allows it to be adapted to the underlying system.

Tasking, or threading, in Ada is supported at the language level with the language providing a specific syntax and semantics for easy use by the programmer. It also allows an abstraction away from the underlying platform. Furthermore the language allows different tasking profiles providing different feature sets depending on the application. The compiler will generate expanded code specific to the chosen tasking profile and feature set which will call the runtime library providing the implementation specific to the underlying platform.

While the list of features highlighted here is not exhaustive it provides an overview of the specific features the Ada runtime library provides to the language, different from a typical standard library. The runtime provides the interface between the expanded code and the underlying system.

GNAT Pro provides a diverse set of runtime libraries for different targets and use cases. These range from the native runtime on native targets, which supports a wide selection of features, such as networking and file handling, to bare

metal runtimes for resource-limited targets, which still enable the use of features like tasking and the secondary stack without the need for an underlying operating system. Bare metal runtimes are typically deployed on embedded real-time systems.

1.3 Introduction to the Edge Avionics Project

The research described in this paper is funded by the Rapid Capabilities Office (RCO) of the UK Royal Air Force (UK RAF) via the Edge Avionics project ('Edge Avionics'). Edge Avionics is a consortium led on behalf of the RCO by the Defence Science and Technology Laboratory (Dstl, an executive agency of the UK Ministry of Defence (MOD) [13]) and delivered by GE Aerospace [14] (the prime), Wind River [15] and AdaCore [16]. The project aims to demonstrate a network of secure units running a distributed application at scale and capable of demonstrating resilience at the network level. A Dstl-owned and modified air platform mission system will be used to check the impact of the new security controls. Through Edge Avionics, the Edge Avionics consortium can substantiate CHERI security claims within a defense environment whilst investigating legacy software rework overheads.

2 GNAT Pro for Morello Ada: Security by Default for Embedded Real Time Systems

This paper presents the architectural details of multiple toolchains for GCC [17] and LLVM-targeted [18] bare-metal Morello systems. In all cases, we have focused on Ada cross-compiler configurations, basing them on the freely licensed open source GNAT Pro for the bare-metal product line, augmented to support an Ada application benefiting from CHERI features.

The toolchain focuses on using CHERI to enforce *spatial memory safety* between objects in memory. For each object in memory (whether on the stack, heap or statically allocated), a capability is created at runtime whose bounds are restricted to the size of the allocated object. Restricting the bounds prevents out-of-bounds access via a pointer to one object from being able to access another object. The compiler and runtime work together to ensure that all memory allocations are correctly bound and with the correct permissions. The compiler manages the allocation of objects on the primary stack, and the runtime manages heap and secondary stack allocations.

The compiler is also responsible for making sure that the bounds and permissions, obtained when an object is allocated, are subsequently used for all accesses made to the object or parts of it. That is a relatively easy task in Ada because the language maintains a strict separation between addresses (of objects) and offsets that may be added to these addresses during regular operations. In cases where this separation is broken, which can happen only by using very low-level devices, the compiler gives an explicit warning.

For objects on the primary stack, the compiler sets up a capability for each object on entry to the stack frame. Each capability is set up to point to the portion of the stack frame that is allocated for the object, with the bounds set to the

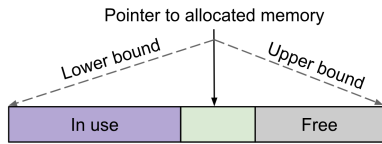


Figure 2: Memory allocation without spatial safety. The allocated pointer (capability) inherits the bounds to the entire heap/stack and can access other allocated objects

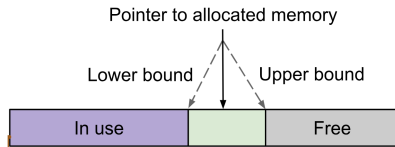


Figure 3: Memory allocation with spatial safety. The allocated pointer’s (capability) bounds are limited to the size of the allocated block. The pointer cannot be used to access other allocated objects

allocated region. These capabilities are inherited from the capability stack pointer (CSP) which prevents out-of-bounds access in case of a stack overflow.

The heap and secondary stack memory allocators in the runtime have been augmented to take advantage of CHERI and provide spatial safety between memory allocations. Each allocator has a capability to the entire block of memory assigned to the allocator, which is used to derive capabilities to fulfill allocation requests. For each allocation, the allocator returns a pointer to the allocated memory with the bounds limited to the size of the allocation to enforce spatial safety. The difference between the enforced bounding of the memory allocation is shown in figures 2 and 3.

For runtimes built with exception propagation enabled, the runtime implements a mechanism to catch CHERI processor exceptions and convert them into Ada exceptions that can be propagated, caught, and handled by user code. This is discussed further in section 3.2.1.

3 Security Assessment Evaluation

Fine-grained memory allocation is a foundational element of the developed security-enhanced Ada runtime. By implementing tightly bounded memory allocation, stringent memory safety checks can be performed by the CHERI hardware. Our research shows that fine-grained memory allocation prevents common vulnerabilities related to memory manipulation, such as buffer overflows and data corruption. These checks ensure that memory accesses remain within predefined bounds, minimizing the potential attack surface and acting as countermeasures to attacks that exploit memory vulnerabilities. Whilst a regular Ada runtime comes with protection over the safe usage of the Ada language, the security-enhanced Ada runtime also protects the parts of the language that are considered unsafe, like memory overlays.

3.1 CHERI Limitations

Overall, CHERI provides the means to increase security assurance. However, it does not guarantee it, furthermore, the benefits can only be realized with the correct usage of the

instructions and registers. More specifically, CHERI is meaningless without a CHERI-compliant software runtime. However, by integrating capabilities directly into the hardware architecture CHERI provides a solid foundation for developing demonstrably more secure software runtimes for building more secure and resilient computing systems.

In addition, the protection offered by CHERI is highly dependent on compiler configuration, and multiple factors must be considered when assessing the security assurance offered by adopting a CHERI-based CPU [19]. One major factor is pure-capability vs hybrid mode. The code uses CHERI capabilities for all pointers when configured in pure-capability mode. In comparison, hybrid mode allows a mix of standard RISC pointers and CHERI capabilities (typically configured via source-code annotation). Whilst hybrid mode has the benefit of integrating legacy systems with greenfield CHERI-enabled development, the downside is that we can no longer guarantee all pointers are capabilities, making security arguments harder to write.

Furthermore, our research has found that Ada code often requires low effort to port to CHERI, and in many cases, we have found that the codebases worked with no changes. Therefore, to focus our work on the highest levels of security assurance we developed bare-metal Ada CHERI compilers that only support pure-capability mode and enforce that all pointers (programmer or compiler generated) benefit from CHERI capability protections. As discussed later in this paper, this required extensive changes to the GNAT Pro Ada runtime [11].

By only supporting pure-capability CHERI, we enforce that all pointers are represented in memory as CHERI capabilities and that manipulation of a capability, and therefore the associated Ada object, can only be performed through capability instructions. Furthermore, the developed runtime must adhere to the strict CHERI rules around pointer integrity and monotonicity, specifically that valid capabilities cannot be created out of thin air; they must be derived from other, valid capabilities and the inherited bounds and permissions cannot be broadened. This results in the following list of unsupported Ada features which would require the creation of pointers (capabilities) to arbitrary memory addresses at runtime:

- `Ada.Tags.Internal_Tag` (used for Ada tagged types to convert an external tag - a string representation of an address - to an Ada tag, which is implemented as a pointer in GNAT)
- `Ada.Tags.Descendant_Tag` (depends on `Ada.Tags.Internal_Tag`)
- Reading Ada tagged objects from streams (via `S'Class'Input`, as this depends on `Ada.Tags.Descendant_Tag`)
- Reading addresses/pointers from streams (more specifically, Ada access types can be streamed to media, but we can not support streaming the data back into an access type)

This prerequisite would unlikely pose a significant issue for developing an Ada application on the presented solution. Furthermore, a high-integrity system, as often found in an embedded real-time system, would likely have been developed under strict guidelines prohibiting using these Ada features. In addition, for the `Ada.Tags.Internal_Tag` limitation, this is not a limitation of the Ada language, but rather reflects the way GNAT currently implements this feature. There are alternative implementations of `Ada.Tags.Internal_Tag` that would not need to create capabilities out of thin air.

3.2 Regular Ada software runtime constraint checks Vs CHERI runtime hardware checks

Ada provides various language-defined run-time checks to protect against detectable bugs such as out-of-bounds array accesses, range violations, integer overflow, and null pointer dereferences. Run-time check failures raise exceptions, which can be caught and handled by user-defined exception handlers to gracefully recover from the error whilst ensuring unsafe instructions are not executed. During the work to port the GNAT Pro to bare-metal Morello architecture, we evaluated the possibility of replacing these software run-time checks with the CHERI hardware run-time checks to reduce the overhead of the checks at run-time. We identified two kinds of run-time checks for consideration: null pointer checks and array bounds checks.

CHERI performs more robust pointer validity checks than Ada. CHERI verifies the validity of a pointer by inspecting the "tag bit" and ensures the control flow cannot dereference pointers with an invalid tag. By contrast, Ada's checks can only detect null pointers; non-null pointers that refer to an invalid memory location can still be dereferenced.

We evaluated using CHERI's bounds checking to implement Ada's semantics for array index checking. However, we found two issues that prevented CHERI from being able to implement the semantics required by Ada's language-defined checks [20]. The rules of the Ada programming language require raising a `Constraint_Error` exception before accessing an array with an index value that is not within the bounds of the array index type. This requires the bounds check to be precise, even for very large bounds. The first issue is that Morello architecture uses a compressed bounds format, ensuring the bounds are precise for objects up to 4 KB [21]. For objects larger than 4 KB, however, the bounds must be aligned to increasingly more significant powers of 2 address boundaries. This prevents the bounds for large objects from being represented precisely and thus requires padding in the memory allocation to align the capability bounds, meaning that CHERI will not detect minor accesses past the end of the array in the padding area. The second issue is that array types in Ada can be unconstrained where the array bounds are not known at compile time. This requires the array's bounds to be stored and passed alongside the array object at runtime, and this additional data must be within the bounds of the underlying CHERI capability. These two issues mean the CHERI bounds can be larger than the bounds checks enforced by the Ada language.

While Ada's compile-time and run-time checks ensure the correct usage of many parts of the language, some "unsafe" parts

do not have associated run-time checks and instead rely on the programmer to ensure correct usage. The term "unchecked" generally indicates Ada language features not covered by language checks, such as `Unchecked_Conversion`, which is used to cast between unrelated types. One particular feature considered unsafe is memory overlays, where an object is allocated (overlaid) at the same address as another object, introducing aliasing. There are no Ada language-defined checks associated with memory overlays, so it is up to the programmer to ensure that the two objects have compatible sizes and alignments and to avoid causing invalid data representations.

CHERI's hardware run-time checks cover this gap and provide memory safety for these unsafe parts of the language. For example, in the case of a memory overlay, the overlaid object inherits the capability of the target object. This ensures that any attempt to access beyond the bounds of the target object will be prevented at run-time.

3.2.1 Beyond Language-defined Run-time Checks

CHERI's hardware-level run-time checking provides additional memory safety assurances beyond language-defined run-time checks. The hardware checks apply to all code, including compiler-generated code that is not otherwise subject to run-time checks. This can reveal errors and defects in parts of the code that would otherwise go unnoticed and could potentially lead to exploitable security vulnerabilities.

While porting the GNAT Pro Ada compilers to Morello, we discovered a regression introduced in an unreleased development version of the compiler front-end that led to an out-of-bounds memory access. The regression occurred in a specific case where a function returns a variable-sized object whose size is known only at run-time. In this case, the compiler allocates memory on the secondary stack to store the returned object. The error was that the compiler used the wrong object size in the call to the secondary stack allocator, resulting in the allocation being too small to store the object. Subsequent memory accesses to the returned object could then access memory beyond the end of the allocation, potentially accessing other adjacent objects on the secondary stack which could have led to an exploitable vulnerability.

This regression was found only by running our existing test suites on Morello with our spatially safe secondary stack allocator. Running the same test suites on conventional architectures did not detect the regression. We also ran the test suites with Valgrind [22] and AddressSanitizer [23] which were also unable to detect the regression as all memory accesses were still within the bounds of the underlying buffer used for the secondary stack.

3.3 Enhanced Security over Language Bindings

Static checks completed by the compiler ensure a significant part of code correctness. While the programmer must define types and function signatures properly, the compiler can check for violations of these constructs. In C, this includes the `const` modifier. If used in a function signature, the caller can be sure that the passed value will remain unmodified throughout the call. While there are ways to circumvent this check, it has to be done explicitly. Without this circumvention, the compiler verifies the value remains unmodified inside the

function. Utilising the `const` modifier is a design decision that improves the understandability of the code and reduces the risk of wrong assumptions.

Ada employs a similar mechanism where function arguments can be specified with the modifiers `in`, `out`, and `in out`. Parameter mode `in` is the default and enforces that a value is only passed into the function and must not be modified. It is similar to the `const` modifier in C. Parameter mode `out` requires the callee to set the value, allowing the caller to assume initialization after the call. The modifier `in out` specifies that the caller must pass an initialized value, which the callee may modify. Additionally, when passing arrays into a function, the array value contains information about its bounds, allowing the compiler to insert runtime checks if required.

While the compiler can do many checks and prevent many problems, its scope is limited to the constructs of the language. However, the compiler only creates machine code from source code and does not connect the built parts of code, typically object files. The linker does this task. The linker, having only access to object files and symbols, cannot check types or even function signatures when linking object files into an application. It resolves symbols in object files with addresses, ensuring the correct place in the final binary is executed when the corresponding function is called.

This limitation is acceptable if the compiler can check types and signatures. Still, it stops working when multiple languages are used in the same project and foreign function interfaces are invoked for calls between languages. A foreign function interface is a feature that allows the programmer to tell the compiler that a particular function is imported and not defined in the same language. Imported means that the compiler will keep references to the imported function unresolved and instead expect the linker to resolve them. For these interfaces to be used, the programmer must declare the imported function to have the same arguments, argument types, argument modes and a calling convention as defined in the other language. It cannot check whether the imported declaration in one language conforms to the exported definition in another.

Consider the following C and Ada code:

```
void print_string(const char *s, size_t len){
    for(size_t i = 0; i < len; i++){
        putchar(s[i]);
    }
}
```

This C function prints a string by iterating over its characters and printing each of them. The input string `S` is passed by pointer and is declared `const`, forbidding the function from modifying it. The following Ada program will import the function and use it to print a string:

```
with Interfaces.C; use Interfaces.C

procedure Main is
    procedure Print_String(S : char_array; Len : size_t) with
        Import,
        Convention => C,
        External_Name => "print_string";
    Some_String : constant char_array := "Hello_World!";
```

```
Backup_String : constant char_array := Some_String;
begin
    Print_String (Some_String, Some_String'Length);
    pragma Assert (Some_String = Backup_String);
end;
```

The Ada program imports the C function using a matching function signature. By telling the compiler `Convention => C`, it will know that the string is passed by a pointer and will prepare the arguments accordingly. To ensure that both Ada and C use compatible function signatures the Ada code uses C types defined in the `Interfaces.C` package. The assertion checks that both strings are equal after the call (i.e., that they have the same contents). The assertion check should never equate to false, as both strings are equal on creation and constant.

But what happens when `Print_String` is modified to change the string it writes to the console? If it was implemented in Ada the new declaration would be:

```
procedure Print_String(S
    : in out char_array; Len : size_t);
```

With this change, the compiler will complain when `Some_String` is passed as it cannot be modified due to being a constant. However, this function is implemented in C, likely in a separate library. Its new implementation looks as follows

```
void print_string(char *s, size_t len){
    for(size_t i = 0; i < len; i++){
        putchar(s[i]);
        s[i] = 0;
    }
}
```

If the change is not identified during development, the Ada compiler will continue to assume that the string is not modified and base its checks on that assumption. Identification of the C code change is manual and error-prone; the Ada compiler cannot distinguish between the different C code bases as it does not parse and incorporate the C code. By design, the linker, responsible for connecting the compiled Ada and C code, does not understand types and calling conventions and will, therefore, also miss this inconsistency. The result is a program that does not behave according to the programmer's intention, even though it should be according to the code and compiler. Furthermore, if the implementation is erroneous, the C function may overflow the buffer provided, corrupting the caller's stack.

While CHERI does not solve the problem statically, it can introduce runtime checks beyond what the compiler can typically do. The caller of a function creates capabilities matching the permissions required by the arguments the caller is expecting to call the callee with. In this example, the string is passed as a capability using the bounds of the string without write permission. The modified or erroneous C implementation may still try to violate these permissions. However, this now results in a CHERI exception. This approach can be employed for all data passed by reference. It does not cause the erroneous program to fail to compile, but it allows

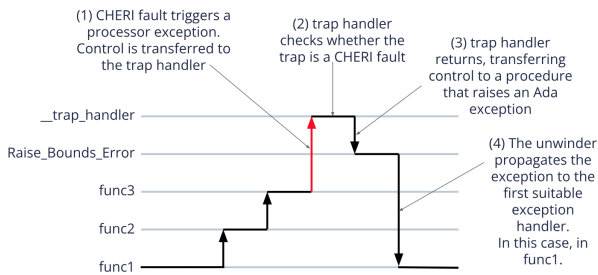


Figure 4: Converting a CHERI exception (raised in `func3`) to an Ada exception and propagating it up the call stack to the handler in `func1`

the problematic condition to be detected early through verification testing. Furthermore, it will enable the program to abort in a defined state or even recover from the error. As with many other improvements CHERI enables, this feature requires thoroughly applying the principle of least privilege in all pointers/capabilities passed to functions.

CHERI improves the situation; however, it does not solve the problem of foreign function interfaces in general. Calling conventions, type sizes, valid values, or even the number of arguments are still unchecked by the compiler. It does, however, apply to the bounds and permissions of data passed by reference, avoiding hard-to-debug memory corruptions or broken assumptions about the immutability of passed data.

3.4 Hardware detected Capability Fault Propagation and Recovery

One feature of the Morello bare-metal Ada runtime is its ability to convert CHERI hardware exceptions into Ada exceptions that can be propagated, caught, and handled by application exception handlers. In traditional bare-metal runtimes, hardware exceptions are handled by a top-level trap handler which typically aborts the entire program. By contrast, the Ada Morello bare-metal runtime implements a trap handler that first determines which kind of CHERI exception has occurred, then returns control back to the call stack that triggered the trap but with the return address altered to call a subprogram that raises an Ada exception. This effectively causes an Ada exception to be raised from the point in the call stack that triggered the CHERI exception. When the Ada exception is raised, the runtime unwinds the call stack until it finds a handler for the exception, at which point control is passed to the handler. This is illustrated in Figure 4 which shows the conversion and propagation of a CHERI exception across function calls.

This mechanism allows the software application to use conventional Ada exception handlers to detect, isolate, and recover from any CHERI exceptions that occur within that code block. In a multitasking environment, this also isolates the exception to the individual task that triggered it, allowing other tasks to continue execution unaffected. The ability to actively detect and respond to memory safety breaches allows the system to isolate compromised elements and initiate recovery procedures, enhancing fault-resilience. The affected system can fail in a “secure but degraded” manner, resulting in unaffected areas of the program being allowed to continue.

This form of fault-detection and recovery is useful when a denial of service attack could trigger a safety hazard or have implications over the availability of mission-critical systems.

3.5 Reduction in Exploitability

CHERI significantly improves application memory safety by checking for violations of memory boundaries. Many exploit techniques require such a violation to work, be it by inserting code directly into the attacked application or modifying the application’s state. These techniques typically require a good knowledge of the application’s internal memory, especially the used address ranges.

In order to counter memory corruption-based attacks, many mitigation mechanisms have been created both on the system and the application level. In applications, memory protection is often improved by inserting runtime checks to detect overflows and abort execution in case of detection. While this improves the application’s security and often allows recovery from the error, it also incurs a performance penalty. Additionally, the protection is limited to the code written in that language. Other system parts, sometimes even the runtime required to execute the application, are not protected.

At the system level, memory protection consists of mechanisms that increase the difficulty of successfully executing an attack. This approach may not prevent memory corruption; however, it makes it harder to take control and manipulate the application behavior. Address Space Layout Randomization (ASLR) is a technique that assigns random parts of the address space to the application. While an attack may still overflow a buffer, it is much harder to guess the correct address for the overflow. For example, triggering the application to jump to a specific address requires knowledge of the address space.

Another approach is to restrict the privileges of different memory regions. More specifically, the Write XOR Execute (W[^]X) principle. The assumption is that code is never modified by the program in its regular state and that modifiable memory, such as the stack and heap, are never used to execute code. While this also does not prevent memory corruption, it prevents placing and executing the attack payload. Even if the payload can be placed through memory corruption, it will likely be placed on the stack or in the heap where it cannot be executed. It also cannot be moved into executable memory as this region is not writable.

A technique to defeat this restriction is Return Oriented Programming. It uses the fact that some relevant metadata for the program execution is still placed on the stack and, therefore, in modifiable memory. This includes the return address, which controls the program’s execution when the current function returns. An exploit using this technique will overwrite the stack, especially the return addresses, with values that cause the CPU to jump to parts of the code that contain the functionality needed by the attacker. These parts, often called gadgets, can be only parts of functions. They can be chained together by writing multiple return addresses into the stack, and after a jump into a gadget has happened, the next jump will be executed once the gadget tries to return.

We have analyzed CHERI's resistance against Return Oriented Programming. While this approach still often includes an initial memory corruption, we assume that the attacker is able to manipulate the stack once at the beginning of the attack. While CHERI would typically prevent this initial condition from happening we wanted to know whether Return Oriented Programming could leverage an initial vulnerability to further exploit the system.

At first, we validated the W^X property of CHERI by creating examples that violate this principle while otherwise keeping all capabilities valid. Our tests have been executed and validated on an unprotected AArch64 platform. CHERI successfully rejected executing the stack by raising a capability permission fault. It also detected the code modification and raised a capability-sealed fault. A capability sealed fault is raised if a sealed capability is either dereferenced or has been modified before use. Sealed capabilities are used for capabilities referencing code. They cannot be modified or dereferenced but can be used as a jump target for the program counter.

Furthermore, we created two test cases to modify jump targets. The first test executed a common approach in ROP by overflowing a buffer on the stack and thereby overwriting the return address of the calling function. As all examples are written in Ada, the language caught that attack with a runtime check. Disabling runtime checks made the attack work on an AArch64 target. As expected on CHERI, this attack caused a capability-bound fault as soon as the program tried to write outside the allocated buffer on the stack.

The second test tried to manipulate the control flow more directly. It consists of a routine that takes a function pointer, adds an offset and calls the resulting function pointer. Adding an offset of zero should yield the same result as calling it without an offset while adding an offset greater than zero will cause the call to jump somewhere into the middle of the function or behind it. We generated the base function pointer from an existing function to ensure we start from a valid capability. On a non-CHERI target, this worked even with runtime checks enabled. Our CHERI solution detected the violation and raised a capability tag fault. While the created capability was valid, modifying it with an offset, even if it was zero, invalidated it. The reason is that CHERI uses sealed capabilities to represent function pointers, and sealed capabilities are immutable.

For an attacker that has access to all the information about the program but cannot modify its code directly, we conclude that a successful Return Oriented Programming attack is very unlikely, if not impossible. Even if many assumptions about CHERI, such as the bounds checks for capabilities, are invalidated, it still has additional layers of defense that prevent the unintended execution of the program's code. Even with the ability to manipulate the stack without triggering an exception, an attacker must replace the return address with a valid sealed capability.

Generally we notice that CHERI provides more than just resistance against memory corruptions. It also restricts the control flow into a narrow path, preventing deviations from

the programmer's intended functionality. Even if some of CHERI's fundamental properties are violated, the remaining constraints still prevent or at least increase the difficulty of an effective attack.

3.6 Performance

The Edge Avionics project aims to produce a demonstrator Avionics system that showcases the security benefits of CHERI. The final prototype is not intended for flight. Therefore, Arm's Morello development board is a good choice for the final target demonstrator platform. The Morello board microprocessor is a CHERI-enabled prototype CPU based on Arm's Neoverse N1, as found in the N1SDP evaluation board. As stated by Arm, "This is a high-performance superscalar, out-of-order pipeline design. The existing 64-bit Armv8.2-A support in the CPU was retained and support for the new Morello architecture was added" [24]. Performance metrics are essential to understanding the feasibility and impact of adopting a CHERI microprocessor architecture, particularly for high-integrity and safety/security-critical avionics. Our research through the Edge Avionics programme will contribute towards the final demonstrator platform and should the work be taken into commercial production, the Morello development board would not be selected as the final target hardware. An extensive performance analysis is only needed for the final target platform intended for flight, which is outside of the project's scope. CHERI microprocessors have yet to achieve airworthiness certification at the time of writing. However, the University of Cambridge Computer Laboratory has already completed extensive performance studies around the prototype Morello microarchitecture, and the results provide a strong argument that future, commercially available and fit-for-flight CHERI microprocessor solutions will be able to cope with modern-day demands of defence-related avionics. Cambridge writes, "results to date give us strong confidence that CHERI support can be tightly and cleanly integrated into future Arm architectures" [25]. The report stipulates that "the dynamic performance aims for Morello were to create a hardware design able to enable the evaluation of the usage of capabilities within rich established software ecosystems and to demonstrate their practical viability and security benefit." [25] Therefore, whilst the performance was a factor in the design of the Morello CPU, it is expected there is room for significant optimizations and that second and third-generation CHERI microprocessor architectures and subsequent hardware implementations will be higher performing in terms of execution speed, memory footprint and energy consumption. Cambridge backs up this claim by stating: "It is reasonable to project that the goal of 2%-3% overhead for deterministic spatial and referential memory safety is achievable with an optimized instruction-set architecture on a performance-optimized microarchitecture." [25]. The performance penalty predicted with future CHERI microprocessor architectures and subsequent compiler designs is acceptable, adding to the viability of the Edge Avionics project.

4 Airworthiness Security Methods and Considerations

Airworthiness is the discipline of ensuring air vehicles are safe. Airworthiness Security forms part of that same dis-

cipline, focusing on security aspects that, should they fail, would lead to safety hazards. More specifically, a security case is argued that claims system security risks do not lead to unacceptable safety risks. Regulatory organizations like the Federal Aviation Administration (FAA) in the U.S. and the European Union Aviation Safety Agency (EASA) have circulated advisories around the need to detect and prevent unauthorized electronic interactions within air vehicles, primarily to ensure existing and future air vehicles remain safe. Satisfying advisory circular requirements around unauthorized electronic interactions is required before the awarding of airworthiness certifications. At the time of writing, the FAA or EASA are not mandating a particular solution; however, industrial consortium working groups within the European Organisation for Civil Aviation Equipment (EUROCAE) and RTCA (previously known as the Radio Technical Commission for Aeronautics) have put considerable effort into two jointly developed sets of Airworthiness Security publications. Our research has focused on how the described "Security by Default" approach can help meet the objectives stated within the EUROCAE and RTCA "Airworthiness Security Methods and Considerations".

The European Organisation for Civil Aviation Equipment (EUROCAE) ED-203A "Airworthiness Security Methods and Considerations" [26] foreword states that ED-203A is technically identical to RTCA DO-356A [27], and this is also true of ED-202A [2] and RTCA DO-326A [1]. Furthermore, these standards and guidelines are equally applicable to the defense industry as well as the civilian aerospace industry. For example, first published in May 2023, the UK Ministry of Defence (MOD) announced new regulations for the Military Aviation Authority (MAA) [28]. The report includes reference to Regulatory Articles (RA) 5890 [29], which states: "The MAA recognises the risk assessment and mitigation process detailed in RTCA DO-326A / EUROCAE ED-202A and associated standards RTCA DO-356A / EUROCAE ED-203A as an acceptable means of compliance."

Our work has indicated three promising areas where our proposed solution to "Security by Default" can help satisfy Airworthiness Security objectives: Security Measures, Vulnerability logging and Refutation testing.

4.1 DO-326A/ED-202A Security Measure

To produce a convincing argument over the safe management of avionics security risk, we must show evidence that identifies all threat conditions and scenarios that could lead to loss of privacy, integrity or availability of identified security assets. Second, all attack paths must be understood and addressed. Applying risk treatment to an attack path amounts to allocating and assessing the effectiveness of one or more security measures and their ability to satisfy allocated security requirements. Where an attack vector involves memory safety vulnerabilities like a buffer overflow, we can argue a CHERI microprocessor architecture is a security measure that can reduce or stop the damage caused by the attack. Suppose the attack intends to expose a security asset within the system, i.e., violate a security requirement regarding asset privacy. In that case, correctly using CHERI's fine-grain memory protection will result in a high-assurance security measure; the attacker

may be able to trigger an exploit, but the hardware trap will detect the violation and guard against unauthorized memory reads/writes. The same feature provides a security measure that enforces the security asset's integrity; by bounding a memory address we ensure neighboring data is not overwritten and corrupted. In addition, whilst the security measure must still detect the event even if the attack only intends to cause disruption or loss of availability (for example, a denial of service attack), it must also satisfy security requirements that minimize or eradicate the loss of service, for example, recovery, isolation, or damage limitation. Our proposition described within this paper argues that the propagation of CHERI capability faults into Ada runtime exception handlers provides detection and countermeasure options to respond to the loss of service attack and, therefore, acts as an additional high assurance security measure.

4.2 Security Verification Objectives

The aim of refutation in the context of the Airworthiness Security Process is to refute the allegation of exploitable vulnerabilities [30]. The Airworthiness Security Process [2] [1] describes refutation as: "an independent set of assurance activities beyond analysis and requirements. As an alternative to exhaustive testing, refutation can be used to provide evidence that an unwanted behavior has been precluded to an acceptable level of confidence. NOTE: Refutation is also known as Security Evaluation in some contexts." [26] [27]. The refutation activities aim to identify any unexpected situations where the system would unexpectedly transition into a non-secure state (or, more generally, violate a minimal invariant guaranteeing the system's security) [30]. The difficulty with refutation testing is in the consistent and repeatable detection of the transition. Consistency and repeatability make it feasible to argue for an elevation in security assurance and, for Airworthiness Security, this must be adequately described within the Plan for Security Aspects of Certification (PsecAC). The PsecAC is the initial phase within the Airworthiness Security Process, and it is here that we set our security goals and how we intend to security test our application. Much like a "Plan for Safety Aspects of Certification" within the parent process "Software Considerations in Airborne Systems and Equipment Certification" (ED-12C and DO-178C [31]), integrators need to ensure regulatory authority accepts the plan before commencing with development and test phases.

Our approach enforces anomaly detection through regular Ada runtime constraint checks [20] and through the developed CHERI-hardware-enforced pure-capability Ada runtime. This dynamic verification feature captures unsafe memory instructions that would otherwise result in memory violations, such as out-of-bounds reads/writes. Not only can we isolate security assets in deployed systems, but we greatly enhance security verification testing as more anomalies can be detected. To understand why this is important, consider the resultant behavior of a standard (non-CHERI) CPU executing an application not using Ada runtime constraint checks. When a triggered software bug results in an out-of-bounds memory read or write instruction, the system could exhibit behavior that can be detected, for example, a segmentation fault may get signaled; however, it could equally go undetected

such that the system continues to operate but also transitions into a state where the security can no longer be guaranteed. The combination of an Ada pure-capability runtime executing on a CHERI microprocessor architecture eliminates this possibility; all out-of-bounds reads/writes will be captured by either the Ada runtime or the CHERI hardware. In both cases, the transition into a non-secure state will be visible to the verification suite so that the bug can be identified, logged and mitigated at a higher level in the safety plan or fixed and retested. It is also important to recognise the symbiosis of the pure-capability Ada runtime and the CHERI hardware capability checks; without the combination the guaranteed detection is lost and the quality of the refutation degraded.

The Airworthiness Security Process Guidelines in ED-203A and DO-326A [26] [1] state that refutation encompasses multiple disciplines, including "Dynamic Code Analysis". This specific refutation testing technique analyzes the system's behavior whilst the system is executing. An example of dynamic code analysis would be monitoring a non-safe or non-secure sequence of instruction calls made to the processor (i.e., detection of buffer overflows). Dynamic code analysis can be enforced within the semantics of programming languages via run time constraint checks or tools that detect memory corruption bugs via code instrumentation added during additional compilation passes [30]. Our research has shown that existing memory detection tools like Address Sanitizer [23] and Valgrind [22] can't detect the complete set of memory violations that our approach can (see section 3.4). In both cases, anomalies can only be detected when a test case triggers the scenario that exhibits the unwanted behavior. However, our research has uncovered sequences where Address Sanitizer and Valgrind fail to detect the transitions that our solution captures. Vulnerability identification is a critical aspect of any security process. As it is widely recognised that non-safe memory instruction calls form the basis of the majority of exploitable software bugs, being able to dynamically and consistently detect and guard against memory violations provides a security safety net should all other measures fail.

4.3 Vulnerability Logging / Fault-Recovery / Fail-Secure

In addition, our "Security by Default" research argues that propagating CHERI hardware detected faults into Ada software handlers makes it feasible to isolate system components such that fail-secure-but-degraded is possible. Without this feature, CHERI-based systems can still protect security assets. However, fault-recovery is only possible through intervention from a third-party monitoring system, such as a hypervisor. However, the main difference between this approach and the one proposed is that by dynamically detecting the impending violation at the point just before the failure condition, the state of the system, the triggering conditions and any other relevant information can be recorded within a security log file. Regulatory Article 1202 describes a framework approach for In-Service Air System Cyber Compliance. It is noted that this method is based on the requirements of the US National Institute of Standards and Technology (NIST) Cybersecurity Framework, which advocates the phases of "Identify", "Protect", "Detect", "Respond" and "Recover", note that the

National Cyber Security Centre (NCSC) also provides a Cyber Assessment Framework (CAF) that shares the principles of the NIST Framework. Two aspects of this methodology where the proposed solution plays a role are "Detect" and "Respond". "Detect" is described as being "introduced to enable timely detection of cyber security Incidents that may impact Air Safety, such as continuous monitoring and security log files" [29]. Therefore, capturing and isolating attacks is essential to satisfying the Detection requirements. In addition, the "Respond" phase is described as "once a cyber incident affecting Air Safety has occurred, the level of response is key in supporting the ability to contain the impact". This requirement is aimed at "business continuity plans" and "associated response plans" and having the ability to detect, capture, isolate and report the attack directly within the affected system is clearly beneficial.

4.4 Software Supply Chain Security

Modern-day large-scale systems often require collaborative efforts spanning large geographical regions that exacerbate the complexity around software supply chain security. The software supply chain is made up of all aspects of software development across all phases of the software development lifecycle. This includes development tools that have direct access to the source code and pose a risk to security assurance. A compiler's primary responsibility is to translate source code into machine code. Assuring that this translation is correct amounts to traceability studies that include binary-to-source code analysis [32]. Our work included porting a developed CHERI pure capability Ada runtime with different compiler back-ends, namely GCC [17] and LLVM [18]. Having more than one compiler solution is beneficial as it allows for novel software supply chain security verification techniques, like differential testing. Here, we argue that the integrity of the development tool is maintained by comparing it to the behavior of the alternative simply by feeding the same inputs into both, verifying the output, and observing the state. Voting algorithms are frequently used in high-integrity systems to increase the assurance of processed data. For example, flight control systems may sample data from multiple sensors and use algorithms to check the consistency and decide which value to use. The same argument can be applied to the security assurance of software development tools. However, this approach requires multiple independent solutions that, whilst the sampled data will be identical (i.e. the source code, in the case of a compiler), perform the same functionality and generate output that satisfies the translation requirements (the generated CPU instructions perform the functionality defined by the source code) with differing algorithmic designs. Developing both GCC [17] and LLVM [18] Ada Morello bare-metal compilers allows this argument to be made.

5 Further work

To further extend the approach described, we propose developing additional software runtime components that enhance the capabilities of CHERI hardware extensions. Security assurance can be further elevated by integrating features such as Temporal Memory Protection and Compartmentalization. Beyond spatial memory protection, software

can extend CHERI's capabilities to include temporal memory protection. Temporal Memory Protection helps prevent vulnerabilities like use-after-free errors, which is achieved through careful memory management and the use of capabilities to track and control memory lifecycles. Examples include tools like CHERIvoke [33] and Cornucopia [34] that characterize pointer revocation using CHERI Capabilities for Temporal Memory Safety. Compartmentalization is concerned with adding protection around untrusted libraries such that separate heap allocations are used and compartmentalized code can only access code or data in another compartment through a well-defined interface. Examples include CHERIoT [35]. In addition, support could be added for `Ada.Tags.Internal_Tag` in Morello GNAT which would remove the limitation on streaming Ada tagged types. Finally, future work will focus on the latest microprocessor architectures supporting CHERI. While the research conducted within this paper used Arm's Morello platform, the next phase of work will likely be on a CHERI-RISC-V CPU.

6 Conclusions

The paper summarizes research and development into a "Security by Default" approach to real-time embedded systems by leveraging the Arm Morello CHERI ISA extensions and a bare-metal security-enhanced Ada runtime. More specifically, a layered approach to security is described that demonstrates the benefits of memory-safe programming languages executing on memory-safe microprocessors. This combination allows Ada developers to benefit from an enhanced security toolchain and execution environment for high-integrity real-time systems. In addition, the paper proposes a fault resilience approach to bare-metal software security design by propagating CHERI hardware capability bounds exceptions into bare-metal application code exception handlers. Furthermore, our experience with CHERI has shown that it is an excellent verification target due to the advanced anomaly detection features of hardware capabilities and that porting Ada code to CHERI is often no effort. In addition, had it not been for this work, a security vulnerability could have made its way into deployed software, and our continuous integration suite now benefits from executing our Ada runtime regression tests on CHERI. Our work included analyzing the benefits of a CHERI pure-capability runtime and a CHERI-compliant microprocessor to airworthiness certification. As described in section 4, our developed solution could satisfy multiple security objectives; more specifically, it can be used as a deployed security measure guarding against high-security assurance level vulnerabilities and a dynamic analysis security verification tool for refutation testing. The results and insights presented in this research open additional avenues for strengthening the security of embedded real-time systems, ultimately contributing to safer, more reliable and more secure technology.

References

- [1] RTCA, "DO-326A - Airworthiness Security Process Specification," 12 2014. for the U.S. Federal Aviation Administration FAA.
- [2] EUROCAE, "ED-202A - Airworthiness Security Process Specification," 6 2014. for the European Union Aviation Safety Agency (EASA).
- [3] National Cyber Security Centre, "Secure by Default," *NCSC publications*, 2018. <https://www.ncsc.gov.uk/information/secure-default> [Accessed: (31/10/2023)].
- [4] Arm®, "Arm® Architecture Reference Manual Supplement Morello for A-profile Architecture," *Arm® Morello publications*, vol. A.k, 2022. <https://developer.arm.com/documentation/ddi0606/latest> [Accessed: (31/10/2023)].
- [5] National Security Agency, "Cybersecurity Information Sheet - Software Memory Safety," *National Security Agency Publications*, 2022. https://media.defense.gov/2022/Nov/10/2003112742/-1/-1/0/CSI_SOFTWARE_MEMORY_SAFETY.PDF [Accessed: (02/11/2023)].
- [6] Z. Ling, H. Yan, X. Shao, J. Luo, Y. Xu, B. Pearson, and X. Fu, "Secure boot, trusted boot and remote attestation for ARM TrustZone-based IoT Nodes," *Journal of Systems Architecture*, vol. 119, p. 102240, 2021.
- [7] Frédéric Pothon, Quentin Ochem, "AdaCore Technologies for DO-178C / ED-12C," *AdaCore publications*, 2017. <https://www.adacore.com/uploads/books/pdf/AdaCoreTechnologiesForDO178C-web.pdf> [Accessed: (31/10/2023)].
- [8] AdaCore, "White Paper - Disruptive Technology for Military-Grade Software," *AdaCore publications*, 2021. <https://www.adacore.com/uploads/techPapers/Developing-Military-Grade-Software.pdf> [Accessed: (31/10/2023)].
- [9] Jean-Louis Boulanger, Quentin Ochem, "AdaCore Technologies for CENELEC EN 50128:2011," *AdaCore publications*, 2018. <https://www.adacore.com/books/cenelec-en-50128-2011f> [Accessed: (31/10/2023)].
- [10] Benjamin M. Brosgol, Jean-Paul Blanquart, "AdaCore Technologies for Space Systems Software Supporting Qualification for ECSS-E-ST-40C and ECSS-Q-ST-80C," *AdaCore publications*, 2021. <https://www.adacore.com/uploads/books/pdf/AdaCore-Tech-Space-Systems.pdf> [Accessed: (31/10/2023)].
- [11] AdaCore, "AdaCore / bb-runtimes (Public)," 2024. <https://github.com/AdaCore/bb-runtimes> [Accessed: (03/11/2024)].
- [12] Robert N. M. Watson, Simon W. Moore, Peter Sewell, Peter G. Neumann, "An Introduction to CHERI," *UCAM-CL-TR-941, ISSN 1476-2986*, 2019. <https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-941.pdf>.

- [13] Defence Science and Technology Laboratory, “About Us,” 2024. <https://www.gov.uk/government/organisations/defence-science-and-technology-laboratory/about> [Accessed: (03/11/2024)].
- [14] GE Aerospace, “About Us,” 2024. <https://www.geaerospace.com/> [Accessed: (03/11/2024)].
- [15] Wind River, “Wind River: About Us,” 2024. <https://www.windriver.com/contact> [Accessed: (03/11/2024)].
- [16] AdaCore, “About Us,” 2024. <https://www.adacore.com/company/about> [Accessed: (03/11/2024)].
- [17] Richard M. Stallman et al., “Using the GNU Compiler Collection,” 2012. <https://gcc.gnu.org/onlinedocs/gcc-4.4.2/gcc/>.
- [18] C. Lattner and V. Adve, “LLVM: A compilation framework for lifelong program analysis and transformation,” in *CGO*, (San Jose, CA, USA), pp. 75–88, Mar 2004.
- [19] Nicolas Joly, Saif ElSherei, Saar Amar – Microsoft Security Response Center (MSRC), “Security analysis of CHERI ISA,” *MSRC-Security-Research/papers/2020*, 2020. <https://github.com/microsoft/MSRC-Security-Research/blob/master/papers/2020/Security%20analysis%20of%20CHERI%20ISA.pdf>.
- [20] AdaCore, “Ada Conformity Assessment Authority: Ada Reference Manual,” 2024. <http://ada-auth.org/arm.html> [Accessed: (03/11/2024)].
- [21] Jonathan Woodruff, Alexandre Joannou, Hongyan Xia, Anthony Fox, Robert Norton, Thomas Bauereiss, David Chisnall, Brooks Davis, Khilan Gudka, Nathaniel W. Filardo, A. Theodore Markettos, Michael Roe, Peter G. Neumann, Robert N. M. Watson, Simon W. Moore, “CHERI Concentrate: Practical Compressed Capabilities,” *IEEE Transactions on Computers* 68(10), 2019. <https://doi.org/10.1109/TC.2019.2914037>, <https://www.cl.cam.ac.uk/research/security/ctsrtd/pdfs/2019tc-cheri-concentrate.pdf>.
- [22] “Valgrind,” 2023. <https://valgrind.org/> [Accessed: (03/20/2024)].
- [23] Clang 19.0.0git documentation, “AddressSanitizer,” 2023. <https://clang.llvm.org/docs/AddressSanitizer.html> [Accessed: (03/20/2024)].
- [24] Arm, “Creating the Morello Technology Demonstrator,” 2022. <https://community.arm.com/arm-community-blogs/b/architectures-and-processors-blog/posts/creating-the-morello-technology-demonstrator>.
- [25] Robert N. M. Watson, Jessica Clarke, Peter Sewell, Jonathan Woodruff, Simon W. Moore, Graeme Barnes, Richard Grisenthwaite, Kathryn Stacer, Silviu Baranga, Alexander Richardson, “Early performance results from the prototype Morello microarchitecture,” September 2023. <https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-986.pdf>.
- [26] EUROCAE, “ED-203A - Airworthiness Security Methods and Considerations,” 6 2018. for the European Union Aviation Safety Agency (EASA).
- [27] RTCA, “DO-356A - Airworthiness Security Methods and Considerations,” 9 2014. for the U.S. Federal Aviation Administration FAA.
- [28] UK Ministry of Defence, “Cyber security for airworthiness: new MAA regulations,” 2023. <https://www.gov.uk/government/news/cyber-security-for-airworthiness-new-maa-regulations/> [Accessed: (03/20/2024)].
- [29] UK Gov, “RA 5890 – Cyber Security for Airworthiness and Air Safety – Type Design and Changes / Repairs to Type Design,” 2023. https://assets.publishing.service.gov.uk/media/656866f2cc1ec500138eef7b/RA5890_Issue_2.pdf [Accessed: (03/20/2024)].
- [30] Paul Butcher, “Guidelines and Considerations Around ED-203A / DO-356A Security Refutation Objectives,” *AdaCore Papers*, 2021. <https://www.adacore.com/uploads/techPapers/Guidelines-around-ED203A-and-DO356A-Security-Refutation-Objectives.pdf>.
- [31] RTCA / EUROCAE, “Software Considerations in Airborne Systems and Equipment Certification, DO178C/ED-12C,” 2011.
- [32] AdaCore, “Source Code to Object Code Traceability Study,” 2016. <https://www.adacore.com/uploads/books/pdf/traceability-sample.pdf>.
- [33] Hongyan Xia, Jonathan Woodruff, Sam Ainsworth, Nathaniel W. Filardo, Michael Roe, Alexander Richardson, Peter Rugg, Peter G. Neumann, Simon W. Moore, Robert N. M. Watson, Timothy M. Jones, “CHERivoke: Characterising Pointer Revocation using CHERI Capabilities for Temporal Memory Safety,” in *In Proceedings of the 52nd IEEE/ACM International Symposium on Microarchitecture (IEEE MICRO 2019)*, pp. 12–16, 2019. <https://www.cl.cam.ac.uk/research/security/ctsrtd/pdfs/201910micro-cheri-temporal-safety.pdf> [Accessed: (31/10/2023)].
- [34] Nathaniel Wesley Filardo, Brett F. Gutstein, Jonathan Woodruff, Sam Ainsworth, Lucian Paul-Trifu, Brooks Davis, Hongyan Xia, Edward Tomasz Napierala, Alexander Richardson, John Baldwin, David Chisnall, Jessica Clarke, Khilan Gudka, Alexandre Joannou, A. Theodore Markettos, Alfredo Mazzinghi, Robert M.

Norton, Michael Roe, Peter Sewell, Stacey Son, Timothy M. Jones, Simon W. Moore, Peter G. Neumann, Robert N. M. Watson, “Cornucopia: Temporal Safety for CHERI Heaps,” in *In Proceedings of the 41st IEEE Symposium on Security and Privacy (Oakland 2020)*, pp. 18–20, 2020. <https://www.ncsc.gov.uk/information/secure-default> [Accessed: (31/10/2023)].

[35] Amar, Saar and Chisnall, David and Chen, Tony and Wesley, Nathaniel Filardo and Laurie, Ben and Liu, Kunyan and Norton, Robert and Moore, Simon W. and Tao, Yucong and Watson, Robert N. M. and Xia, Hongyan, “CHERIoT: Complete Memory Safety for Embedded Devices,” in *proceedings of the 56th IEEE/ACM International Symposium on Microarchitecture*, Association for Computing Machinery, Oct. 2023.

Problems and New Approaches for Crypto-Agility in Operational Technology

Tobias Frauenschläger and Jürgen Mottok
Laboratory for Safe and Secure Systems (LaS³)
Technical University of Applied Sciences Regensburg
93053 Regensburg, Germany
{tobias.frauenschlaeger, juergen.mottok}@oth-regensburg.de

Abstract—In recent years, cybersecurity has also become relevant for Operational Technology (OT). Critical systems like industrial automation systems or transportation systems are faced with new threats, and therefore require the implementation of thorough security measures. Regulations further mandate the deployment and regular verification of these security measures. However, OT systems differ from well-known systems of classic Information Technology (IT), such as mission times spanning decades, infrequent updates only during on-site maintenance, or diverse devices with varying support for security measures.

The growing field of crypto-agility examines approaches to integrate security measures in an agile and flexible way, making updates easier and, therefore, encouraging a more frequent deployment of them. This paper contributes to this research field in the context of secure communication in two ways. We first examine the current state of crypto-agility by providing an overview of existing measures for OT systems. Then, we propose a new architecture concept with different deployment approaches to integrate security measures in a crypto-agile way. Based on a security library with a generic interface and a flexible proxy application, our architecture is capable of securing both new OT systems and existing ones via retrofit.

Keywords—Security, Crypto-Agility, Automation, Industrial Control Systems, SCADA, Transportation, Real Time, Communication Systems, Cryptography, Proxy, Gateway, Retrofit

I. INTRODUCTION

Cybersecurity is an emerging topic in the field of *Operational Technology* (OT). Nowadays, automation and control systems like *Industrial Control Systems* (ICS) are threatened by cyberattacks just like enterprise systems. This is especially important for automation systems within industrial processes or critical infrastructures, such as transportation or supply systems. Due to these increasing threats, various new regulations have been passed [1], [2] to improve resilience against cyber threats. Among various topics, operators of critical infrastructures in Germany have to prove the effectiveness of the implemented security measures to the Federal Office for Information Security (BSI) on a periodic basis. Due to the limited expected lifetime of cryptographic algorithms published by regulation bodies, e.g. by the BSI [3], and the vast amount of potential vulnerabilities [4] within these systems, deployed security measures have to be updated and maintained continuously.

Compared to IT enterprise systems, however, OT systems differ greatly in terms of updates and maintenance operations. Typical OT systems are designed with a mission time of up to 25 years, until the equipment is replaced on schedule. Software updates are installed only when strictly necessary during infrequent on-site maintenance (based on the common „never change a running system“ mentality). Hence, security measures can barely be updated or newly integrated after initial deployment. This inertial technological enhancement is aggravated by the systems consisting of devices from numerous manufacturers with vastly differing security capabilities.

As a result, the effort to create a secure system based on a common set of state-of-the-art measures and best-practices is complicated and expensive, if at all possible.

To overcome this problem, the arising research field of *crypto-agility* examines solutions to simplify the maintainability of security measures. Especially for the considered OT systems with their long service life, thorough measures of crypto-agility are key to future-proof the systems against cyberattacks and to increase the expected lifetime of the security features, and, hence, of the overall systems. In the first part of this work, we provide an overview of the current state-of-the-art regarding crypto-agility in OT systems, focusing on secure communication within OT systems. We analyze existing security measures for OT communication and then identify problems and limitations thereof, which could be improved by thorough crypto-agility capabilities. Based on the identified state-of-the-art, we propose a new solution concept to increase the level of crypto-agility in OT systems. A key aspect of our concept is to create a fast migration path for existing systems to retrofit security features with agility capabilities to counteract the described inertia. Lastly, we verify the viability of our concept in a case study to examine the impact on overall system performance metrics.

In summary, the paper makes the following contributions:

- 1) We give an overview of the current state of security measures within OT communication systems, elaborating on problems and limitations regarding crypto-agility capabilities.
- 2) Then, we present our concept to add crypto-agility capabilities into both existing and new OT systems with its various deployment approaches.
- 3) In a case study, we verify the viability of the concept and compare the different deployment approaches, with a focus on retrofitting existing systems.

The remaining paper is structured as follows. In Section II, the current state-of-the-art regarding crypto-agility in OT communication systems is presented. Based on a general overview of OT security and a definition of crypto-agility, current problems from both a software and a hardware perspective are discussed. Section III presents related work that already approaches the identified problems of current systems, and introduces approaches to improve crypto-agility comparable to ours. In Section IV, we present our concept and explore different approaches to actually deploy the concept within real OT systems. To verify the concept, the results of our case study are discussed in Section V. Finally, Section VI draws a conclusion and hints at future work.

II. CURRENT STATE-OF-THE-ART

In this chapter, we give an overview of security measures in OT communication systems and elaborate on problems

and limitations regarding crypto-agility. Initially, we briefly introduce OT communication systems (Part II-A) and present currently available security measures (Part II-B). Thereafter, we provide a definition of crypto-agility and elaborate the different branches of this research field for our work in Part II-C. Lastly, we analyze the identified problems and limitations of current measures from a software (Part II-D) and hardware perspective (Part II-E).

A. OT Communication Systems

Before we dive into the research field of crypto-agility, an overview of current OT communication systems is given. In general, an enterprise system can be split into multiple layers based on the *Purdue Enterprise Reference Architecture* (PERA) model [5], covering both its IT and OT domain. Makrakis et al. presented a modern adaptation of this model [6], depicted in Figure 1.

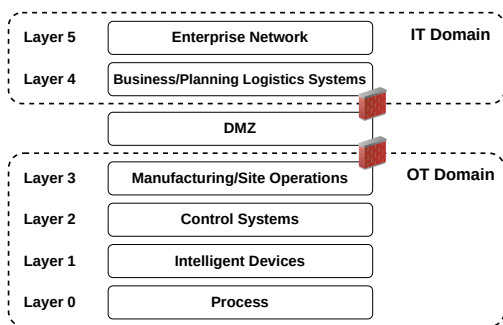


Figure 1. Modern adaptation of the Purdue Enterprise Reference Architecture (PERA) model, based on [6]. The layers are divided into IT and OT domain, separated by the DMZ (demilitarized zone) to secure the communication between the two domains.

Figure 1 shows the separation of IT and OT domains with their respective layers. Each layer implements a specific functionality of the overall system and further abstracts the underlying layers (e.g., a manufacturing process, or a complex transportation system). As the IT layers typically contain functionality that requires external communication (e.g. customer access or cloud services over the public internet), there is a strict isolation of the IT from the OT domain to protect the OT from malicious actors. Only absolutely necessary connections between IT and OT are allowed for the overall system to work, which are supervised and controlled by a special DMZ layer that integrates thorough security measures (indicated by the two firewall icons in Figure 1).

This separation and isolation of the OT domain with only very limited access from the IT is a key security aspect of current systems, as cyberattackers have to gain access to the OT before they can attack it. However, current developments regarding *IT/OT-Convergence* weaken this strict separation, and more connections are allowed to enable new and innovative functionality within the overall system (e.g. live monitoring of a process from the IT domain, predictive maintenance). Furthermore, OT systems grow more and more complex, resulting in a larger potential for configuration errors that ultimately lead to an increased number of entry points for attackers. On top of this, in many systems, cellular connections over public networks are used nowadays in addition to the wired network, breaking the isolation between the IT and OT domain. In total, the attack surface for the OT domain is growing. As a result, more thorough security measures have been created and deployed in OT systems, primarily derived from established IT security solutions.

Within the OT domain, each layer consists of different components connected by a communication system, autonomously handling a part of the system or process to abstract it to the next higher layer. This leads to a tree-like structure, with each layer being connected to the one above and below. Communication happens both vertically and horizontally. Vertical communication is primarily used by an upper layer for supervising and managing tasks of a lower layer. For tasks that are split onto multiple components, horizontal communication enables coordination and synchronization. Typically, the lower the layer, the greater the requirements for real-time behavior and reliability.

Within the OT layers, there exist both unicast point-to-point and multipoint communication flows, using various application protocols. Point-to-point connections are typically based on the client-server model, whereas the multipoint flows use a publish-subscribe model over a multicast system (also often referred to as producer-consumer). In both categories, application protocols building upon different layers of the OSI reference model can be found. Protocols building directly upon layer 2 (“L2” in the following) are used for time-critical communication, typically using Ethernet links. Application protocols building upon layer 4 (“L4” in the following) are rather used for less demanding connections regarding timing requirements. Among the most common point-to-point protocols are (non-exhaustive):

- OPC UA Client-Server (L4: TCP/IP)
- IEC 61850 MMS (L4: TCP/IP)
- Modbus TCP (L4: TCP/IP)
- Siemens S7 Comm (L4: TCP/IP)
- PROFINET RT (L2: Ethernet)

In the multipoint category, widely used protocols are (non-exhaustive):

- OPC UA Publish-Subscribe (L4: UDP/IP)
- MQTT (L4: TCP/IP)
- IEC 61850 GOOSE (L2: Ethernet)

Many of the application protocol specifications consist of multiple actual protocols with different characteristics for different use cases (e.g., an L2-based publish-subscribe protocol for real-time traffic together with an L4-based client-server protocol for supervisory functionality).

B. OT Security

A key security aspect of OT systems is the strict separation from the IT domain, with only very limited flows in between. The approach of separating subsystems where possible is further adapted within the OT domain by thoroughly utilizing network segmentation. Both within a layer of the PERA model and also across multiple layers, separated functional *islands* are formed with only very limited and properly secured connections between them. This approach is often further improved by adding virtual private network (VPN) technologies into the systems. These segmented system architectures limit the attack surface to a minimum, as lateral movement of a malicious actor is considerably restricted.

From a cryptographic perspective, most protocols have been adapted to support security measures to protect communication flows from malicious actors. Within the OT domain, common security goals to be achieved are:

- Availability (no malicious actor can prevent access to a system service for a legitimate entity)
- Integrity (transmitted data is protected against malicious modification)

- Entity Authenticity (a malicious actor cannot impersonate a legitimate entity)
- Data Authenticity (transmitted data originates from an authentic source)
- Confidentiality (a malicious actor is unable to eavesdrop sensitive information from the system)

Most of the protocols build upon well-established standards to achieve these goals. For example, L4-based protocols typically recommend using the *Transport Layer Security* (TLS) protocol to integrate security measures without a direct modification of the actual application protocol (e.g., for IEC 61850 MMS [7], for Modbus TCP [8], and also for the multipoint protocol MQTT for connections to a central message broker [9]). One notable L4-based exception is OPC UA, which does not build upon TLS, but rather integrates security measures directly into the application protocol [10]. Conceptually, however, the integrated cryptographic measures are very similar to the ones in TLS (same cryptographic algorithms, similar authentication mechanisms using digital certificates).

Multipoint protocols, however, cannot integrate typical security protocols like TLS, as those are designed for exactly two peers that perform a handshake during connection establishment. Within a multipoint group, however, there are typically more than two peers (one or more publishers, various subscribers), which all require the same cryptographic keys to apply the security measures. Hence, a key distribution mechanism is required, typically implemented using an additional node in the system. Each participating node in a specific multipoint group first contacts this *key server* to obtain the current group keys. This initial point-to-point connection between the node and the key server is secured using mechanisms already described above, both protecting the exchanged group keys and properly verifying the peer's legitimacy to join the group. For example, such mechanisms are specified for OPC UA [10] and for IEC 61850 GOOSE [7].

For L2-based protocols with real-time requirements, however, security measures are only cautiously specified and recommended, as the overhead of cryptographic operations may break the strict timing requirements. For instance, no cryptographic measures are officially specified or recommended for the PROFINET RT protocol by standard bodies due to this reason. However, research in this direction already proposed technically viable solutions to achieve both security and real-time behavior (e.g. in [11]).

In general, there are standardized security measures for various protocols available. Together with a properly designed, segmented system architecture, secure OT systems can be created and deployed today. However, to properly integrate the described security measures, all participating devices have to support the specific features. As already indicated in the introduction, OT systems and their devices are typically designed for mission times of up to a few decades, resulting in a slow deployment and adoption of new (security) features in existing systems. Furthermore, even if a completely secure OT system is newly deployed today, the ongoing development in the area of crypto-analysis and the increasing number and capabilities of cyberattackers necessitate ongoing effort to keep the initially achieved level of security over the long mission time. Hence, update capabilities and fast migration concepts are required to change and adapt security measures easily within deployed systems.

C. *Crypto-agility Definition*

Based on the literature works by Alnahawi et al. [12] and Mehrez & Omri [13], the following general definition of the term *crypto-agility* is derived: crypto-agility describes the capability of updating and replacing security measures during the lifetime of a component. Specifically, this means:

- Update the implementation of existing security measures (e.g. to fix a vulnerability)
- Update the list of supported cryptographic algorithms and their security parameters (e.g., add new algorithms, remove old ones, increase the key size)
- Incorporate and adapt to new functionality transparently (e.g. use hardware acceleration instead of a software implementation)
- Incorporate regional security regulations and comply with regional peculiarities (e.g. ShangMi ciphers for the Chinese market)
- Create transition mechanisms to enable safe and secure migrations to new security measures

The research field of crypto-agility is boosted by the ongoing effort to define and integrate new, quantum-resistant cryptographic algorithms into systems to protect them against the threat of quantum computers, known as *Post-Quantum Cryptography* (PQC) [14]. As elaborated, for example, by Ott & Peikert [15] or Paul [16], the migration to PQC demonstrates the necessity of crypto-agility, especially for OT systems. However, it is also a perfect opportunity to create and implement appropriate capabilities to achieve long-term security. Furthermore, proper transition mechanisms are a key aspect for OT systems, as the long mission times and the scarce update capabilities of equipment complicate the deployment of new security functionality on a system-wide level. Hence, crypto-agility must also consider retrofit options for already deployed systems.

Another important aspect of crypto-agility is further elaborated on by Sikeridis et al. [17]. They state that crypto-agility should not only be considered as a capability of a single instance within a system (e.g., a software library, a used protocol, or a single device), but also as an attribute of a complete enterprise system. Hence, all layers of the PERA model (see Figure 1) have to be considered. Their „enterprise-level“ view of crypto-agility considers the overall system and infrastructure, with all devices and the integrated security measures. Furthermore, they added key aspects like central control and maintenance, and overall documentation of deployed measures to their view. Paul and Niethammer also identified that an expanded view of crypto-agility is required in automation systems to provide a real benefit for long-term security [18].

For the remainder of our work, we define two dimensions of crypto-agility, between which we differentiate in the following analysis parts and in the presentation of our concept. On the one hand, there is *Implementation-Agility*. This dimension considers ways to easily update, extend or replace the implementation of security measures without adapting or changing the higher-level application using it. Hence, this dimension covers the list of supported security measures of equipment within a system, as well as the retrofit of security measures into devices and systems without prior support of those. On the other hand, we define *Configuration-Agility*, which encompasses all aspects regarding the configuration of the currently used set of security measures within a device or system from the list of supported ones. This covers the

maintenance and coordination of events like migrating a system to a new algorithm once all nodes support it.

In the next two parts, we take a more detailed look at the current state of crypto-agility capabilities within the presented security measures for OT communication systems and identify problems from both a software and hardware perspective.

D. Software-related Problems

Analyzing crypto-agility capabilities of secure OT communication from a software perspective, we identified problems in both of our defined dimensions. The main problem regarding implementation-agility is the tight coupling of security features to specific applications. To integrate security measures into an application, developers have to use common software libraries like OpenSSL, Botan, or WolfSSL directly from within the application itself. Those libraries commonly offer an extensive *Application Programming Interface* (API) for specific features (e.g., to establish TLS connections or to verify signatures). This setup results in various problems regarding crypto-agility that are already elaborated on in the literature, e.g., by Green and Smith [19] or by Georgiev et al. [20]. Available APIs are commonly very complex, especially for software developers without profound knowledge of security. In addition, proper and thorough documentation of the APIs is oftentimes lacking. Hence, using the libraries is very error-prone, leading to potential security vulnerabilities.

Furthermore, the resulting tight coupling between application and security functionality complicates an update of the security library (e.g. with support for a new algorithm) or a configuration adjustment (e.g. key length of an algorithm or logic to verify a peer). When a library update changes the API, updating the application is necessary, too. This increases the effort associated with an update for both the manufacturer of the component and the operator of the system, resulting in a slower deployment of fixes and new features. Ultimately, this leads to many systems being vulnerable to existing attacks and using older, potentially insecure implementations or measures. To improve implementation-agility, the application and the security implementation needs to be decoupled to enable independent changes to the security features without modifications to the application.

Regarding configuration-agility, the biggest problem is the static integration of a fixed set of security measures into the application without a way for external configuration of important parameters (e.g., algorithm selection, key length). The application manufacturer is solely responsible for the selection of supported functionality and its configuration in their individual product. This means that operators have no means of adapting the security measures running on their equipment to their needs without the manufacturer's support. Furthermore, the user-facing interfaces to configure security measures typically offer only limited options, and are typically integrated into the general maintenance interface of the application. Hence, each individual device must be configured individually in case of a security configuration change, drastically increasing effort and also coupling security maintenance to application maintenance. In typical systems with equipment from multiple manufacturers and of different releases (i.e. older and newer devices), this ends up in a „best-possible“ security configuration using the common set of supported measures. The result is unlikely to follow the current state-of-the-art, considering the slow deployment of new equipment within OT systems and the dependency on

all manufacturers of used equipment to provide updates with new features for their products.

To improve configuration-agility, a centralized and easy way for operators to configure and manage the used security measures is necessary, independent of the application and with a manufacture-independent interface. This allows for a uniform, system-wide selection and an easier-to-coordinate migration process.

E. Hardware-related Problems

The hardware capabilities of deployed equipment also influences the achievable level of crypto-agility. This is especially important in OT systems, as the devices are oftentimes based on resource-constrained embedded hardware rather than powerful enterprise hardware. Furthermore, dependability and functional safety regulations, and the possibly rough environmental conditions of these systems further limit the options regarding hardware designs. We identified two hardware-related problems, both within the dimension of implementation-agility.

Firstly, the constrained processing resources of the devices limit the possibilities of software updates with newer cryptographic algorithms. This is especially relevant in the context of PQC, as the new algorithms have significantly greater requirements regarding CPU processing power and memory usage. For example, on a medium-sized microcontroller with 192 kB of available RAM, a TLS handshake using PQC algorithms alone occupies around 35 % of RAM, compared to around 1 % using classical elliptic-curve-based algorithms [21]. Even if a software update is possible to integrate these algorithms into deployed devices, the remaining available resources could be insufficient to make the new setup work properly. Hence, available hardware resources can impose restrictions on software-based implementation-agility.

Secondly, next to the hardware implications on software, there also arise problems due to hardware-assisted cryptography. In more recent microcontrollers and microprocessors, hardware peripherals are available to accelerate the calculations of cryptographic algorithms. This hardware acceleration greatly improves performance and sometimes enables the integration of security into these kinds of resource-constrained devices in the first place, as timing requirements could be achieved that are otherwise infeasible. Those peripherals, however, only support a defined static set of algorithms without a way to update them once a newer algorithm should be deployed. Furthermore, in case a vulnerability within the hardware implementation is identified, deployed chips cannot be upgraded, and fixing the flaw in the chip design is also very costly for the manufacturer. Hence, once the algorithms supported by an accelerator within a device are considered insecure or outdated, the device loses the hardware acceleration and has to fall back to software implementations, which suffer from the earlier described resource limitations.

Therefore, hardware-assisted cryptography needs to be upgradable after deployment. This enables integrating hardware acceleration of new cryptographic algorithms or fixing identified vulnerabilities in the design.

Next to the processor-internal hardware acceleration, there also exist external hardware modules that can be integrated into a device to add security measures with hardware assistance. These modules are commonly referred to as *Secure Elements* or *Trusted Platform Modules* (TPM) and are commonly available as either solder-down chips, small pluggable PCBs, or exchangeable plastic chip cards. Generally, those modules are used to store long-term keys related to the

cryptographic identity of the connected host device. The viability of this approach for OT systems has already been verified in different experimental investigations [22]–[24]. When soldered on the devices’ PCB, those modules, however, suffer from the same problem as the internal peripherals because their fixed functionality cannot be upgraded once deployed. The exchangeable modules, on the other hand, significantly improve crypto-agility, as both hardware capabilities (implementation-agility) and also stored cryptographic keys with different parameters (configuration-agility) could be upgraded after deployment. Depending on the flexibility of the API that is used on the host application to access the module’s functionality, crypto-agility could be further improved, as it is possible to swap the pluggable module without the need for updating the host application.

Such architectures, however, possess their own specific attack surface. As the communication interface between such a module and the host using it is typically not cryptographically secured, a malicious actor could tamper with the transmitted data and bypass the security features. Furthermore, as there is no cryptographic coupling between the module and the host, theft of a module results in a malicious actor taking over the valid cryptographic identity stored on the module. This opens the door for impersonation attacks. There are measures available for some types of external modules to protect the setups from the threats described above (e.g. [25]). However, to the best of our knowledge, there is currently no adaption of such protective measures that is suitable for OT systems, as these are typically based on human interaction with the setup when the module is used. In OT systems with no on-site human presence besides infrequent maintenance work, those measures are, therefore, not feasible. With those shortcomings addressed in the future, however, such modular setups would greatly benefit crypto-agility, as they would enable easy upgrades of hardware-based cryptography.

Based on the identified problems of current security measures for OT communication regarding both software- and hardware-based crypto-agility, the next section presents related work that already addresses those shortcomings.

III. RELATED WORK

Before we introduce our concept to improve crypto-agility within OT systems, related work regarding the identified problems is presented. We split the presentation into solutions for the development of new equipment (Part III-A) and solutions for retrofitting existing systems (Part III-B).

A. Proposed Solutions for New Developments

To address the tight coupling between an application and security features, there are publications to provide simpler and more generic interfaces between the two to achieve better separation. For example, O’Neill et al. have two publications to outsource security functionality from applications to the operating system. In [26], they introduce the *Secure Sockets API* that enables applications to establish TLS connections using the common POSIX socket API. All configuration of the TLS connections is done on the device-level using a configuration file independent of the individual application. In [27], they propose the same idea for authentication functionality as an operating system service. Hence, applications pass certificates to the OS via a generic API to verify them. Both presented separation approaches simplify updating the underlying security functionality independent of the application (implementation-agility) and enable central management

of the actual used cryptographic algorithms (configuration-agility).

The already mentioned architecture by Sikeridis et al. [17] also features a novel *Crypto Provider* software component that provides a flexible and abstract API for the application to isolate security from it, thereby facilitating independent updates (implementation-agility). Another central aspect of their presented software module is an additional control-plane interface for service and management that enables an operator to centrally configure all Crypto Providers within a system at once (configuration-agility).

Regarding improvements to implementation-agility from a hardware perspective, there is literature on integrating *Field Programmable Gate Arrays* (FPGAs) into devices and synthesize security functionality on them (e.g. for VPN functionality [28]). This would enable both fast hardware acceleration of algorithms and future update capabilities, as the FPGA could be reprogrammed in the field through a software update of the device. Furthermore, work within the enterprise context currently moves security functionality away from a server onto an attached programmable network interface card (NIC), so-called *SmartNICs* (e.g., [29], [30]). This modularization improves crypto-agility, as the SmartNICs are independent hardware modules connected to the server via generic hardware and software interfaces, enabling partial hardware upgrades in case of a new security measure that requires an updated SmartNIC, without affecting the main server.

B. Retrofitting Existing Deployments

All presented approaches to address the problems of current systems require substantial modifications of the applications or even of the whole device to integrate the proposed Crypto-agile security measures. In many currently deployed systems, however, it is not possible to update the applications or devices in such a fundamental way. Hence, retrofit approaches are necessary to provide a fast migration path for existing systems. This means that crypto-agility capabilities have to be integrated transparent to the application.

In the context of secure communication, a very promising approach for this integration is a design based on *proxies* that are integrated into the existing communication paths. They then intercept the traffic from the existing devices and apply the desired security measures. Such proxy designs have already been demonstrated in the OT context within the literature [31]–[34]. The traffic is typically intercepted either on the underlying transport layer (OSI layer 4) or directly on the application layer (OSI layer 7). The proxy is either deployed within the existing device as a software service (if that is possible, for example within an Embedded Linux system) or within an additional *Bump-in-the-Wire* (BitW) device that is integrated into the physical communication path. This integration of security measures, which is completely independent of the existing application or device, enables a retrofit of existing systems, but also generally achieves independence of the manufacturers of actual OT equipment. This final advantage of the proxy approach is a key aspect of our concept, as it results in a faster deployment of crypto-agility compared to the dependency on various manufacturers to integrate said features into their devices.

Each presented related work addresses some identified problems and limitations of security measures for OT communication regarding their crypto-agility capabilities and, thus, creating a solid foundation for our concept to build upon.

IV. PROPOSED ARCHITECTURE

In this section, we present our architecture concept to improve crypto-agility within OT systems. Our concept is similar to the work of Sikeridis et al. [17], however, adapted to meet the requirements of OT systems not considered in their enterprise-focused work. Our design-goal is to create a solution that improves crypto-agility both for new devices and also existing ones via a retrofit approach. The retrofit aspect is a key feature, as already deployed OT systems and devices need agile security measures to create a fast migration path to long-term security. For this process, however, new developments have to be considered simultaneously to create a reusable solution that can be widely adopted. We have designed various software modules to provide security measures for OT devices, which contain capabilities for both implementation-agility and configuration-agility to improve the achieved level of crypto-agility. In the following, we first present the detailed integration into actual devices, separated into deployment in new applications (Part IV-A) and retrofit for existing ones (Part IV-B). Afterwards, Part IV-C contains a discussion of the concept and its various approaches.

A. Deployment in new Applications

We created the *Agile Security Library* (ASL) to provide common features for secure communication behind a small, generic API for applications to use, similar to the Crypto Provider in [17]. The deployment approach is depicted in Figure 2. Newly developed applications simply use the generic interface to communicate with external peers. All functionality of the library is based on well-established standards and protocols to guarantee interoperability with other implementations. The small, generic API is the key feature to improve implementation-agility, as more functionality can be changed and upgraded within the library without modifying the interface for the application. Hence, it is easier to update the library independent of the application.

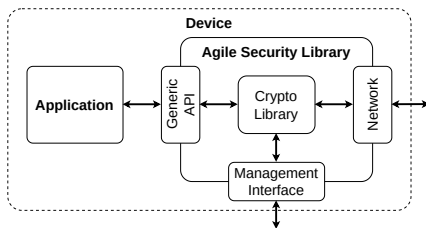


Figure 2. Deployment of agile security measures as a software library that can be used by applications. This approach requires the application to use the new small, generic API.

Along the interfaces for actual data processing, the library contains an additional management interface for the configuration of all security related functionalities. On the one hand, this moves security configuration away from the application onto the device-level, further improving their mutual independence. On the other hand, a distinct management interface simplifies the configuration for an operator as well as the implementation of a centralized control and management platform (i.e. fleet management). Those measures improve configuration-agility for devices incorporating the ASL.

The actual configuration of endpoints for applications is based on *profiles*. A profile contains all relevant parameters for an endpoint related to secure communication. For example, an endpoint profile describes the configuration of a TLS server with its supported cipher suites and its certificate chain. Via the management interface, profiles can be created

and modified. The profiles and their associated data (e.g. certificates or private keys) are stored in an isolated and secured location within the device. On application startup, a specific profile is loaded with an API call to get an endpoint context. If no profile is specified, a default profile is used. This behavior makes sure that safe defaults are implemented. Using the obtained context, secure sessions can be created within the application for connections with peers.

Due to the generic API of the library, security functionality can be provided for communication systems on different OSI layers. Depending on the profile configuration, the desired security features are applied transparently to the application data without considering other OSI layers. The library further supports both synchronous and asynchronous program flows to provide flexibility for application developers.

Finally, the library is optimized for deployment in OT devices by focusing on aspects like minimal resource footprints and minimal dependencies on other libraries. Furthermore, it is designed to be used within various operating systems to offer flexibility and increase deployment possibilities.

B. Retrofit Deployment

To retrofit the crypto-agility capabilities of the ASL into already deployed OT systems without modifications to applications, we created an architecture design using the aforementioned transparent proxies. The proxies are placed within the communication path to intercept the application traffic and integrate security measures transparently for the existing peers. Within a proxy, the ASL is used to incorporate the improvements regarding implementation-agility and configuration-agility. Due to the transparent deployment of the proxy next to the application without any direct coupling, both dimensions of crypto-agility are even further improved compared to the direct integration of the ASL into applications. We elaborated on two integration approaches for the proxy, as described below.

The first approach is to deploy the proxy within the existing device as an additional software service. Depending on the particular device, this integration may require assistance from the manufacturer (for the installation of the supplementary application) or may be accomplished by the operator themselves, provided that the device is not locked down (i.e. the operator can install the supplementary services independently). As the proxy is a standalone service running besides the application, independent updates and configuration are straightforward.

The internal design of the proxy within the device is shown in Figure 3. The proxy application uses the ASL to incorporate both implementation-agility and configuration-agility. To manage the proxy itself, an additional management interface is present.

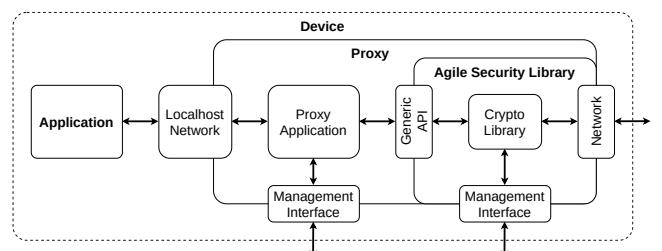


Figure 3. Deployment of the agile security measures within a proxy application deployed on the device itself. In this setup, the application does not need any modifications, but the proxy must be deployed on the device and be transparently integrated into the network path.

To implement the transparent integration into the communication path, the proxy intercepts the network traffic on the desired OSI layer from within the networking infrastructure of the operating system. This deployment results in the messages already wrapped within a suitable transport protocol (e. g., TCP, UDP, or a Layer 2 Ethernet frame). There are then two strategies for the proxy to handle the messages received from the application.

The first strategy is to extract the upper layer payload from these messages, secure them with the configured cryptographic measures, and finally pack them in a new message of a transport protocol. This results in the proxy terminating the connection with the application, requiring it to simultaneously establish a second connection with the remote peer. This setup breaks the end-to-end connection between the two applications and, therefore, influences protocol functionality like flow control. However, this strategy is otherwise fully transparent for both existing peers, providing maximum compatibility.

The alternative strategy is to wrap the complete received message including the metadata of the transport protocol within a secure message and send it to the peer. This mode of operation does not influence the end-to-end flow control of the existing connection between the peers, as each message is *tunneled* over the secure connection. Such tunnel behavior is default in regular VPN systems. However, both peers have to support the tunnel functionality (either by deploying a proxy or by processing the tunnel messages themselves), limiting interoperability and increasing the deployment effort.

Adding proxies to a system to intercept network traffic and add functionality this way is well-established within the cloud infrastructure domain. There exist various proxy applications for cloud servers to act as load-balancers or as security endpoints (e. g. NGINX or Envoy). However, those applications are optimized for cloud environments with their specific protocols and are built upon various technologies common for these systems (e. g. containerized applications and powerful enterprise hardware). Hence, using these available proxy applications in OT systems is not viable, especially considering the retrofit of existing devices.

Consequently, we created our own proxy application specifically for OT systems. This application can work both as a forward proxy (actively establishing connections to a peer) and as a reverse proxy (waiting for incoming connections). Furthermore, it has minimal software dependencies and its memory footprint is optimized to be as small as possible to maximize portability onto existing devices while simultaneously delivering the highest possible performance to limit interference with the communication flow.

Some available, more powerful commercial devices for OT systems nowadays also already feature application designs based on containers (e. g. using Docker) to increase the flexibility of application development and deployment. For such systems, the proxy could also be deployed as a container to be integrated into the containerized application complex.

For cases where an existing device is locked down or has not enough resources available to deploy the proxy as a service directly, we created an alternative approach based on a *Bump-in-the-Wire* device. In this scenario, depicted in Figure 4, the BitW device is integrated into a communication path as an interceptor. On this additional device, our proxy application is deployed to provide the same overall functionality for the system as the approach above.

This deployment approach further increases the achieved

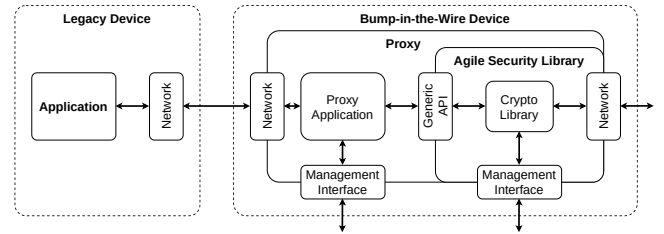


Figure 4. Deployment of the agile security measures within a standalone BitW device within the network path. In this setup, the legacy does not need any modifications. Only the additional device must be transparently integrated into the network path.

level of crypto-agility, as both hardware and software are independent of the existing device. Furthermore, basically any system can be retrofitted by deploying BitW devices. In one of our previous work, a BitW security gateway has already been presented for a very specific use case within energy supply systems [34], which served as the foundation for the more general and flexible BitW proxy approach in this work.

Besides using a standalone external device to deploy the proxy application, other already existing devices within the OT systems could also be considered to host the proxy. For example, modern network equipment like switches or firewalls feature powerful processors and are typically based on common operating systems. Hence, it would also be possible to deploy the proxy on those devices. This setup would result in equal crypto-agility capabilities, while avoiding the additional BitW devices within the system.

C. Discussion

The presented concept to integrate crypto-agility capabilities for both new and existing devices with its different deployment approaches covers all use cases within OT systems. However, each approach has its individual advantages and disadvantages. For that reason, we present a discussion of the three approaches in the following.

1) *Achieved level of crypto-agility*: Comparing the three deployment approaches regarding the achieved level of implementation-agility, the BitW proxy deployment wins. It offers the most extensive separation of security measures from an application and enables both independent hardware and software updates without any modification of the application. Deploying the proxy within an existing device, we achieve a high degree of software independence, but the proxy is still bound to the available hardware resources on the device. Finally, the direct integration of the ASL within a newly created application offers the least amount of implementation-agility, as we still have a direct coupling between application and security functionality via the small, generic API. Considering configuration-agility, all three approaches feature the same management interface, resulting in the same capabilities.

2) *Impact on Communication behavior*: When considering the impact of the deployment approaches on the communication behavior of a single device and of the overall system, the sole usage of the ASL is to be preferred. Due to the direct integration into an application, the overhead for each message is kept minimal, resulting in the smallest latency increase and the smallest bandwidth reduction. Comparing the two proxy deployments, the resulting system influence depends on the hardware and software environment. In case the existing device has enough resources available to deploy the proxy directly, the resulting influence can be small (however, still

larger than the direct library integration due to networking overhead). When resources are limited, the influence can grow larger, negatively impacting the communication behavior. The same applies to the BitW deployment: when powerful hardware is used, the influence can be quite low. With less resources available, the negative impact on system performance increases.

3) *Costs and Deployment Effort*: In this category, we have to clearly differentiate between new developments and retrofit deployments. For newly developed devices, the integration of the ASL directly into the application is the most viable approach, as it limits costs and effort for the manufacturer and also requires no additional integration steps or maintenance overhead for the operator after installing a new device with these capabilities. Considering the retrofit of existing devices, the two proxy approaches score differently, depending on the perspective of the manufacturer or the operator. For the manufacturer, the BitW deployment is more attractive, as it involves no work regarding existing devices. Furthermore, new devices can be sold to customers, even in case they use existing devices from other manufacturers. For operators, on the other hand, the direct integration into existing devices is better suited, as additional hardware costs are avoided. The deployment effort for both proxy approaches from a maintenance-perspective of the operator is comparable.

4) *Safety and Reliability*: Finally, the influence of the three deployment approaches regarding the overall safety and reliability of the system is considered. Assuming that all newly created devices incorporate measures for secure communication and, hence, a software library to implement those, anyway, the direct integration of the ASL performs best of all three approaches. The small, generic API for the application decreases the effort for developers to safely and securely integrate the security measures into their applications and devices. For an operator, a common management interface for various devices also decreases the effort to properly configure the security measures, resulting in less errors.

Both proxy deployment approaches for retrofit add additional software and possibly also additional hardware (BitW approach) directly into the main communication paths within the OT systems. As a result, both the hardware and software of the proxy have to be developed with a high level of assurance regarding safety and reliability to not negatively influence the overall system (e. g. a significant increase of the Mean-Time-to-Failure). Due to the limited scope and complexity of the proxy application and the possibly dedicated hardware development for exactly the desired use case, those requirements could be met with moderate effort, however.

In summary, the presented concept with its three deployment approaches enables the integration of agile security capabilities into both existing and new devices. With the key focus on the retrofit aspect of existing systems, a fast migration path is created until all deployed devices are capable of crypto-agility themselves. Using the dedicated management interface, an operator can configure the system based on the individual threat model and security goals, independent of the actual OT application.

V. CASE STUDY: SECURING IEC 61850 MMS COMMUNICATION

To demonstrate the viability of our concept, we created reference implementations for both the ASL and the proxy application. Using these, we conducted a case study securing a test environment based on the IEC 61850 MMS protocol to evaluate the performance and system influence of our designs.

In the following, we first present more details about the reference implementations in Part V-A. As indicated in the discussion above (see Part IV-C), the direct integration of the ASL into new applications should behave very similarly to an integration of a „classical“ security library, as only the API differs. Hence, we use the performance of this integration as a reference for the comparison of our proxy approaches within the case study. At first, we focus on the proxy application and measure its resource consumption (Part V-B) as well as the achievable bandwidth (Part V-C). Thereafter, we present various measurements to validate the timing influence of the proxy integration on system behavior compared to the ASL in Part V-D.

A. Reference Implementations

To maximize compatibility and portability of our approach, and also to demonstrate the viability in a broad device spectrum, we created our reference implementations for both Embedded Linux and the bare-metal Zephyr RTOS. Thus, we can demonstrate our approaches for both microcontroller-based and also more powerful microprocessor-based systems. For the underlying cryptographic library, we use WolfSSL, as this library supports a broad spectrum of different architectures and scales well from small microcontrollers up to powerful processors. For our agile security library, we created a wrapper around WolfSSL with our small, generic API and the additional management interface. Currently, the profiles with the endpoint configuration are implemented as JSON files which are loaded during initialization. A thorough user-facing interface with remote access is not yet implemented, however.

Our proxy application is implemented in C and runs on both Linux and Zephyr. It is currently limited to TCP/TLS functionality, acting as both a TLS forward and a reverse proxy. Thus, TCP connections can be intercepted and *upgraded* to TLS. The forward proxy waits for incoming TCP connections and upgrades them to TLS. The reverse proxy, on the other hand, waits for incoming TLS connections and forwards all traffic to a TCP connection. We only allow TLS in version 1.3 and only support mutual authentication, so both server and client have to authenticate using a valid certificate chain. Once all connections are established, data transmission is possible in both directions. The proxy can handle multiple parallel connections to multiple different hosts. The configuration is also done using a JSON file, with a proper management interface on the roadmap for future extension. Furthermore, other protocols are planned to be integrated in the future, e. g. UDP/DTLS or support for OPC UA Secure Channels.

For our case study, we created a setup using the IEC 61850 MMS protocol, as it is based on TCP and its security recommendations are built upon TLS (see Part II-B). The complete setup is depicted in Figure 5. We created both an MMS server and client using the open-source library libiec61850. The server contains a data model with various variables. After connection establishment (TCP handshake and MMS handshake), the client periodically reads single data objects to track value changes. The IEC 61850 library also offers optional support for TLS encryption directly integrated into the code, based on the library mbedTLS. This existing interface has been the basis for our integration of the ASL directly into the application.

To represent a realistic and reproducible test system in the laboratory, we deployed both the MMS server and client on two Raspberry Pi 4 with the Raspberry Pi OS lite, as this

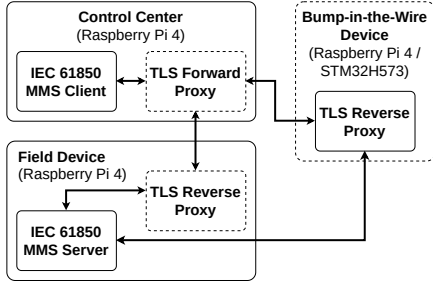


Figure 5. Test setup for our case study based on the IEC 61850 MMS protocol. Dashed lines indicate that the device or software module is only present in a subset of the performed tests.

hardware should roughly be on an equal performance level as typical commercial devices for OT systems and is widely available for validation of our measurements. The network between the devices is based on a single Gigabit Ethernet switch to minimize the influence of network latency.

For deployment of our proxy application on an external device, we limit the case study to a BitW proxy only for the server side in front of the Raspberry Pi to minimize the number of test cases. The BitW device in the test setup is based on either another Raspberry Pi 4 running Raspberry Pi OS lite or a microcontroller system based on the STM32H573i evaluation board from STMicroelectronics (Cortex-M33 with 250 MHz) running Zephyr RTOS.

B. Resource Consumption

The resource consumption of the proxy application is measured to estimate the overhead that is loaded onto a device when the proxy is deployed as an additional service besides the application. As this deployment approach is much more likely for Embedded Linux systems due to their better extensibility, we focus our measurements on this OS. To obtain the size of the executable, we compiled the proxy with statically linked dependencies. The resulting executable is roughly 2.3 MB in size. The proxy application for the Zephyr RTOS could not be measured in isolation, as it is embedded into a complete firmware image for the microcontroller. But the current firmware, containing the proxy and all supplementary code, fits onto a microcontroller with 1 MB of flash space.

To obtain the RAM footprint of the proxy during execution, the tool *Valgrind* with its heap profiler *massif* has been used. As stack space of Linux processes is also dynamically allocated, this measurement can obtain both Heap and Stack usage of a process. The peak memory usage of the proxy occurs during the TLS handshake, when the peer certificate chain is parsed. When only a single TLS handshake is performed using certificates based on the ECC `secp521r1` algorithm, the peak memory usage reaches 117.6 kB. In a synthetic scenario with 10 simultaneous handshakes, the setup peaks at 389.6 kB. This footprint is considered to be small enough to make the proxy deployable on a typical Embedded Linux device. For microcontroller-based systems, the footprint is approximately the same, as the same code is used. On those systems, however, the number of parallel connections must probably be constrained to sustain viable RAM footprints.

Regarding CPU resources, the proxy application follows the *best-effort* principle. Hence, in case of limited free processing resources, the data processing performance of the proxy simply decreases. This effect, however, has to be validated on a per-device basis.

C. Bandwidth

To measure the achievable bandwidth, we used the well-known tool *iPerf*. Each of the two Raspberry Pi 4 run one instance of the tool, either as a client or as a server. We tested both the setup with the proxy deployed on the Pi itself and the one with the proxy on another Pi acting as a BitW device. Two test cases have been defined, one with a single connection for the bandwidth measurement and one with four parallel connections to use all CPU cores of the Raspberry Pis.

For both deployment approaches, we achieved nearly identical measurements. In the test case with only a single connection and transmission in one direction, both approaches achieve a bandwidth of around 180 MBit/s. When looking at the CPU resource consumption, only a single CPU core of the Pis is used, running at 100%. The test case with four parallel connections utilizes all four CPU cores to 100% each, as the proxy application creates a new thread for each connection, achieving a bandwidth of around 620 MBit/s (in both deployment approaches). Based on these results for both deployment approaches, we conclude that the AES implementation within the TLS library is the limiting factor regarding bandwidth. In the case of a test setup with full-duplex data transmission over the same connections, the achievable bandwidth per thread of around 180 MBit/s is split onto the two directions, cutting the total bandwidth roughly in half for each direction.

We also measured the setup with the STM32 microcontroller as BitW device. However, we identified the network stack on Zephyr to be a limiting factor regarding network bandwidth, achieving poor performance of only around 13 MBit/s in total using *iPerf*. Hence, those numbers are only a limited indication of the proxy performance running on the microcontroller.

In summary, the achieved bandwidth values show that the proxy approach is generally viable and reaches acceptable data rates. When deployed on a hardware platform with hardware acceleration for AES, the bandwidth should also reach the Ethernet link limit of 1 GBit/s.

D. Timing Influence

Finally, we measured the influence of the proxy integration on key timing parameters within the test setup compared to the direct integration of the ASL. The test setup from above leads to multiple test cases in which we obtained the timing parameters: insecure TCP connection between client and server, direct integration of the ASL into the application, proxy deployment directly on both devices as a software service and server-side BitW proxy deployment (Raspberry Pi and STM32 microcontroller). All TLS implementations use ECC `secp384r1` certificate chains with one intermediate certificate, and only mutual authentication is enabled. For each timing parameter and test case, we took 10000 measurements and calculated the mean value and the 99th percentile.

The first measurement is the time to establish an IEC 61850 MMS connection. This includes TCP connection establishment, TLS handshake (if present in the test case) and MMS handshake. The results are depicted in Figure 6. The microcontroller-based values are omitted, as the TLS handshake takes around 300 ms on average, which would impede the visualization in the violin plot.

As can be seen, the integration of TLS into the handshake massively increases the connection establishment time in general, compared to the plain TCP setup. Considering the

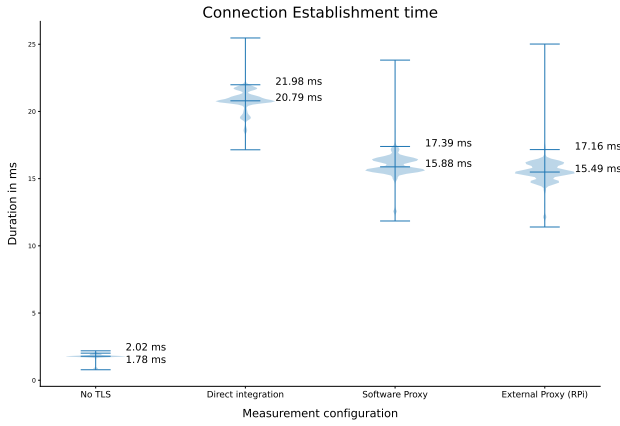


Figure 6. Violin plots of the connection establishment time for the different test cases (excluding the microcontroller-based BitW proxy). The written values indicate the mean value and 99th percentiles for each measurement.

amount of data that is exchanged, and the cryptographic calculations performed during a TLS handshake with mutual authentication, however, such an increase is expected. Surprisingly, the measurements show that the two proxy deployments are even a bit faster than the direct integration of the ASL into the MMS application. This worse result of the direct integration is probably due to unoptimized data flows and worse event handling within the MMS application compared to the optimized proxy application, as the cryptographic code and the TLS configuration are identical in all deployments. The two proxy deployments show no noticeable difference, indicating that both approaches are viable. In total, the achieved handshake times with both proxies demonstrate that the proxy approach in general is a viable alternative to a direct integration. The much larger values from the microcontroller-based setup generally are not ideal. However, within OT systems, the connection is typically kept open for a long period, decreasing the negative impact of the longer connection establishment time. Hence, we also consider the proxy deployment on a microcontroller a viable approach regarding the connection establishment time.

The second measurement captures the time to read a single data object from the server. In this test, the connections are already fully established. Hence, we only measure the time overhead of the data processing and the resulting latency caused by the proxy. Figure 7 shows the results as violin plots.

The direct integration of the ASL only marginally increases the measured time to read a single data object compared to the plain TCP setup. This results from the minor overhead of the TLS record protocol, mainly the AES encryption and decryption, running on a powerful processor of the Raspberry Pi. The two Raspberry Pi-based proxy deployments add only a minor latency overhead of 0.15 ms and 0.2 ms, showing that a proxy on a powerful hardware platform only marginally influences timing parameters. The minor latency increase in the case of an external deployment compared to a software deployment is probably caused by the additional physical network transmission over Ethernet compared to the in-memory-transmission over localhost within the software deployment.

The hardware of the STM32 microcontroller features a hardware accelerator for the AES algorithm. Hence, we created two versions of the Zephyr-based BitW proxy firmware: one with and one without utilizing this accelerator. As

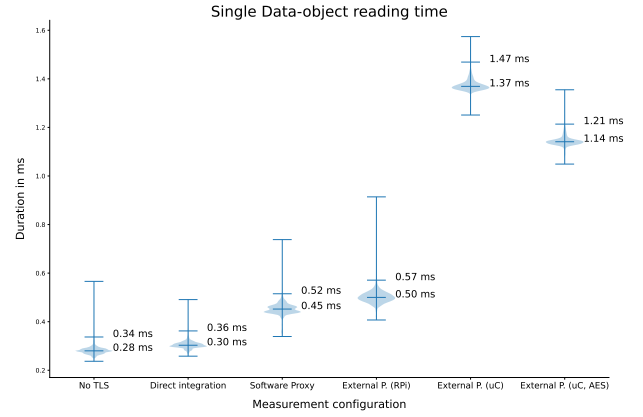


Figure 7. Violin plots of the time to read a single data object. The written values indicate the mean value and 99th percentiles for each measurement. The right-most measurement of the microcontroller-based deployment utilized the AES hardware acceleration of the microcontroller.

can be seen in the two right-most plots of Figure 7, both measurements add a noticeable amount of additional latency compared to the Raspberry Pi-based proxies. However, the measured values are still small compared to typical values of real-world network latency in Ethernet networks without hard real-time requirements, hence not causing a large additional delay. The hardware acceleration of the AES algorithm decreases the latency by around 0.2 ms. However, the value is still more than double the value of the Raspberry Pi based BitW proxy. This indicates that the software processing overhead of the TLS protocol is bigger than the required time to actually encrypt and decrypt the messages. Considering the much lower price and the much simpler hardware design of a microcontroller-based system compared to a Raspberry Pi, the deployment of the proxy on a microcontroller is definitely a viable approach in case the timing requirements of the system allow it.

To conclude, the results of the presented case study show the general viability of our proxy approach to retrofit agile security measures into existing systems, both within an existing device as a software service as well as within a BitW device with various hardware capabilities.

VI. CONCLUSION AND OUTLOOK

In this paper, we presented an analysis of the current state of crypto-agility in OT systems regarding secure communication and identified problems and possible improvements. Furthermore, we proposed a new architecture concept to integrate security measures into both current and future systems while improving crypto-agility. Finally, we demonstrated the viability of our concept with its deployment approaches and conducted a thorough analysis of the influence of the new setup onto system behavior.

In the future, we plan to further improve our reference implementations with support for more protocols (e. g. UDP and DTLS, OPC UA, IPsec, MACsec) and use cases (e. g. multicast communication). Furthermore, we want to examine the influence of the concept on systems with real-time requirements. Finally, we want to elaborate on further enhancements to our agile security library, both in terms of the coupling to the application (e. g. integration into the secure sockets API) and more thorough internal implementation agility (e. g. internal modularization).

ACKNOWLEDGEMENT

The presented work is part of the research project *KRITIS Scalable Safe and Secure Modules (KRITIS³M)*, which is funded by the Project Management Jülich (PtJ) and the German Federal Ministry for Economic Affairs and Climate Action (BMWK) under funding code 03EI6089A.

REFERENCES

- [1] Bundesamt für Sicherheit in der Informationstechnik. (2021, May) Zweites Gesetz zur Erhöhung der Sicherheit informationstechnischer Systeme (IT-Sicherheitsgesetz 2.0). Publication.
- [2] C. Singh, "European cyber security law in 2023: A review of the advances in the Network and Information Security 2 Directive 2022/2555," *Cyber Security: A Peer-Reviewed Journal*, vol. 7, no. 1, pp. 82–92, September 2023. [Online]. Available: <https://ideas.repec.org/a/aza/cs/j000/y2023v7i1p82-92.html>
- [3] Bundesamt für Sicherheit in der Informationstechnik. (2024, February) Technische Richtlinie TR-02102 - Kryptographische Verfahren: Empfehlungen und Schlüssellängen. Publication.
- [4] S. D. D. Anton, D. Fraunholz, D. Krohmer, D. Reti, D. Schneider, and H. D. Schotten, "The Global State of Security in Industrial Control Systems: An Empirical Analysis of Vulnerabilities around the World," *IEEE Internet of Things Journal*, pp. 1–16, 2021.
- [5] T. Williams, "The Purdue Enterprise Reference Architecture," *IFAC Proceedings Volumes*, vol. 26, no. 2, Part 4, pp. 559–564, 1993, 12th Triennial World Congress of the International Federation of Automatic control. Volume 4 Applications II, Sydney, Australia, 18–23 July. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1474667017485326>
- [6] G. M. Makrakis, C. Koliass, G. Kambourakis, C. Rieger, and J. Benjamin, "Industrial and Critical Infrastructure Security: Technical Analysis of Real-Life Security Incidents," *IEEE Access*, vol. 9, p. 165295–165325, 2021. [Online]. Available: <http://dx.doi.org/10.1109/ACCESS.2021.3133348>
- [7] *IEC 62351: Power systems management and associated information exchange – Data and communications security*, International Electrotechnical Commission Std.
- [8] Modbus Organization, "MODBUS/TCP Security Protocol Specification," 2018. [Online]. Available: https://modbus.org/docs/MB-TCP-Security-v21_2018-07-24.pdf
- [9] OASIS MQTT Technical Committee, "MQTT Version 5.0 Specification," 2019. [Online]. Available: <https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.pdf>
- [10] OPC Foundation, "OPC 10000-2 UA Part 2: Security," 2023. [Online]. Available: <https://opcfoundation.org/developer-tools/documents/view/159>
- [11] T. Müller and H. D. Doran, "PROFINET Real-Time Protection Layer: Performance Analysis of Cryptographic and Protocol Processing Overhead," in *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1, 2018, pp. 258–265.
- [12] N. Alnahawi, N. Schmitt, A. Wiesmaier, A. Heinemann, and T. Graßmeyer, "On the State of Crypto Agility," in *18. Deutscher IT-Sicherheitskongress*. SecuMedia Verlags-GmbH, February 2022, pp. 103 – 126.
- [13] H. A. Mehrez and O. E. Omri, "The Crypto-Agility Properties," in *Proceedings of the 12th International Multi-Conference on Society, Cybernetics and Informatics (IMSCI 2018)*, 2018. [Online]. Available: <https://www.iiis.org/cds2018summer/papers/ha536vg.pdf>
- [14] National Institute for Standards and Technology. Post-Quantum Cryptography. NIST. Accessed: March 27th, 2024. [Online]. Available: <https://csrc.nist.gov/projects/post-quantum-cryptography>
- [15] D. Ott and C. Peikert, "Identifying Research Challenges in Post Quantum Cryptography Migration and Cryptographic Agility," 9 2019. [Online]. Available: <http://arxiv.org/abs/1909.07353>
- [16] S. Paul, "On the Transition to Post-Quantum Cryptography in the Industrial Internet of Things," Ph.D. dissertation, Technische Universität Darmstadt, Darmstadt, 2022. [Online]. Available: <http://tuprints.ulb.tu-darmstadt.de/21368/>
- [17] D. Sikeridis, D. Ott, S. Huntley, S. Sharma, V. K. Dhanasekar, M. Bansal, A. Kumar, A. U. N. D. Beveridge, and S. Veeraswamy, "ELCA: Introducing Enterprise-level Cryptographic Agility for a Post-Quantum Era," 2023. [Online]. Available: <https://ia.cr/2023/1539>
- [18] S. Paul and M. Niethammer, "On the importance of cryptographic agility for industrial automation: Preparing industrial systems for the quantum computing era," *At-Automatisierungstechnik*, vol. 67, pp. 402–416, 2019, crypto Agilität braucht: * Agile APIs * Secure Software Update * Dokumentation über Crypto Einsatz (welche Algos an welcher Stelle für welche Funktion).
- [19] M. Green and M. Smith, "Developers are Not the Enemy!: The Need for Usable Security APIs," *IEEE Security and Privacy*, vol. 14, pp. 40–46, 2016.
- [20] M. Georgiev, S. Iyengar, S. Jana, R. Anubhai, D. Boneh, and V. Shmatikov, "The Most Dangerous Code in the World: Validating SSL Certificates in Non-Browser Software," *Proceedings of the 2012 ACM conference on Computer and communications security*, pp. 38–49, 2012.
- [21] G. Tasopoulos, J. Li, A. P. Fournaris, R. K. Zhao, A. Sakzad, and R. Steinfeld, "Performance Evaluation of Post-Quantum TLS 1.3 on Resource-Constrained Embedded Systems," in *Information Security Practice and Experience*, C. Su, D. Gritzalis, and V. Piuri, Eds. Cham: Springer International Publishing, 2022, pp. 432–451.
- [22] O. Kehret, A. Walz, and A. Sikora, "INTEGRATION OF HARDWARE SECURITY MODULES INTO A DEEPLY EMBEDDED TLS STACK," *International Journal of Computing*, vol. 15, pp. 22–30, 2016.
- [23] R. Matischek and B. Bara, "Application Study of Hardware-Based Security for Future Industrial IoT," in *2019 22nd Euromicro Conference on Digital System Design (DSD)*, 2019, pp. 246–252.
- [24] O. Gilles, D. G. Pérez, P. A. Brameret, and V. Lacroix, "Securing IIoT communications using OPC UA PubSub and Trusted Platform Modules," *Journal of Systems Architecture*, vol. 134, p. 102797, 1 2023. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S138376212200282X>
- [25] M. Talamo, M. Galinium, C. H. Schunck, and F. Arcieri, "Secure Messaging Implementation in OpenSC," *Journal of Information Security*, vol. 03, pp. 251–258, 2012.
- [26] M. O'Neill, S. Heidbrink, J. Whitehead, T. Perdue, L. Dickinson, T. Collett, N. Bonner, K. Seamons, and D. Zappala, "The Secure Socket API: TLS as an Operating System Service," in *27th USENIX Security Symposium*, 2018. [Online]. Available: www.usenix.org/conference/usenixsecurity18/presentation/oneill
- [27] M. O'Neill, S. Heidbrink, S. Ruoti, J. Whitehead, D. Bunker, L. Dickinson, T. Hendershot, J. Reynolds, K. Seamons, and D. Zappala, "TrustBase: An Architecture to Repair and Strengthen Certificate-based Authentication," in *26th USENIX Security Symposium (USENIX Security 17)*. USENIX Association, 8 2017, pp. 609–624. [Online]. Available: <https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/oneill>
- [28] A. Salman, M. Rogawski, and J. P. Kaps, "Efficient hardware accelerator for IPsec based on partial reconfiguration on Xilinx FPGAs," *Proceedings - 2011 International Conference on Reconfigurable Computing and FPGAs, ReConFig 2011*, pp. 242–248, 2011.
- [29] D. Kim, S. Lee, and K. Park, "A Case for SmartNIC-accelerated Private Communication," in *4th Asia-Pacific Workshop on Networking*. ACM, 8 2020, pp. 30–35. [Online]. Available: <https://dl.acm.org/doi/10.1145/3411029.3411034>
- [30] B. Pismenny, H. Eran, A. Morrison, D. Tsafir, A. Yehezkel, and L. Liss, "Autonomous NIC Offloads," *ASPLOS '21: Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, vol. 18, 2021. [Online]. Available: <https://doi.org/10.1145/3445814>.
- [31] T. A. Rizzetti, P. Wessel, A. S. Rodrigues, B. M. D. Silva, R. Milbradt, and L. N. Canha, "Cyber security and communications network on SCADA systems in the context of Smart Grids," *Proceedings of the Universities Power Engineering Conference*, vol. 2015–November, 11 2015.
- [32] O. Givehchi, K. Landsdorf, P. Simoens, and A. W. Colombo, "Interoperability for industrial cyber-physical systems: An approach for legacy systems," *IEEE Transactions on Industrial Informatics*, vol. 13, pp. 3370–3378, 12 2017.
- [33] W. Hupp, A. Hasandka, R. S. D. Carvalho, and D. Saleem, "Module-OT: A hardware security module for operational technology," *2020 IEEE Texas Power and Energy Conference, TPEC 2020*, pp. 1–6, 2020.
- [34] T. Frauenschläger and J. Mottok, "Security-Gateway for SCADA-Systems in Critical Infrastructures," in *2022 International Conference on Applied Electronics (AE)*, 2022.

Session We.1.A
ML/AI for Critical System 1

Wednesday 12th June

10:00

–

Auditorium

Software-Only Semantic Diverse Redundancy for High-Integrity AI-Based Functionalities

Martí Caro^{†,‡}, Axel Brando[†], Jaume Abella[†]

[†]Barcelona Supercomputing Center (BSC) [‡]Universitat Politècnica de Catalunya
Barcelona, Spain Barcelona, Spain

Abstract—Dual (DMR) and Triple Modular Redundancy (TMR), often with some form of diversity, are used in safety-critical systems to realize those functionalities at the highest integrity level providing fault detection and/or tolerance capabilities. Redundant executions are intended to provide bit-level identical results and, upon any mismatch, an error is assumed and recovery actions taken as needed.

In this paper, we note that many emerging AI-based functionalities are intrinsically stochastic (e.g., camera-based object detection), and hence, their correctness must be judged semantically, with room for variations across correct outcomes (e.g., confidence must be above a given threshold). Building on this observation, we propose strategies to create DMR and TMR implementations of AI-based functionalities that bring not only fault tolerance against random hardware faults, but also against AI model inaccuracies. Those strategies, which can be realized with software-only means and ported to virtually any computing platform, build on input data modifications affecting the inference computations, but not the expected semantic output (e.g., introducing some limited random noise in the input data).

Index Terms—Diversity, Redundancy, Safety, DNN, AI

I. INTRODUCTION

Artificial Intelligence (AI) is increasingly used in safety-critical systems as an enabler for autonomous operation. For instance, camera-based object detection implemented with Deep Neural Networks (DNN) is used across a wide variety of applications and domains such as autonomous driving cars [1], rendezvous and docking operations in space [2], and in-cabin pilot monitoring in avionics [3] among others.

AI has been used so far mostly in fail-safe systems [4], where AI software has been generally relieved from inheriting safety requirements by incorporating appropriate non-AI monitors. However, autonomous operation, often related to fail-operational systems, does not generally allow keeping AI software without safety requirements. In fact, since AI software is in charge of high-integrity operations, such as steering and braking in automotive, or docking in space, integrity requirements can easily be the highest ones (e.g., DAL-A in avionics, ASIL-D in automotive). Therefore, it is common that the corresponding safety guidelines impose the use of diverse redundancy as a mandatory safety measure.

So far, diverse redundancy has been built with bit-level correctness in mind. For instance, lockstep processors use 2 or 3 identical cores running the same software with just some time staggering so that their internal state differs at any point in time and, upon a fault affecting all of them simultaneously, the errors generated (if any) are expected to differ and, at least, be detected. In this case, any discrepancy in the results is regarded as an error even if it is semantically innocuous for a specific application, since it is hard to tell whether it is or not a priori. However, in the context of AI software, such as DNNs, many applications do not require bit-level correctness and, instead, perform stochastic processes where bit-level

different outcomes can be regarded as semantically identical. For instance, two object detectors identifying the same object class as the most likely one, with confidence values above a detection threshold, and with highly overlapping bounding boxes, can be regarded as providing identical outputs even if confidence values and bounding boxes differ.

This paper exploits this observation, i.e., *redundant AI software may afford bit-level differences across semantically identical outcomes*, to realize diverse redundancy in more efficient ways. In particular, we consider solutions where diverse redundancy can be used not only to deal with random hardware faults, but also to mitigate inaccuracies brought by AI models themselves (e.g., false positives and false negatives). Moreover, we do so containing system design and operation costs by preserving the original AI model. Otherwise, the cost of designing, training, and verifying two models could be prohibitive, as well as the cost of fetching twice the amount of AI model parameters, which may be in the order of several GBs in the case of some DNNs for camera-based object detection [5].

Our solution builds upon applying semantically-neutral transformations to the input data of the AI model (e.g., images for camera-based object detection) that alter inference computations so that random hardware faults are mitigated due to physical redundancy, but also small AI model inaccuracies can be mitigated due to input data diversity. In particular, we realize our transformations in the input images used by the You Only Look Once (YOLO) [6] object detector and show that a variety of image transformations yield comparable results to the original case, and they can be combined with different schemes to form a TMR system.

The rest of the paper is organized as follows. Section II provides some background and related work on diverse redundancy. Section III presents our proposal. Section IV evaluates it. Finally, Section V concludes this work.

II. BACKGROUND AND RELATED WORK

High-integrity systems and components require the use of some form of redundancy for, at least, error detection, and possibly correction. Data storage and communication often build on some form of error correction codes. For instance, single error correction double error detection (SECDED) codes for data storage are a popular solution for large memories, and can also be used to protect end-to-end communications by forwarding those codes along with the data.

However, combinatorial logic, and computing components in general, often need to resort to full redundancy. Dual (DMR) and Triple Modular Redundancy (TMR) are effective solutions for error detection and correction respectively if faults affect one of the redundant components only. However, some types of faults, such as those affecting clock signals and power

networks, can lead to simultaneous errors in all redundant components. Hence, some form of diversity is wanted across redundant instances to avoid scenarios where errors are identical and cannot be detected by comparing outputs.

The usual solution to realize DMR and TMR with diversity in CPUs consists of operating redundant cores with some time staggering so that, despite redundant cores are identical and execute identical software, their state at any time instant differs, and the simultaneous impact of a fault is expected to cause different errors – if any. The most popular case of such scheme is known as Dual Core LockStep (DCLS for short), and realized in some commercial microcontrollers, such as the Infineon AURIX processor family [7]. Similar schemes have been devised for GPUs [8], as well as software-only alternatives for cores [9]. However, all those solutions build upon bit-level accuracy so, while they could be applied directly for DNNs (either in CPU, GPU, or ad-hoc accelerators), they cannot be used to mitigate DNN model inaccuracies.

Some solutions exploit semantic redundancy for data varying little over time (e.g., camera-based object detection at high frames per second rates) [10], [11]. However, differently to our proposal, those solutions only provide error detection capabilities and can only be applied to problems with some (high) degree of timing redundancy. Our approach, instead, provides intrinsic error correction capabilities and poses no requirements on the timing relations across input data.

In our previous work [12], we presented TRUST, a scheme combining temporal redundancy, such as the previous works, as well as spatial redundancy to provide a DCLS-like solution, but with the secondary accelerator operating with lower-precision arithmetic to reduce complexity and power. While TRUST provides error correction capabilities, such as ITLS, it does not increase the model accuracy, and is still bound to problems where temporal redundancy exists. Moreover, ITLS could be realized with software support only, as opposed to TRUST.

Ensemble methods have previously been adopted to enhance the accuracy of DNNs [13], [14]. However, those works introduce substantial memory and computational overheads since weights change across models. To address this challenge, Gao et al. [15] introduced an ensemble approach that combines the outputs of three or four lower-complexity models (e.g., an ensemble of ResNet22, ResNet32, and ResNet44) to replace the original model (e.g., ResNet110). In their study, Gao et al. compare their ensemble technique with a Triple Modular Redundancy (TMR) method, and show that it incurs smaller overheads. However, their ensemble approach requires three different models with different weights, while we rely on the use of a single model. Therefore, their approach increases the cost of designing, training, and verifying several models which may be prohibitive. Furthermore, their evaluation focuses on object classification, which is an easier task than object detection.

Jon’s et al. recent work [16] bears the closest resemblance to ours. Authors propose the use of image transformations to detect – but not correct – anomalous situations that could cause a potential faulty situation in an autonomous system. In particular, authors use a DNN to calculate the steering angle of a robot while driving between two lines, which is a much simpler task than the object detection task considered in our work. Their approach consists in executing the DNN with the original image and then performing a follow-up execution

with a very slightly modified image, such that the steering angles calculated for both images are expected to be identical, i.e. within a calculated uncertainty threshold. However, in our work, we allow a higher degree of modification to be able to increase the overall system robustness through detecting a larger number of objects between the different components of the TMR system. We attempted to apply such approach to our problem, but the results were not promising, since the output of a DNN for object detection is composed of millions of values, and we found that the calculated uncertainty threshold, with the image modifications used in our work, is too large in fault-free cases, such that it cannot detect faulty operations.

III. SOFTWARE-ONLY SEMANTIC DIVERSE REDUNDANCY

Our approach builds on realizing a redundant scheme for the inference with a given AI model by altering the inputs across redundant instances in a way that outputs are expected to remain identical semantically speaking, and merging them conveniently, as illustrated in Figure 1. Note that to achieve maximum performance, a hardware solution comprising three parallel physical compute units must be made available. However, if developing a hardware solution is not an option, our solution could be implemented solely through software means. This can be achieved by executing the three input images in parallel on a single GPU, following the work from S. Alcaide et al. [17] which demonstrates how to implement a software-only TMR scheme using a single GPU.

For the sake of easing the explanation, we focus on a TMR scheme applied on the popular You Only Look Once v4 (YOLOv4) [6] object detection software for images, which is used in a plethora of applications, including commercial automotive systems, such as Apollo [18]. In particular, the task at hand is object detection, where, for each object present within an image, the model has to classify it (i.e., determine the class of the object, such as, a vehicle or a person), and determine its position and size by creating a bounding box (i.e., a rectangle surrounding the object).

A. Efficient Diverse Redundancy

As shown in some recent work, DNNs such as the one in YOLOv4 may require fetching some GBs of weights to process each image [5], whereas images occupy up to few MBs. Hence, memory bandwidth is often saturated to fetch the DNN weights. Based on this fact, our redundancy scheme aims at preserving the same DNN model for all redundant inference processes with the aim of enabling appropriate execution approaches that allow sharing weights fetched across the redundant processes.

Therefore, if weights are fixed, the simplest way to introduce diversity with software-only means consists of altering the input data. Our goal is applying modifications to the input data so that it differs across the redundant processes, but they are semantically equivalent so that inference should lead to the same object detections. In particular, we have considered some of the image transformations provided by the CLoDSA open library [19], which would map to $F(x)$, $G(x)$ and $H(x)$ in the figure:

- Applying histogram equalization (EQ).
- Image sharpening (SH).
- Dropout (DR) by setting some pixels to zero.
- Applying Gamma correction (GC).
- Blurring the image applying a Gaussian filter (GB).

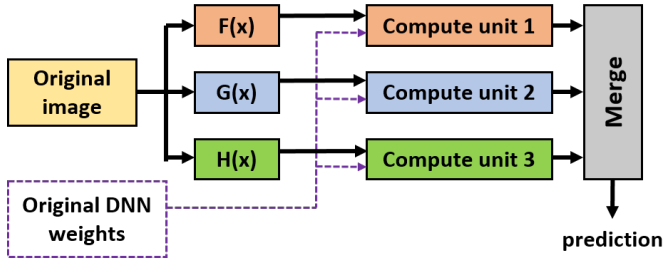


Fig. 1: Overview of our proposed TMR diverse redundancy scheme.

- Applying Gaussian noise (GN).
- Blurring the image applying the median filter (MB).
- Adding salt and pepper pixels (SP).
- Raising pixel values by a set amount (RV).
- Shifting the image some pixels to the right (RS).
- Shifting the image some pixels to the left (LS).
- Shifting the image some pixels to the top (TS).
- Shifting the image some pixels to the bottom (BS).
- Rotating the image clockwise at a certain angle (CR).
- Rotating the image anticlockwise at a certain angle (AR).
- Horizontally flipping the image (HF).
- Vertically flipping the image (VF).

To realize the TMR scheme, we select three different choices between the image transformations above and the original image, and perform inference for the three of them using the identical YOLOv4 DNN model. With this, we obtain three potentially different object detection outputs that need being combined properly.

B. Combining Predictions

Rather than attempting to identify wrong predictions – if any – across the diverse and redundant predictions, we merge those predictions with the aim of making correct predictions supersede erroneous ones (see Figure 1). We build on the fact that predictions, even if correct, may not be identical at bit level. Hence, we opt for combining predictions whose detection class matches (e.g., a pedestrian is detected), and whose bounding box overlaps enough (e.g., bounding boxes have an intersection over union higher than 50%). However, it remains open how to set the final bounding box and confidence for the detection. Regarding the bounding box, note that model A can provide a bounding box overlapping above the threshold with the one from model B and the one from model C, but the ones from models B and C may overlap below the threshold. In this case, we still regard detections as the same detection and attempt to combine them.

Confidence. To set the confidence of the detection, we explore different schemes as follows:

- **Voting:** a detection is regarded as true if 2 out of 3 DNN models provide a confidence level above the detection threshold (e.g., using the default confidence threshold of 50% to determine that a detection should be correct).
- **Averaging:** a detection is regarded as true if the average of the confidence values for the three models is above the detection threshold.
- **Maximum:** a detection is regarded as true if at least one model indicates a confidence level above the detection threshold.

Bounding box. To set the bounding box of the detection, we perform it as follows:

- **Voting:** we retain the bounding box of the detection with the highest confidence among those being combined.
- **Averaging:** the bounding boxes of the 3 DNN models are averaged to obtain a final bounding box.
- **Maximum:** note that in this case, there may be overlapping bounding boxes that represent the same object. Therefore, a Non-Maximum Suppression (NMS) post-processing is applied to filter out repetitions. The NMS consists in filtering out the detections that represent the same object by comparing the bounding boxes of the detections of the same class, and if the bounding boxes have sufficient overlap (i.e., Intersection Over Union > 0.5), the bounding box with the highest confidence is kept, and the other overlapping bounding boxes are discarded.

Note that other approaches to determine the resulting bounding box are possible, such as selecting the largest bounding box among those being combined instead of considering the confidence level. However, using a bounding box not provided by any of the DNNs explicitly brings additional uncertainty. Hence, we stick to using a method that selects one of the relevant bounding boxes.

C. Application to Safety-Relevant Systems

Deploying safety-relevant systems based on AI, with AI software inhering safety requirements, is still an open challenge [4]. Some relevant standards have seen the light very recently, such as ISO 5469 and ISO 21448 (aka SOTIF), but their practical application is not yet solved. Moreover, they do not provide specific guidance to tailor diverse redundancy. For instance, it is unclear how many levels and degrees of diverse redundancy are needed to make risk residual and system accuracy high enough. In our view, as part of the system architecture, diverse redundancy must exist at different levels, e.g., with multiple sensors, and with redundant and diverse processing for the data of each sensor. Our proposal aims at providing an efficient realization for the latter, hence enabling system designers with appropriate solutions to architect their AI-based safety-critical systems.

IV. EVALUATION

In this section, we introduce the evaluation setup (YOLO implementation, dataset, evaluation metrics, and fault injection framework), we discuss the results obtained with each image transformation when executing a single model, and we discuss the results obtained with our scheme, considering the different image transformations and results merging schemes in fault-free and faulty scenarios.

A. Setup

We use a 32-bit floating point Tensorflow Keras implementation of YOLOv4 [20] with the publicly available pre-trained YOLOv4 parameters with the training subset of the Common Objects in Context (COCO) dataset [21]. We evaluate our proposal using the default 608x608 (image width x image height) network size of YOLOv4 and a reduced version using a 320x320 network size which roughly gives a 2x inference speed up at the cost of a slight accuracy drop [22], hence showing the effectiveness of our scheme regardless of the chosen network size. We evaluate our proposal with COCO [21],

using the validation subset, and keeping only those images that contain objects such as vehicles and pedestrians, which delves 870 images for evaluation.

However, the COCO dataset is not specific to AD, hence we have also evaluated our scheme with the KITTI dataset, designed specifically for AD applications. We have also considered using other AD datasets such as Udacity [23], Berkeley DeepDrive [24], and Waymo [25], but KITTI was the only one delivering relevant results for the pretrained YOLOv4 model due to using a comparable object labelling policies to the COCO dataset used for training. The KITTI dataset contains 7481 images captured from onboard vehicle cameras, and also includes three temporally preceding frames of each image, but captured at a very low FPS, which makes detections across images highly independent. Hence, we resorted to evaluating each image independently. The main limitation when using the KITTI dataset with the pretrained YOLOv4 model is that some classes (e.g., bus) are not labelled, and some classes (e.g., person, and motorbike) have significantly different labelling policies to those of COCO. Therefore, we have restricted our evaluation with the KITTI dataset to those classes with fewer discrepancies to obtain accurate results (i.e., car, van, and truck object classes). Another difference w.r.t. the COCO dataset is that the KITTI dataset includes some regions of the image – mostly background regions – that have not been labelled. Therefore, predictions made within these unlabelled zones are ignored during the evaluation process.

As evaluation metrics, we use the *Accuracy* (ACC) and *Mean Average Precision* (mAP). To obtain those, first we have to compute the *Intersection over Union* (IoU), i.e., the fraction of the intersection of the bounding boxes w.r.t. the union of those bounding boxes. In particular, we do so for the final objects detected by our proposal and the groundtruths. Since the detection process is stochastic and subject to some variation, whether an object is regarded as detected or not is done with a threshold, which in our case is 0.5, as set in other works [26]. This leads to true positives (TP) if the IoU is above the threshold, false positives (FP) if the IoU is below the threshold for a predicted object, and false negatives (FN) if the IoU of a groundtruth is below the threshold.

The accuracy (ACC) is obtained as follows:

$$ACC = \frac{TP}{TP + FP + FN} \quad (1)$$

The mAP is a more complex metric that is often the reference for object detection evaluation. The mAP leverages the TP, FP, and FN counts across the existing object classes to obtain an accuracy assessment for each object class, and then averages the individual accuracies to obtain an overall accuracy assessment. Given the complexity to introduce the details of this metric, and due to space constraints, we refer the interested reader to other publications for detailed explanations of this metric [27].

Regarding our fault injection campaign, we focus on random hardware faults impacting the computation (i.e., transient faults). We have injected transient faults in the result generated by multiplication or addition operations of the convolutional layers, as these layers account for over 99.5% of the total number of operations to process an image. Furthermore, random faults have only been considered to affect the sign or exponent of the floating-point number representation (i.e., 9 highest order bits), since random bit flips in the mantissa

Configuration	TP	FP	FN	ACC	mAP
BL	3087	373	3104	47,03%	48,69%
RV	3085	372	3106	47,01%	48,65%
GC	3069	375	3122	46,74%	48,38%
TS	3061	353	3130	46,78%	48,27%
HF	3058	378	3133	46,55%	48,16%
BS	3050	358	3141	46,57%	48,14%
LS	3046	336	3145	46,67%	48,09%
RS	3037	379	3154	46,23%	47,84%
GN	3021	348	3170	46,20%	47,67%
EQ	3002	370	3189	45,76%	47,38%
SP	2991	341	3200	45,79%	47,16%
AR	2843	307	3348	43,75%	44,87%
CR	2834	292	3357	43,71%	44,85%
GB	2706	231	3485	42,14%	42,97%
DR	2659	287	3532	41,05%	42,03%
SH	2618	278	3573	40,47%	41,32%
MB	2550	239	3641	39,66%	40,39%
VF	317	36	5874	5,09%	4,88%

TABLE I: Results of each Image Transformation analysed independently with the COCO dataset (320x320 network size).

Configuration	TP	FP	FN	ACC	mAP
EQ	23023	3035	9727	64,34%	68,27%
GC	22737	2970	10013	63,65%	67,37%
TS	22424	2861	10326	62,97%	66,53%
GB	22323	2380	10427	63,54%	66,50%
RV	22375	2766	10375	63,00%	66,42%
BS	22301	2732	10449	62,85%	66,19%
HF	22298	2737	10452	62,83%	66,12%
RS	22267	2837	10483	62,57%	66,10%
BL	22247	2742	10503	62,68%	66,07%
LS	22229	2704	10521	62,70%	65,97%
CR	21464	2250	11286	61,33%	63,93%
AR	21425	2163	11325	61,37%	63,88%
GN	21338	2543	11412	60,46%	63,34%
SP	20805	2467	11945	59,08%	61,71%
MB	20556	2022	12194	59,12%	61,32%
SH	18424	2270	14326	52,61%	54,54%
DR	14882	1788	17868	43,09%	44,00%
VF	229	42	32521	0,70%	0,62%

TABLE II: Results of each Image Transformation analysed independently with the KITTI dataset (320x320 network size).

are mostly masked and do not lead to semantic changes in the outputs of the model [28], [29].

Tensorflow Keras poses difficulties to inject faults in the intermediate results of a layer, since layers operate as black boxes. To overcome this limitation, we have implemented a custom convolutional layer (same functionality but less efficient implementation). We choose an operation (addition or multiplication) randomly across all those operations in all convolutional layers. To inject a fault in a specific result of a specific layer, we execute the Keras model until the previous layer, replace the target layer with our custom one, inject the fault as needed (flipping a random bit in the sign or exponent), and resume the execution of the Keras implementation from the following layer onwards passing the result of our custom layer as needed.

B. Results of Independent Configurations

Tables I, II, III, and IV show the TP, FP, and FN counts, as well as the ACC and mAP for each image transformation with

Configuration	TP	FP	FN	ACC	mAP
TS	3887	473	2304	58,33%	61,12%
RV	3868	467	2323	58,10%	60,94%
HF	3866	456	2325	58,16%	60,88%
LS	3856	435	2335	58,19%	60,88%
BL	3865	466	2326	58,06%	60,87%
BS	3860	439	2331	58,22%	60,82%
RS	3856	446	2335	58,10%	60,76%
GC	3859	470	2332	57,93%	60,74%
GN	3776	405	2415	57,25%	59,61%
SP	3708	398	2483	56,28%	58,55%
EQ	3699	450	2492	55,70%	58,24%
CR	3676	466	2515	55,22%	57,83%
AR	3675	504	2516	54,89%	57,72%
SH	3240	324	2951	49,73%	51,15%
GB	3213	267	2978	49,75%	51,05%
MB	2894	295	3297	44,62%	45,71%
DR	2335	298	3856	35,98%	36,40%
VF	366	34	5825	5,88%	5,60%

TABLE III: Results of each Image Transformation analysed independently with the COCO dataset (608x608 network size).

Configuration	TP	FP	FN	ACC	mAP
EQ	26193	4435	6557	70,44%	77,59%
GC	26093	4401	6657	70,23%	77,14%
BS	25930	4047	6820	70,47%	76,86%
BL	25935	4093	6815	70,39%	76,86%
RV	25949	4142	6801	70,34%	76,85%
HF	25894	3998	6856	70,46%	76,75%
LS	25890	4132	6860	70,20%	76,73%
TS	25822	4165	6928	69,95%	76,50%
RS	25811	4135	6939	69,98%	76,46%
AR	25607	3488	7143	70,66%	76,03%
CR	25533	3542	7217	70,35%	75,81%
GN	25548	3880	7202	69,75%	75,64%
SP	25229	3763	7521	69,10%	74,65%
GB	24796	2900	7954	69,55%	73,93%
SH	24765	3301	7985	68,69%	73,21%
MB	22545	2172	10205	64,56%	67,36%
DR	13191	1239	19559	38,81%	38,83%
VF	568	86	32182	1,73%	1,56%

TABLE IV: Results of each Image Transformation analysed independently with the KITTI dataset (608x608 network size).

the COCO and KITTI datasets using 320x320 and 608x608 network sizes.

First, note that the VF image transformation produces very low ACC and mAP results for all the datasets and network sizes evaluated, since the pretrained model has not been trained with vertically flipped objects. However, we keep this transformation for completeness.

In the case of COCO with 320x320 network size, the Baseline configuration provides the highest ACC and mAP, but with a network size of 608x608 there are 4 image transformations (TS, RV, HF, LS) providing up to 0.25% higher mAP than the baseline. For KITTI and 320x320 network size, there are 8 image transformations providing up to 2.2% higher mAP than the baseline, and with a 608x608 network size there are 3 image transformations (EQ, GC, BS) providing up to 0.73% higher mAP than the baseline.

Note that the pretrained YOLO model used COCO's training subset. Hence, the baseline model was expected to obtain high accuracy when evaluating similar images (e.g., COCO

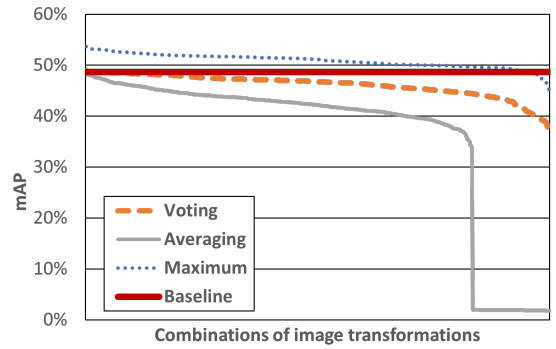


Fig. 2: Sorted mAP for all TMR $F(x)$, $G(X)$, and $H(x)$ permutations for all merging algorithms using COCO (320x320 network size).

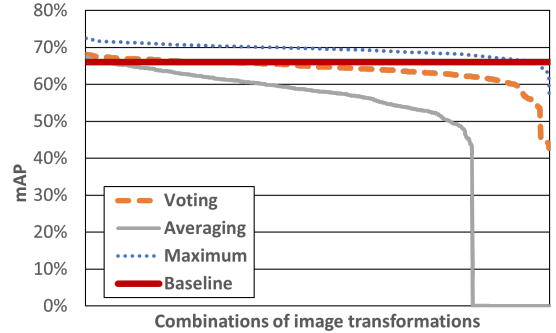


Fig. 3: Sorted mAP for all TMR $F(x)$, $G(X)$, and $H(x)$ permutations for all merging algorithms using KITTI (320x320 network size).

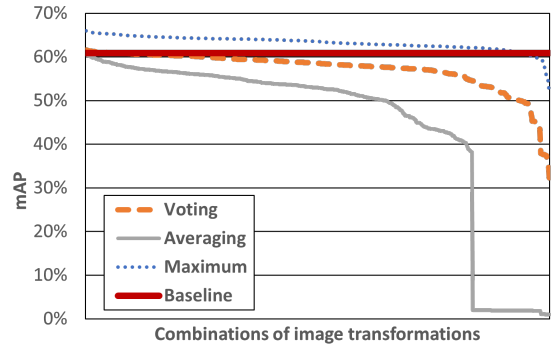


Fig. 4: Sorted mAP for all TMR $F(x)$, $G(X)$, and $H(x)$ permutations for all merging algorithms using COCO (608x608 network size).

validation subset), but for different datasets, such as KITTI, it was indeed expected that some image transformations could surpass the accuracy of the baseline model.

C. Results of the Merging Algorithms

We have obtained the TP, FP, and FN counts, as well as the ACC and mAP values for all TMR configurations and merging algorithms, namely Voting, Averaging and Maximum. Since the number of combinations is too large to allow reporting data for all those configurations (817 per merging algorithm), we show the mAP for all configurations in Figures 2, 3, 4,

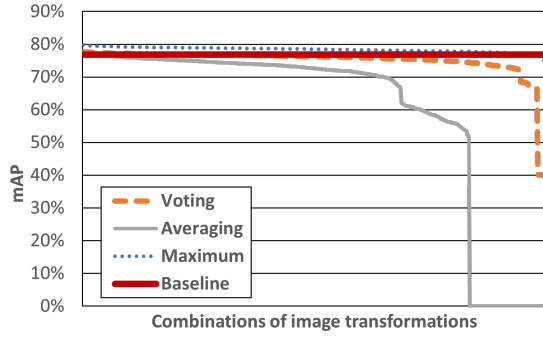


Fig. 5: Sorted mAP for all TMR $F(x)$, $G(X)$, and $H(x)$ permutations for all merging algorithms using KITTI (608x608 network size).

BASELINE					
Configuration	TP	FP	FN	ACC	mAP
BL	3087	373	3104	47,0%	48,69%
VOTING					
Configuration	TP	FP	FN	ACC	mAP
BL, RV, BS	3098	357	3093	47,31%	48,97%
RV, LS, BS	3091	290	3100	47,69%	48,95%
BL, LS, BS	3087	296	3104	47,59%	48,87%
BL, RV, TS	3093	362	3098	47,20%	48,85%
GC, LS, BS	3083	283	3108	47,62%	48,85%
AVERAGING					
Configuration	TP	FP	FN	ACC	mAP
BL, BL, BL	3087	373	3104	47,03%	48,69%
BL, GC, RV	3079	370	3112	46,93%	48,59%
BL, GN, RV	3057	369	3134	46,60%	48,24%
GC, GN, RV	3052	371	3139	46,51%	48,15%
BL, GC, GN	3050	363	3141	46,54%	48,15%
MAXIMUM					
Configuration	TP	FP	FN	ACC	mAP
HF, RS, TS	3435	630	2756	50,36%	53,65%
HF, RS, BS	3432	645	2759	50,20%	53,56%
HF, LS, RS	3422	620	2769	50,24%	53,48%
HF, LS, BS	3418	616	2773	50,21%	53,45%
EQ, LS, TS	3409	606	2782	50,15%	53,39%

TABLE V: TMR results for the top-5 configurations of each merging algorithm using COCO (320x320 network size).

and 5, and the detailed results for the top-5 (in terms of mAP) for each algorithm in Tables V, VI, VII, and VIII.

Results in the figure are sorted independently for each merging algorithm, meaning that the n^{th} best configuration for one algorithm may differ for the other algorithms despite being aligned w.r.t. the X-axis. Hence, the X-axis is not labelled. However, showing the data this way allows us to get several conclusions: (1) Maximum provides consistently the best results in terms of mAP across all merging algorithms; (2) Averaging is consistently worse, in terms of mAP, than the baseline when using the COCO dataset, but it is slightly better than the baseline in very few cases when evaluating the KITTI dataset; (3) Voting is slightly better than the baseline in some cases; and (4) all merging algorithms provide a smooth degradation of the mAP across image transformation combinations, hence meaning that, while some transformations may delve better results than others, there is not a strong dependence on very few combinations, so the approach provides robust

BASELINE					
Configuration	TP	FP	FN	ACC	mAP
BL	22247	2742	10503	62,68%	66,07%
VOTING					
Configuration	TP	FP	FN	ACC	mAP
EQ, GB, GC	22852	2614	9898	64,62%	68,10%
EQ, GC, LS	22867	2584	9883	64,72%	68,03%
EQ, GC, BS	22879	2642	9871	64,64%	68,02%
EQ, GC, TS	22875	2645	9875	64,63%	68,02%
HF, EQ, GC	22868	2651	9882	64,60%	67,98%
AVERAGING					
Configuration	TP	FP	FN	ACC	mAP
EQ, GB, GC	22700	2337	10050	64,70%	67,54%
EQ, GC, RV	22690	2704	10060	64,00%	67,39%
HF, EQ, GC	22636	2429	10114	64,35%	67,33%
BL, EQ, GC	22649	2667	10101	63,95%	67,28%
EQ, GB, RV	22569	2292	10181	64,41%	67,18%
MAXIMUM					
Configuration	TP	FP	FN	ACC	mAP
EQ, GB, RS	24495	4609	8255	65,57%	72,53%
EQ, RS, BS	24536	4808	8214	65,33%	72,46%
EQ, RS, TS	24517	4908	8233	65,10%	72,44%
EQ, GB, LS	24458	4503	8292	65,65%	72,43%
EQ, LS, BS	24506	4756	8244	65,34%	72,39%

TABLE VI: TMR results for the top-5 configurations of each merging algorithm using KITTI (320x320 network size).

BASELINE					
Configuration	TP	FP	FN	ACC	mAP
BL	3865	466	2326	58,06%	60,87%
VOTING					
Configuration	TP	FP	FN	ACC	mAP
HF, LS, TS	3901	403	2290	59,16%	61,60%
RV, LS, TS	3900	416	2291	59,03%	61,54%
GC, LS, TS	3897	410	2294	59,04%	61,49%
HF, RS, TS	3895	422	2296	58,90%	61,48%
HF, RV, BS	3893	417	2298	58,91%	61,43%
AVERAGING					
Configuration	TP	FP	FN	ACC	mAP
BL, BL, BL	3865	466	2326	58,06%	60,87%
BL, GC, RV	3862	460	2329	58,07%	60,84%
BL, HF, GC	3829	423	2362	57,89%	60,42%
BL, GN, RV	3827	425	2364	57,84%	60,37%
HF, GC, RV	3825	421	2366	57,85%	60,35%
MAXIMUM					
Configuration	TP	FP	FN	ACC	mAP
HF, RS, BS	4214	691	1977	61,23%	66,01%
HF, LS, TS	4207	713	1984	60,94%	65,87%
HF, LS, BS	4201	697	1990	60,99%	65,84%
HF, RS, TS	4204	715	1987	60,87%	65,82%
RS, LS, TS	4195	710	1996	60,79%	65,72%

TABLE VII: TMR results for the top-5 configurations of each merging algorithm using COCO (608x608 network size).

results.

Results in the tables offer a different angle to our analysis. For the COCO 320x320 configurations, in the case of Voting, we note that the bottom shift (BS), left shift (LS), and raise value (RV) transformations, as well as the baseline (BL), are the most popular ones since they appear in 4, 3, 3, and 3 of the top-5 TMR combinations respectively, and only Gamma correction (GC) appears once. In the case of Averaging, the best combination, despite not providing diversity, consists of

BASELINE					
Configuration	TP	FP	FN	ACC	mAP
BL	25935	4093	6815	70,39%	76,86%
VOTING					
Configuration	TP	FP	FN	ACC	mAP
HF, EQ, GC	26199	4101	6551	71,09%	77,76%
EQ, GC, LS	26196	4105	6554	71,08%	77,74%
EQ, GC, RS	26184	4083	6566	71,09%	77,70%
EQ, GC, BS	26170	4122	6580	70,98%	77,66%
EQ, LS, BS	26118	3862	6632	71,34%	77,61%
AVERAGING					
Configuration	TP	FP	FN	ACC	mAP
HF, EQ, GC	26057	3879	6693	71,14%	77,32%
BL, HF, EQ	26011	3748	6739	71,27%	77,24%
EQ, GC, RV	26064	4152	6686	70,63%	77,21%
HF, EQ, RV	26007	3781	6743	71,19%	77,20%
BL, EQ, GC	26052	4124	6698	70,65%	77,20%
MAXIMUM					
Configuration	TP	FP	FN	ACC	mAP
HF, EQ, LS	26974	5877	5776	69,83%	79,80%
HF, EQ, BS	26963	5717	5787	70,09%	79,79%
HF, EQ, TS	26949	5772	5801	69,96%	79,76%
EQ, LS, BS	26947	5863	5803	69,79%	79,73%
EQ, LS, TS	26947	5901	5803	69,72%	79,72%

TABLE VIII: TMR results for the top-5 configurations of each merging algorithm using KITTI (608x608 network size).

using 3 times the BL. RV, GC, and Gaussian Noise (GN) are also quite frequent, and appear 3 times each one. Finally, in the case of Maximum merging, we observe that slightly shifting the original image in one direction is a frequent choice since 10 out of the 15 choices correspond to top, bottom, left or right shift. Hflip (HF) is also a frequent choice with 4 appearances, and Equalization (EQ) appears once.

For the COCO 608x608 configurations, in the case of Voting, we note that slightly shifting the original image in one direction is a frequent choice since 9 out of the 15 choices correspond to top, bottom, left or right shift, HF appears 3 times, RV appears twice, and GC appears once. In the case of Averaging, the best combination, despite not providing diversity, consists of using 3 times the BL. RV and GC appear three times, HF appears twice, and GN appears once. Finally, in the case of Maximum merging, we observe that slightly shifting the original image in one direction is a frequent choice since 11 out of the 15 choices correspond to top, bottom, left or right shift. HF is the second most frequent choice with 4 appearances.

For the KITTI 320x320 configurations, in the case of Voting, EQ and GC appear 5 times, shifting transformations appear 3 times, and HF appears once. In the case of Averaging the best configuration is no longer the baseline. EQ appears 5 times, GC appears 4 times, GB appears twice, and RV, HF, and BL appear once. Finally, in the case of Maximum merging, shifting appears 8 times, EQ 5 times, and GB appears twice.

For the KITTI 608x608 configurations, in the case of Voting, EQ and shifting appear 5 times each, GC appears 4 times, and HF appears once. In the case of Averaging, EQ appears 5 times, HF and GC appear 3 times, RV and BL appear twice. In the case of Averaging, EQ appears 5 times, and HF and GC appear 3 times. Finally, in the case of Maximum merging, shifting appears 7 times, EQ appears 5 times, and HF appears 3 times.

In summary, for the KITTI dataset, EQ is a particularly good option since it appears in all the TOP-5 configurations for all merging algorithms. The shifting image transformations works particularly well for both COCO and KITTI datasets when using a Voting or Maximum merging algorithm. The BL works particularly well for the COCO dataset when performing an Average merging, and the HF and GC are also noteworthy configurations for the KITTI dataset with an Average merging.

For the COCO dataset with both 320x320 and 608x608 network sizes, we note that Voting may increase a bit TPs and decrease a bit FPs. Hence, despite not providing the best mAP values, it provides an interesting tradeoff since it outperforms the baseline in all fronts. Averaging, instead, tends to decrease TPs while failing to decrease FPs sufficiently, and it is systematically worse than the baseline case. Finally, Maximum tends to increase TPs and FPs, which is expected since any object being detected by at least one of the three redundant inferences is regarded as a detection, and hence, a TP or a FP. Still, the combined effect clearly increases ACC and mAP values w.r.t. the baseline. Overall, looking at the results from the COCO dataset, if we care only about mAP or ACC, Maximum is clearly the best choice. If, instead, FPs are particularly problematic, Voting is the best solution.

For the KITTI dataset with both 320x320 and 608x608 network sizes, looking at the mAP values, Voting is clearly superior than Averaging for all configurations. However, looking at the ACC values, these approaches have closer results. Finally, Maximum tends to increase TPs and FPs, but the combined effect provides the highest mAP values. However, the ACC of the Maximum merging algorithm is slightly lower than the baseline for the KITTI dataset with a 608x608 network size. We ascribe this effect to the fact that the KITTI dataset does not properly label all background objects, which are mostly small. When increasing the network size, the model can detect more smaller objects, hence we observe a FP increase. We validated this observation in a subset of images by means of visual inspection, yet could not properly label the full dataset manually to fully fix this issue.

Overall, if we care only about mAP, Maximum merging is clearly the best choice for all the datasets analysed. If, instead, FPs are particularly problematic, Voting is the best solution.

D. Results of Fault Injection Configurations

We have selected the top-mAP and top-ACC configurations for each merging algorithm, network size, and dataset, and analysed the impact of random fault injections on this subset of configurations. In the case of a single configuration providing both the top-mAP and top-ACC, this configuration with both the top-mAP and top-ACC, and the second highest mAP configuration have been selected instead.

First, we analyse the individual image transformations used in any of the top-mAP and top-ACC combinations indicated above. Tables IX, X, XI, and XII, present the fault-free results as well as the results after fault injection for those image transformations for the COCO 320x320, KITTI 320x320, COCO 608x608, and KITTI 608x608 setups, respectively. The tables provide the FP, FN, and TP count, as well as the ACC and mAP. Since some combinations (e.g., COCO 320x320 Averaging) use an ensemble of up to three baselines, we perform 3 different fault injections in such baseline, which we refer to as BL_1 AF, BL_2 AF, and BL_3 AF. For the

BASELINE					
Configuration	TP	FP	FN	ACC	mAP
BL	3087	373	3104	47,03%	48,09%
INDEPENDENT CONFIGURATIONS					
Configuration	TP	FP	FN	ACC	mAP
BL_1 AF	2866	379	3325	43,62%	44,57%
BL_2 AF	2889	383	3302	43,95%	44,75%
BL_3 AF	2896	380	3295	44,07%	45,02%
LS	3046	336	3145	46,67%	48,09%
LS AF	2864	351	3327	43,78%	44,25%
TS	3061	353	3130	46,78%	48,27%
TS AF	2873	362	3318	43,84%	44,53%
HF	3058	378	3133	46,55%	48,16%
HF AF	2936	391	3255	44,61%	45,87%
RV	3085	372	3106	47,01%	48,27%
RV AF	2913	381	3278	44,32%	45,36%
RS	3037	379	3154	46,23%	48,09%
RS AF	2830	378	3361	43,08%	44,03%
BS	3050	358	3141	46,57%	48,14%
BS AF	2887	376	3304	43,96%	44,73%
GC	3069	375	3122	46,74%	48,38%
GC AF	2910	377	3281	44,31%	45,43%

TABLE IX: Results of the Fault Injection of the individual image transformations part of the combinations with the best mAP and ACC for the COCO dataset (320x320 network size).

BASELINE					
Configuration	TP	FP	FN	ACC	mAP
BL	3865	466	2326	58,06%	60,87%
INDEPENDENT CONFIGURATIONS					
Configuration	TP	FP	FN	ACC	mAP
BL_1 AF	3667	487	2524	54,91%	56,89%
BL_2 AF	3661	463	2530	55,02%	57,01%
BL_3 AF	3633	468	2558	54,56%	56,47%
HF	3866	456	2325	58,16%	60,88%
HF AF	3708	450	2483	55,83%	57,79%
TS	3887	473	2304	58,33%	61,12%
TS AF	3703	471	2488	55,58%	57,74%
BS	3860	439	2331	58,22%	60,82%
BS AF	3681	408	2510	55,78%	57,80%
RS	3856	446	2335	58,10%	60,76%
RS AF	3653	439	2538	55,10%	56,90%
LS	3856	435	2335	58,19%	60,88%
LS AF	3667	418	2524	55,48%	57,41%
RV	3868	467	2323	58,10%	60,94%
RV AF	3664	440	2527	55,26%	57,41%
GC	3859	470	2332	57,93%	60,74%
GC AF	3710	461	2481	55,77%	58,11%

TABLE XI: Results of the Fault Injection of the individual image transformations part of the combinations with the best mAP and ACC for the COCO dataset (608x608 network size).

BASELINE					
Configuration	TP	FP	FN	ACC	mAP
BL	22247	2742	10503	62,68%	66,07%
INDEPENDENT CONFIGURATIONS					
Configuration	TP	FP	FN	ACC	mAP
EQ	23023	3035	9727	64,34%	68,27%
EQ AF	21977	2945	10773	61,57%	64,90%
LS	22229	2704	10521	62,70%	65,97%
LS AF	21000	2606	11750	59,40%	62,11%
GB	22323	2380	10427	63,54%	66,50%
GB AF	21081	2324	11669	60,10%	62,61%
GC	22737	2970	10013	63,65%	67,37%
GC AF	21564	2897	11186	60,49%	63,60%
RS	22267	2837	10483	62,57%	66,10%
RS AF	21114	2742	11636	59,49%	62,51%
BS	22301	2732	10449	62,85%	66,19%
BS AF	21132	2616	11618	59,75%	62,43%
RV	22375	2766	10375	63,00%	66,42%
RV AF	21050	2693	11700	59,39%	62,24%

TABLE X: Results of the Fault Injection of the individual image transformations part of the combinations with the best mAP and ACC for the KITTI dataset (320x320 network size).

BASELINE					
Configuration	TP	FP	FN	ACC	mAP
BL	25935	4093	6815	70,39%	76,86%
INDEPENDENT CONFIGURATIONS					
Configuration	TP	FP	FN	ACC	mAP
HF	25894	3998	6856	70,46%	76,75%
HF AF	24774	3832	7976	67,72%	73,30%
EQ	26193	4435	6557	70,44%	77,59%
EQ AF	24914	4218	7836	67,39%	73,66%
GB	24796	2900	7954	69,55%	73,93%
GB AF	23548	2767	9202	66,30%	69,96%
AR	25607	3488	7143	70,66%	76,03%
AR AF	24431	3348	8319	67,68%	72,37%
GC	26093	4401	6657	70,23%	77,14%
GC AF	24925	4221	7825	67,42%	73,50%
LS	25890	4132	6860	70,20%	76,73%
LS AF	24671	3998	8079	67,14%	72,90%
CR	25533	3542	7217	70,35%	75,81%
CR AF	24278	3399	8472	67,16%	71,91%

TABLE XII: Results of the Fault Injection of the individual image transformations part of the combinations with the best mAP and ACC for the KITTI dataset (608x608 network size).

remaining cases we report the fault-free (e.g., LS) and fault-injected cases (e.g., LS AF).

We observe that, in general, fault-injected configurations have lower TP counts, and except in COCO 320x320, also lower FP counts. ACC and mAP drop similarly for all configurations, with a 2.90% and 3.56% drop on average, respectively.

Tables XIII, XIV, XV, and XVI show the results of our proposal in the non-faulty TMR case, as well as in two faulty TMR cases; (i) with independents faults where for a given image only one of the three configurations is affected by a fault, and (ii) with faults in the same image for all three configurations. In the fault-free scenarios, which would correspond to virtually 100% of the time given that faults occur

seldom, we obtain the following conclusion: (1) Voting and Maximum merging algorithms provide higher mAP and ACC than the Baseline, except for the HF, EQ, LS configuration for the KITTI dataset with a 608x608 network size, where we observe a slight ACC drop w.r.t. the baseline (for reasons previously discussed in Section IV-C), (2) Averaging only produces better results than the baseline with the KITTI dataset since the model was trained with the COCO training subset. Therefore, it is expected that some image transformations could surpass the accuracy of the baseline when using other datasets.

When faults occur independently, we obtain the following conclusions: (1) Voting provides higher mAP and ACC than the Baseline with both datasets and both network sizes

BASELINE					
Configuration	TP	FP	FN	ACC	mAP
BL	3087	373	3104	47,03%	48,09%
VOTING					
Configuration	TP	FP	FN	ACC	mAP
BL_1, RV, BS	3098	357	3093	47,31%	48,69%
BL_1, RV, BS (indep. faults)	3071	356	3120	46,91%	48,55%
BL_1, RV, BS (same frame)	3029	343	3162	46,36%	47,89%
RV, LS, BS	3091	290	3100	47,69%	48,66%
RV, LS, BS (indep. faults)	3058	291	3133	47,18%	48,39%
RV, LS, BS (same frame)	3028	283	3163	46,77%	47,92%
AVERAGING					
Configuration	TP	FP	FN	ACC	mAP
BL_1, BL_2, BL_3	3087	373	3104	47,03%	48,69%
BL_1, BL_2, BL_3 (indep. faults)	2875	414	3316	43,53%	45,13%
BL_1, BL_2, BL_3 (same frame)	2594	447	3597	39,08%	40,50%
BL_1, GC, RV	3079	370	3112	46,93%	48,66%
BL_1, GC, RV (indep. faults)	2945	423	3246	44,53%	46,31%
BL_1, GC, RV (same frame)	2607	459	3584	39,20%	40,73%
MAXIMUM					
Configuration	TP	FP	FN	ACC	mAP
HF, RS, TS	3435	630	2756	50,36%	52,00%
HF, RS, TS (indep. faults)	3415	761	2776	49,12%	50,24%
HF, RS, TS (same frame)	3393	1122	2798	46,40%	44,17%
HF, RS, BS	3432	645	2759	50,20%	51,98%
HF, RS, BS (indep. faults)	3412	788	2779	48,89%	49,75%
HF, RS, BS (same frame)	3398	1130	2793	46,41%	44,26%

TABLE XIII: TMR results for the best mAP and ACC configurations of each merging algorithm using the COCO dataset (320x320 network size).

BASELINE					
Configuration	TP	FP	FN	ACC	mAP
BL	22247	2742	10503	62,68%	66,07%
VOTING					
Configuration	TP	FP	FN	ACC	mAP
EQ, GB, GC	22852	2614	9898	64,62%	68,10%
EQ, GB, GC (indep. faults)	22796	2587	9954	64,51%	67,95%
EQ, GB, GC (same image)	22501	2516	10249	63,80%	67,08%
EQ, GB, LS	22760	2221	9990	65,08%	67,94%
EQ, GB, LS (indep. faults)	22668	2171	10082	64,91%	67,67%
EQ, GB, LS (same image)	22354	2104	10396	64,14%	66,75%
AVERAGING					
Configuration	TP	FP	FN	ACC	mAP
EQ, GB, GC	22700	2337	10050	64,70%	67,54%
EQ, GB, GC (indep. faults)	21607	2375	11143	61,51%	64,21%
EQ, GB, GC (same image)	19734	2387	13016	56,16%	58,56%
EQ, GC, RV	22690	2704	10060	64,00%	67,39%
EQ, GC, RV (indep. faults)	21620	2702	11130	60,98%	64,16%
EQ, GC, RV (same image)	19707	2678	13043	55,63%	58,37%
MAXIMUM					
Configuration	TP	FP	FN	ACC	mAP
EQ, GB, RS	24495	4609	8255	65,57%	72,53%
EQ, GB, RS (indep. faults)	24499	4845	8251	65,17%	71,18%
EQ, GB, RS (same image)	24442	5085	8308	64,60%	69,49%
EQ, GB, BS	24382	4226	8368	65,94%	72,26%
EQ, GB, BS (indep. faults)	24409	4440	8341	65,63%	70,94%
EQ, GB, BS (same image)	24330	4699	8420	64,97%	69,18%

TABLE XIV: TMR results for the best mAP and ACC configurations of each merging algorithm using the KITTI dataset (320x320 network size).

analysed. (2) Maximum provides higher mAP and ACC than the Baseline for the COCO dataset with both network sizes, and for the KITTI dataset with a 320x320 network size. However, for the KITTI dataset with a 608x608 network size, we observe that the HF, EQ, LS (indep. faults) configuration provides higher mAP but lower ACC (for reasons previously discussed in Section IV-C), and the GB, CR, AR (indep. faults)

BASELINE					
Configuration	TP	FP	FN	ACC	mAP
BL	3865	466	2326	58,06%	60,87%
VOTING					
Configuration	TP	FP	FN	ACC	mAP
HF, LS, TS	3901	403	2290	59,16%	61,60%
HF, LS, TS (indep.faults)	3896	401	2295	59,10%	61,51%
HF, LS, TS (same frame)	3822	398	2369	58,01%	60,36%
RV, LS, TS	3900	416	2291	59,03%	61,54%
RV, LS, TS (indep.faults)	3880	418	2311	58,71%	61,23%
RV, LS, TS (same frame)	3831	410	2360	58,04%	60,48%
AVERAGING					
Configuration	TP	FP	FN	ACC	mAP
BL_1, BL_2, BL_3	3865	466	2326	58,06%	60,87%
BL_1, BL_2, BL_3 (indep.faults)	3697	490	2494	55,34%	58,17%
BL_1, BL_2, BL_3 (same frame)	3392	542	2799	50,38%	53,15%
BL_1, GC, RV	3862	460	2329	58,07%	60,84%
BL_1, GC, RV (indep.faults)	3751	472	2440	56,30%	59,05%
BL_1, GC, RV (same frame)	3449	510	2742	51,47%	54,05%
MAXIMUM					
Configuration	TP	FP	FN	ACC	mAP
HF, RS, BS	4214	691	1977	61,23%	66,01%
HF, RS, BS (indep.faults)	4204	743	1987	60,63%	64,32%
HF, RS, BS (same frame)	4186	980	2005	58,37%	58,75%
HF, LS, TS	4207	713	1984	60,94%	65,87%
HF, LS, TS (indep.faults)	4201	823	1990	59,89%	62,69%
HF, LS, TS (same frame)	4188	1071	2003	57,67%	57,59%

TABLE XV: TMR results for the best mAP and ACC configurations of each merging algorithm using the COCO dataset (608x608 network size).

BASELINE					
Configuration	TP	FP	FN	ACC	mAP
BL	25935	4093	6815	70,39%	76,86%
VOTING					
Configuration	TP	FP	FN	ACC	mAP
HF, EQ, GC	26199	4101	6551	71,09%	77,76%
HF, EQ, GC (indep.faults)	26168	4057	6582	71,10%	77,68%
HF, EQ, GC (same frame)	25910	3957	6840	70,59%	76,92%
EQ, GB, LS	25862	3426	6888	71,49%	77,01%
EQ, GB, LS (indep.faults)	25811	3387	6939	71,43%	76,87%
EQ, GB, LS (same frame)	25502	3293	7248	70,75%	75,95%
AVERAGING					
Configuration	TP	FP	FN	ACC	mAP
HF, EQ, GC	26057	3879	6693	71,14%	77,32%
HF, EQ, GC (indep.faults)	25135	3846	7615	68,68%	74,54%
HF, EQ, GC (same frame)	23178	3776	9572	63,46%	68,70%
HF, EQ, AR	25853	3468	6897	71,38%	76,85%
HF, EQ, AR (indep.faults)	24946	3449	7804	68,91%	74,13%
HF, EQ, AR (same frame)	22901	3425	9849	63,31%	67,99%
MAXIMUM					
Configuration	TP	FP	FN	ACC	mAP
HF, EQ, LS	26974	5877	5776	69,83%	79,80%
HF, EQ, LS (indep.faults)	26956	5988	5794	69,59%	78,79%
HF, EQ, LS (same frame)	26921	6183	5829	69,15%	77,33%
GB, CR, AR	26182	4318	6568	70,63%	77,79%
GB, CR, AR (indep.faults)	26171	4449	6579	70,35%	76,77%
GB, CR, AR (same frame)	26142	4694	6608	69,82%	75,31%

TABLE XVI: TMR results for the best mAP and ACC configurations of each merging algorithm using the KITTI dataset (608x608 network size).

configuration provides higher ACC but lower mAP, since this configuration provides the highest ACC in fault-free cases, but does not deliver as high mAP as the top-mAP configuration. (3) Averaging produces worse results than the baseline with all configurations.

When faults occur in the same image for all three configurations, we obtain the following conclusions: (1) for the

COCO dataset, all merging algorithms provide lower accuracy than the baseline, (2) Voting produces the closest accuracy to the baseline, both in terms of ACC and mAP, (3) for the KITTI dataset, both Voting and Maximum have at least one configuration producing higher accuracy (either both ACC and mAP or only one of the metrics) than the baseline.

Overall, we can conclude the following (1) Averaging is consistently worse than Voting and Maximum in all cases, (2) Maximum produces the highest ACC and mAP results in the fault-free scenarios, as well as when faults occur independently, except for the KITTI dataset with a 608x608 network size, where the ACC of Voting is slightly higher, (3) Voting produces the highest mAP results with the COCO dataset when faults occur simultaneously, but regarding the ACC, two configurations produce higher ACC with Voting, while the other two produce higher ACC with Maximum. (4) When faults occur simultaneously for the KITTI dataset, Maximum provides higher ACC and mAP than Voting with a 320x320 network size, but Voting provides higher ACC with a 608x608 network size.

Overall, we can conclude that the Maximum merging algorithm produces the best mAP results, but it provides significantly higher FPs than Voting. Therefore, if FPs are a major concern, Voting is the best solution overall, while if the objective is to obtain the maximum mAP, the Maximum algorithm is the best solution since the fault-free scenario equals to virtually 100% of time, and typically faults will not affect all redundant components at the same time.

V. CONCLUSIONS

AI software is increasingly used in safety-critical systems for functionalities where such software inherits safety requirements. Whenever those requirements relate to the highest integrity levels, AI software must be realized with diverse redundancy, such as, for instance, TMR. Existing solutions based on lockstep processors provide such diverse redundancy with bit-level error detection and correction. In this paper, we note that abundant AI functionalities do not need bit-level correctness and, instead, semantic correctness suffices due to the stochastic nature of the AI-based functionalities implemented. We leverage this observation to present a diverse redundancy scheme for AI models based on applying minor transformations to the input data with the aim of creating diverse, yet semantically identical, predictions, which we use to mitigate, apart from hardware random errors, also AI model errors.

As future work, we plan to extend our work evaluating the different image transformations and merging algorithms in DMR scenarios, which are particularly relevant for some domains, such as automotive.

ACKNOWLEDGEMENT

The research leading to these results has received funding from the European Union's Horizon Europe Programme under the SAFEXPLAIN Project (www.safexplain.eu), grant agreement num. 101069595. This work has also been partially supported by the Spanish Ministry of Science and Innovation under grant PID2019-107255GB-C21/AEI/10.13039/501100011033. M. Caro has received funding under grant PRE2022-102440 funded by MICIU/AEI/10.13039/501100011033 and, by ESF+.

REFERENCES

- [1] International Standards Organization and SAE International, *ISO/SAE PAS 22736:2021: Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles*, 2021.
- [2] SAFEXPLAIN Consortium, "Space case study," <https://safexplain.eu/space-case-study/>, Accessed in Nov-2023.
- [3] L. Lutnyk et al., "Towards pilot-aware cockpits," in *1st International Workshop on Eye-Tracking in Aviation (ETAVI)*, 2020, pp. 121–127.
- [4] J. Perez-Cerrolaza et al., "Artificial intelligence for safety-critical systems in industrial and transportation domains: A survey," *ACM Comput. Surv.*, oct 2023, just Accepted. [Online]. Available: <https://doi.org/10.1145/3626314>
- [5] M. Caro et al., "At-scale assessment of weight clustering for energy-efficient object detection accelerators," in *Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing*, 2022.
- [6] A. Bochkovskiy et al., "YOLOv4: Optimal speed and accuracy of object detection," 2020. [Online]. Available: <https://arxiv.org/abs/2004.10934>
- [7] Infineon, "AURIX multicore 32-bit microcontroller family to meet safety and powertrain requirements of upcoming vehicle generations," Accessed in Oct-2023, <http://www.infineon.com/cms/en/about-infineon/press/press-releases/2012/INFATV201205-040.html>.
- [8] S. Alcaide et al., "High-integrity GPU designs for critical real-time automotive systems," in *DATE*, 2019.
- [9] —, "Software-only based diverse redundancy for ASIL-D automotive applications on embedded HPC platforms," in *DFT*, 2020.
- [10] L. K. Draghetti et al., "Detecting errors in convolutional neural networks using inter frame spatio-temporal correlation," in *IOLTS*, 2019.
- [11] S. Jha et al., "Exploiting temporal data diversity for detecting safety-critical faults in AV compute systems," in *DSN*, 2022.
- [12] M. Caro et al., "Efficient diverse redundant DNNs for autonomous driving," in *2023 IEEE 47th Annual Computers, Software, and Applications Conference (COMPSAC)*, 2023, pp. 18–27.
- [13] T. Strauss et al., "Ensemble methods as a defense to adversarial perturbations against deep neural networks," *arXiv preprint arXiv:1709.03423*, 2017.
- [14] L. Hansen and P. Salamon, "Neural network ensembles," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, no. 10, pp. 993–1001, 1990.
- [15] Z. Gao et al., "Soft error tolerant convolutional neural networks on fpgas with ensemble learning," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 30, no. 3, pp. 291–302, 2022.
- [16] J. Ayerdi, A. Iriarte, P. Valle, I. Roman, M. Illarramendi, and A. Arrieta, "Metamorphic runtime monitoring of autonomous driving systems," 2023.
- [17] S. Alcaide et al., "Software-only triple diverse redundancy on GPUs for autonomous driving platforms," in *50th Annual IEEE-IFIP International Conference on Dependable Systems and Networks-Supplemental Volume (DSN-S)*, 2020, pp. 82–88.
- [18] "Apollo, an open autonomous driving platform," <http://apollo.auto/>, Accessed in Oct-2023.
- [19] J. Heras, "CLoDSA image augmentation library for object detection," <https://github.com/joheras/CLoDSA>, Accessed in Oct-2023.
- [20] C. Li, "YOLOv4 tensorflow keras implementation," <https://github.com/taipingeric/yolo-v4-tf.keras>, Accessed in Oct-2023.
- [21] T. Lin et al., "Microsoft COCO: Common objects in context," 2015.
- [22] A. Bochkovskiy, "YOLOv4 on the darknet framework and publicly available weights," <https://github.com/AlexeyAB/darknet>, Accessed in Oct-2023.
- [23] Udacity, "Udacity self-driving car driving data," Accessed in Oct-2023. [Online]. Available: <https://github.com/udacity/self-driving-car>
- [24] F. Yu et al., "BDD100K: A diverse driving video database with scalable annotation tooling," *CoRR*, vol. abs/1805.04687, 2018. [Online]. Available: <http://arxiv.org/abs/1805.04687>
- [25] P. Sun et al., "Scalability in perception for autonomous driving: Waymo open dataset," *CoRR*, vol. abs/1912.04838, 2019. [Online]. Available: <http://arxiv.org/abs/1912.04838>
- [26] M. Everingham et al., "The pascal visual object classes (VOC) challenge," *International Journal of Computer Vision*, vol. 88, no. 2, 2010.
- [27] R. Padilla et al., "A survey on performance metrics for object-detection algorithms," in *2020 International Conference on Systems, Signals and Image Processing (IWSSIP)*, 2020, pp. 237–242.
- [28] G. Li et al., "Understanding error propagation in deep learning neural network (DNN) accelerators and applications," in *SC17: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2017, pp. 1–12.
- [29] A. Bosio et al., "A reliability analysis of a deep neural network," in *2019 IEEE Latin American Test Symposium (LATS)*, 2019, pp. 1–6.

Formal description of ML models for unambiguous implementation

Adrien Gauffriau *Airbus, France* Iryna De Albuquerque Silva *ONERA, France* Claire Pagetti *ONERA, France*

I. INTRODUCTION

Machine learning (ML) applications have been gaining considerable attention in the field of transportation. However, their use in real-life operational safety-critical products, in particular in the aeronautical domain subject to stringent certification, raises several issues regarding functional correctness, compliance with requirements, formal verification, safety or implementation. In order to tackle those issues, new guidelines – named ED 324/ ARP 6983 standard [EUR21] – are currently drafted by the EUROCAE WG-114/SAE G-34 joint working group that cover the whole spectrum of the system development including the data sets composition, the ML model design and its implementation. In this paper, we focus only on the implementation of the ML model.

A. Context

In the ML current practices, a *training framework* is used to design an ML model and the resulting ML model is then deployed on the target with a *deployment framework*. It is up to the training framework or a designer to export the trained model description in an exchange format and up to the deployment framework to import the ML model description. The left part of figure 1 shows those practices. The deployment framework is most of the time an ML model interpreter, that can accommodate any type of ML model architecture, and that allocates at runtime the execution on the different available accelerators (e.g. GPU, FPGA) of the target. These ML frameworks have been designed 1) to ease as much as possible the deployment of models for the users and 2) to optimize as much as possible the execution performance (usually expressed in trillion operations per second – TeraOp/s). As a result, they are very impressive and allow for complex deployments and optimizations. Those are hard, if not impossible, to reproduce for a programmer without using ML libraries (e.g. CUDNN on NVIDIA).

The counterpart of such an approach is 1) the absence or limited information of internal computation and allocation; 2) small modifications and adaptations of the ML model (e.g. when exporting the ML model description or when quantizing on the fly some matrix multiplication to execute on deep learning accelerator – DLA). If this grey/black box approach is acceptable and suitable for mainstream applications, it is a blocking point for highly safety-critical applications. As a result, the main objective of the ARP 6983 standard with respect to the implementation process is the *semantics preservation* of the off-line trained model on the final hardware platform. This means that the execution of the ML model in

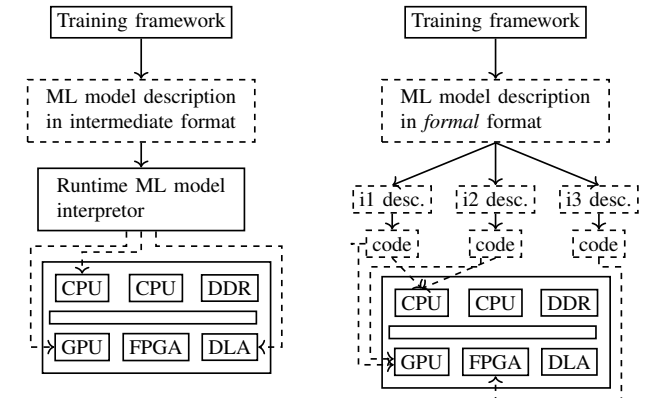


Figure 1. Left part: current ML deployment practice and right part: proposed aeronautical practice.

the training framework should be exactly reproduced on the embedded target during execution. To reach that objective, an alternative development process is proposed as illustrated in the right hand part of figure 1.

B. Assurance development process

The principle of this trustable development process (figure 1) is to ensure the semantic preservation at each step of the development cycle (e.g. between training framework and description, between description and code, between code and executable). This is guaranteed thanks to a series of requirements provided in the ARP 6983.

Requirement 1. First, the trained model must be formally and unambiguously defined in what we call an adequate *format*. Such a *format* must come with a formal syntax and semantic, and should be agnostic of any (training and/or deployment) framework.

Requirement 2. Second, the implementation process should allow several deployments on hardware platforms and it must be known beforehand how the ML model will be mapped on the accelerators and when. This entails in particular that the *format* should allow several types of deployment such as distribution, parallelization or pipelining. Thus, there is a step that consists in splitting the ML model description (in the chosen format) as a series of *item* descriptions. Indeed, in the avionics domain, a target processor is decomposed as a set of software (SW) and/or hardware (HW) *items*. Let us consider for instance an ULTRASCALE+ (ZCU102) platform [Xil19]: it is composed of several ARM cores, a GPU and an FPGA. If the ML model is spread over the different components, in

particular on one ARM and the FPGA, there will necessarily be several items. Indeed, the ARM associated code will be considered as one SW item and will go through the ED-12C/DO-178C [RTC11] development process. Whereas the hardware design of the FPGA will be considered as another item (HW) and will go through the ED-80/DO-254 [RTC00] development process.

Requirement 3. Third, the implementation has to follow the usual aeronautical development standards (e.g. ED-12C/DO-178C [RTC11] for software). Thus, the description of a model within the format must be compliant with a certified implementation process. This last objective concerns the capability to implement an item description following standards such as ED-12C/DO-178C [RTC11] or ED-80/DO-254 [RTC00]. Among the requirements from those standards, two are related to the format. First, it must offer full traceability: looking at the generated code (e.g. C or CUDA), it is humanly possible to trace back to the original exported model. Second, the execution must be *predictable*, meaning that it is possible to assess a WCET (Worst Case Execution Time) [WEE⁺08]. This entails that the code is expected to be allocated statically on the resources, all the memory allocations are static and the schedule (here the sequence of operations) is also static.

C. Contributions

Our general objective is to define an approach compatible with the ARP 6983 requirements presented in the prior section. We focus on a representative subset of deep neural networks (DNNs) that is feed-forward neural networks trained off-line. Our main goal is the definition of an adequate format, with a formal syntax and semantics, able to describe both 1) a global ML model, and 2) any parallelized allocation on several items, the behaviour of which is equivalent to the global model.

For requirement 1. There are several initiatives to propose an intermediate format between trained models and their implementation such as ONNX [BLZ⁺19] (Open Neural Network Exchange). After a thorough evaluation of different existing formats, we identified NNEF (Neural Network Exchange Format) [The22] as the most suited for our purpose: syntax and semantics are public and moreover the authors made a strong effort to provide a formal specification.

Since NNEF provided a potential candidate, we decided to construct our format on top of it. As it is now, NNEF describes formally the global ML model. Indeed, the semantics of NNEF is almost fully defined (see section II). What is however missing in NNEF is the clear formalization of the *execution model*, that is the formal behaviour behind a series of NNEF *instructions*. To fix this missing element, we rely on Petri nets, a usual representation of program behaviours [Pet77]. This choice is also consistent with the need to express distributed behaviour on items, as Petri nets also allow to model all combinations of execution: sequence, pipeline, recursion or parallel [Old86].

For requirement 2. We illustrate in section III why decomposing ML model into items is of interest. To that end, we extend the syntax and semantics of NNEF. We rely in particular on *logical data exchange* among items to distribute the computation and express the semantics with *coloured Petri*

nets. We formally show that the synchronization of the Petri nets behaves as the *original* non distributed Petri net.

For requirement 3. We do not propose a complete DO-178C compatible approach but instead show a reasonable implementation approach on the XAVIER platform [NV119a] that we believe could be with some effort compliant with the ED-12C/DO-178C [RTC11].

II. FORMAT OF DESCRIPTION – NNEF

KHRONOS standardization group¹ has defined the NNEF (Neural Network Exchange format) format with a specification that provides a syntax and a semantic. We focus on the NNEF syntax and semantics elements needed for our purpose. The reader can refer to [The22] for a complete description of NNEF.

A. Brief Reminder on Neural Network

The field of artificial intelligence has gained much research attention in the past years. The power of AI resides in the capacity of solving highly complex problems [GBC16]. Machine learning domain describes the study and development of statistical algorithms that are able to efficiently generalize on unknown input data after the extraction of patterns from a similar, and representative, data set. Neural networks are a class of ML algorithms. A neural network implements a mathematical function $F_{\mathcal{N}}$ that aims at approximating a continuous real-valued function [HSW89], [SZ06]. $F_{\mathcal{N}}$ is composed of different mathematical functions called *layers*.

There are two types of deep neural networks: feed-forward neural networks and recurrent neural networks. Recurrent neural networks (RNN) feature layers that take as input some of their output (or the output of a successor layer), thus creating cycles. In feed-forward variants it is not true. We are not addressing RNN in this paper. A common representation of a feed-forward deep neural network (FDNN) is in the form of a *directed acyclic graph* (DAG) defining how its layers are connected together.

Definition 1 (Feed-forward Deep Neural Network): A feed-forward deep neural network $\mathcal{N} = (V, E)$ is a *directed acyclic graph*, wherein:

- V is the finite set of vertices of the graph, which represent its layers $l \in V$;
- $E \subseteq V \times V$ is the set of edges of the graph, representing the data flow within the neural network.

In order to construct the possible flows of data within the feed-forward deep neural network, it is necessary to define what are the predecessors and successors of a vertex, or layer.

Definition 2 (Predecessors / successors of a layer): The direct predecessors (resp. successors) of a layer l are defined as the layers of the set $Pre(l) = \{l' \in V \mid (l', l) \in E\}$ (resp. $Succ(l) = \{l' \in V \mid (l, l') \in E\}$). The predecessor transitive closure of l is the set of all its predecessors layers noted $Pre^*(l) = \bigcup_{n=1}^k Pre^n(l)$, wherein:

$$Pre^n(l) = \begin{cases} Pre(l), & \text{if } n = 1 \\ \bigcup_{l' \in Pre^{n-1}(l)} (Pre(l') \cup \{l'\}), & \text{otherwise} \end{cases}$$

¹<https://www.khronos.org/>

A layer can be classified into input, output and intermediate, or hidden. An input layer l only consumes input data, i.e., $Pre(l) = \emptyset$. Similarly, a final layer l only produces output data, i.e., $Succ(l) = \emptyset$. We represent $V_I \subseteq V$ as the set of input layers and $V_O \subseteq V$ as the set of output layers. Note that $V_I \cap V_O = \emptyset$. The remaining layers, $l \notin V_I \cup V_O$, are the hidden layers. Finally, as a straight consequence of directed acyclic graphs properties, $l \notin Pre^*(l)$.

The function performed by a layer is of the form $f_l : \mathbb{R}^m \rightarrow \mathbb{R}^n$, wherein m and n represent respectively the input and output dimensions of the given layer's function.

Definition 3 (Function associated to a FDNN): Let $\mathcal{N} = (V, E)$ be a feed-forward deep neural network and $V_O \subseteq V$ be the set of output layers. Let us denote the function associated to a set of layers such that:

$$\forall U \subseteq V, F_U = \begin{cases} (F_{l_1}, \dots, F_{l_n}), & \text{if } U = \{l_1, \dots, l_n\} \\ F_\emptyset, & \text{if } U = \emptyset. \end{cases} \quad (1)$$

wherein:

$$\forall l \in V, F_l = f_l(F_{Pre(l)}) \quad (2)$$

We note $F_{\mathcal{N}} = F_{V_O}$ the function associated to a feed-forward deep neural network.

Example 1 (Single-path feed-forward deep neural network): It corresponds to the particular case of a feed-forward deep neural network, wherein: $V = \{l_1, \dots, l_n\}$ and $\forall i \geq 2$, $Pre(l_i) = \{l_{i-1}\}$. Therefore, $V_I = \{l_1\}$, $V_F = \{l_n\}$. Such an example is the LUNET-5 shown in figure 2.

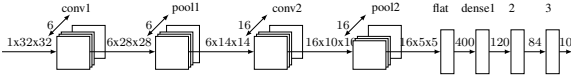


Figure 2. LUNET-5 neural network

According to Definition 3 the function associated to a single-path DNN is the composition function: $F_{\mathcal{N}}(x) = \circ f_{l_n} \circ \dots \circ f_{l_1}(x)$ wherein m_{l_1} in the input dimension of f_{l_1} and p_{l_n} is the output dimension of f_{l_n} .

B. Neural Network Description in NNEF

Our first goal concerns the definition of a format, with a formal syntax and semantics, able to describe any ML model such as the LUNET-5 of example 1. Let us explain why NNEF fulfils this goal. An NNEF description is composed of two parts: 1) A computation graph described in a human readable text file; and 2) the parameters provided in multiple raw data file. The fact that the description is textual is important for the traceability between the specification (output of the training framework) and the embedded code.

The computation graph file describes all parameters needed and operations to be done. More precisely, each line of the computation graph is an elementary instruction (NNEF compound fragment) that may be split into several atomic operations (NNEF primitives). The result of each instruction is stored in a named variable, that represents a tensor, and which can be used as input for other instruction(s). To compute an operation all its inputs variables shall be available.

Remark 1: NNEF description follows a SSA (static single assignment) form [CFR⁺89] which helps the implementation process. It is usual to translate a program in its SSA form before compilation or optimization passes [BBD⁺17].

Example 2: Let us illustrate how to specify a neural network with an example. The NNEF textual specification of the LUNET-5 detailed in example 1 is given in the listing 1. First, all parameters are declared and stored as variables. $e1$ is the input tensor of size $1 \times 32 \times 32$, $v1$ contains the 6 kernels of size $1 \times 5 \times 5$ of the first convolution and $v2$ is the bias applied at the end of the first convolution. The parameters needed by the layers should all be defined as variables in the description file.

```
graph torch_jit_export(e1) -> (out)
{
  e1 = external<scalar>(shape = [1, 1, 32, 32]);
  v1 = variable<scalar>(shape = [6, 1, 5, 5],
    label = 'v1');
  v2 = variable<scalar>(shape = [1, 6], label = 'v2');
  v3 = variable<scalar>(shape = [16, 6, 5, 5],
    label = 'v3');
  v4 = ...; v5 = ...; v6 = ...; v7 = ...;
  v8 = ...; v9 = ...; v10 = ...;

  o1 = conv(e1, v1, v2, stride = [1, 1], dilation
    = [1, 1], padding = [(0, 0), (0, 0)], groups
    = 1);
  o2 = relu(o1);
  o3 = max_pool(o2, size = [1, 1, 2, 2], stride =
    [1, 1, 2, 2], dilation = [1, 1, 1, 1],
    padding = [(0, 0), (0, 0), (0, 0), (0, 0)],
    border = 'ignore');
  o4 = conv(o3, v3, v4, ...; o5 = relu(o4);
  o6 = max_pool(o5, ...; o7 = reshape(o6, shape =
    [0, -1]);
  o8 = linear(o7, v5, v6); o9 = relu(o8);
  o10 = linear(o9, v7, v8); o11 = relu(o10);
  o12 = linear(o11, v9, v10);
  out = softmax(o12, axes = [1]);
}
```

Listing 1. LUNET-5 with NNEF syntax

After the variables declaration, comes the computation graph itself. The output of the first convolution is stored in the variable $o1$. When calling the function / compound fragment `conv`, the user must instantiate the full set of parameters for this type of layer: input tensor, kernels, bias, stride, dilation, padding and groups. Every parameter appears explicitly in the definition and there is no ambiguity. For instance, the way to declare the padding explicitly states how the padding applies on top / bottom / right / left. After the convolution, the activation function has to be explicitly applied to $o1$ and thus is not hidden in the convolution, ensuring again an unambiguous description. The first pooling layer results in variable $o3$. Reading the description, we recognize the LUNET-5 detailed before. The flat layer is encoded with a more expressive function `reshape` that allows several reshaping. The dense layer is called `linear`. The post-processing softmax is also explicit.

The NNEF specification also provides the link between instructions (e.g. `conv`) appearing in the file and their associated mathematical functions. Subsequently, we will not use NNEF terms, because the NNEF standard uses the generic term `fragment` for both. We illustrate this with the max pooling

layer only.

C. Max Pooling Layer Semantics

Let us illustrate issues that may arise without a formal description. Let us first remind the functional semantics of a max pooling layer where a padding (and no dilatation) is applied. Thus, the function is defined by $\mathcal{P}ool_{k,s} \circ \mathcal{P}_{p,v}$ where each function is defined below.

Definition 4 (Padding associated function – $\mathcal{P}_{p,v}$): Let $p = (p_t, p_b, p_l, p_r)$ be a 4-tuple of integers representing the padding to be applied on each border of a 3D-tensor and v the float value to be used for the padding. The padding function $\mathcal{P}_{p,v}$ applied on a 3D-tensor I of size (n_h, n_w, n_c) outputs a 3D-tensor $O = \mathcal{P}_{p,v}(I)$ of size (o_h, o_w, o_c) with $o_h = n_h + p_t + p_b$, $o_w = n_w + p_l + p_r$ and $o_c = n_c$ such that

$$O_{x,y,z} = \begin{cases} v & \text{if } (x \leq p_t) \text{ or } (x > n_h + p_t) \text{ or} \\ & (y \leq p_l) \text{ or } (y > n_w + p_l) \\ I_{x-p_t, y-p_l, z} & \text{otherwise} \end{cases}$$

Definition 5 (Pooling layer associated function – $\mathcal{P}ool_{k,s}$): Let $s = (s_h, s_w)$ be the stride parameters and $k = (k_h, k_w)$ be the height and width of the window. The pooling applied on a 3D-tensor I of size (n_h, n_w, n_c) outputs a 3D-tensor $O = \mathcal{P}ool_{k,s}(I)$ of size (o_h, o_w, o_c) with $o_h = \lfloor \frac{n_h - k_h}{s_h} + 1 \rfloor$, $o_w = \lfloor \frac{n_w - k_w}{s_w} + 1 \rfloor$ and $o_c = n_c$ with $O_{x,y,z} = \max(I[s_h \cdot (x-1) + 1 : s_h \cdot (x-1) + k_h + 1][s_w \cdot (y-1) + 1 : s_w \cdot (y-1) + k_w + 1][z])$. Here, $I[s_{11} : s_{21}, \dots, s_{1k} : s_{2k}]$ represents the slice of I of all the values $I_{s_{11}+x_1, \dots, s_{1k}+x_k}$ with $i \in [1, k]$ and $x_i \in [1, s_{2i} - s_{1i}]$.

The NNEF syntax of a max pooling layer describes the padding values with an enumerate string. Ignoring the border for a max pooling layer is equivalent to pad with the minimum float value (MIN_F). This corresponds to the neutral operator of the max function. Looking now at the `max_pool` elementary instruction according to NNEF documentation [The22], it is defined in the pseudo-code by 2 atomic operations. 1) `argmax_pool` that computes an array of index (corresponding to the max in each pool); 2) `sample` that returns for an array of index, an array with corresponding values. This pseudo-code indeed encodes the expected function $\mathcal{P}ool_{k,s} \circ \mathcal{P}_{p,v}$.

Of the importance of unambiguous description. We propose to highlight the importance of making unambiguous textual descriptions of ML models by comparing two state-of-the-art training frameworks, namely PYTORCH and KERAS (with TENSORFLOW).

- In KERAS, a padding inside a max pooling layer can only be declared by a string in $\{\text{"valid"}, \text{"same"}\}$. "valid" means no elements to add, while "same" means that padding is added on the right and on the bottom borders only to fit the size of the pool.

```
# KERAS SYNTAX for pool1
MaxPool = tf.keras.layers.MaxPooling2D(
    pool_size=(2,2), strides=(2,2), padding
    = 'same')

# NNEF for KERAS SYNTAX
max_pool_keras = max_pool(input, size =
    [1, 1, 2, 2], stride = [1, 1, 2, 2],
    dilation = [1, 1, 1, 1], padding =
```

```
[(0, 0), (0, 0), (0, 1), (0, 1)],
border = 'ignore');
```

This corresponds to $\mathcal{P}ool_{[2,2],[2,2]} \circ \mathcal{P}_{[0,1,0,1],MIN_F}$.

- In PYTORCH, a padding inside a max pooling layer can only be declared by one or two integers. In case of one integer, this defines the number of elements to add to each border (top, bottom, left and right). In presence of 2 integers, the first gives the number of elements to add at the top and bottom, and the second at the left and right.

```
#PYTORCH SYNTAX for pool1
MaxPool = NN.MaxPool2d(2, 2,
    1) # Kernel Size, Stride,
    Padding

#NNEF for PYTORCH SYNTAX
max_pool_torch = max_pool(input, size = [1,
    1, 2, 2], stride = [1, 1, 2, 2],
    dilation = [1, 1, 1, 1], padding = [(0,
    0), (0, 0), (1, 1), (1, 1)], border =
    'ignore');
```

This corresponds to $\mathcal{P}ool_{[2,2],[2,2]} \circ \mathcal{P}_{[1,1,1,1],MIN_F}$

The semantics of KERAS and PYTORCH are not equivalent, and there is no possibility to convert one into another at once. The only way to make a valid conversion is to explicitly add a padding layer before the max pooling.

D. NNEF Execution Model Semantics

The semantics of the *execution model*, that is the formal behaviour behind a series of NNEF instructions, is not explicitly given by the standard. It assumes that one instruction can be executed when all its inputs are computed and available. Thus, executing the instructions in sequence following the order of the NNEF guarantees a correct execution. There are two types of instructions: those reading parameters from binary files or input tensors and layer-associated instructions based on one or several atomic operations. An instruction is always of the form

$$var = operation(v_1, \dots, v_k, cst_1, \dots, cst_j);$$

where var is a variable computed by the *operation* (any NNEF fragment), v_i are either variables computed beforehand, the input tensor or fixed parameters (e.g. kernels), and cst_i are constant (e.g. stride). The NNEF execution model can be formally expressed using the Petri net formalism [Pet77].

Translation 1: A NNEF description, composed of n instructions, generates a Petri net (P, T, V) where:

- the set of places P corresponds to all variables appearing in the NNEF description (i.e. n places for a description of n instructions);
 - a token in a place means that the associated variable is available for computation;
 - initially there are as many tokens in each parameter-associated place as the parameter is needed in the instructions and as many tokens in the input tensor place as the input tensor is used by the instructions;
 - there is a unique final place corresponding to the last variable computed in the NNEF file;

- the set of transition names T corresponds to all instructions appearing in the NNEF description;
- $V \subseteq 2^P \times T \times \mathbb{N} \times P$ defines the set of transitions. A transition can be fired if there is a token in all input places. When it fires, the transition removes a token from each of these places and generates as many tokens as defined on the edge in the unique output place.
 - each instruction $var = op(v_1, \dots, v_k, cst_1, \dots, cst_j)$ generates the transition given in figure 3 where p is the number of time var will be consumed by other instructions;
 - when only one token is produced by a transition, we omit the integer.

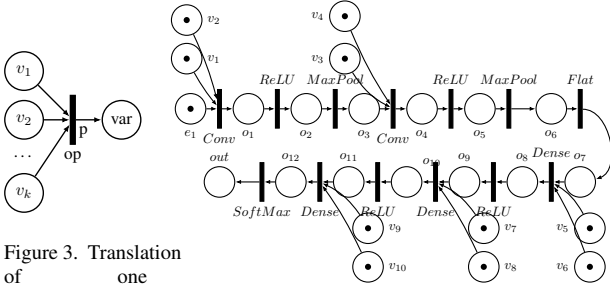


Figure 3. Translation of one instruction

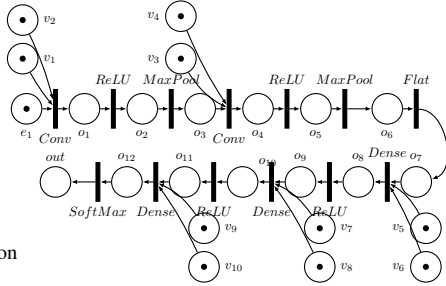


Figure 4. NNEF semantics of the LUNET-5 express with Petri net. Initial marking

The semantic of the Petri net clarifies the execution order that is unclear in the NNEF formalism. Especially, the order of the textual file should not impose a unique order when several valid ones may exist.

Example 3: The LUNET-5 model of figure 2 with its associated NNEF description in listing 1 has the associated Petri net given in figure 4. We recognize the instructions sequence that describes the computation of the neural network graph. There is a unique possible schedule for this NNEF description, but we will see later other NNEF models that accept several schedules (see section III).

Remark 2: The Petri net of figure 4 only defines the semantics of a single inference pass. It is usual to repeat the inference pass in order to process new inputs (e.g. in a periodic manner). This can also be represented using the Petri formalism by sending back tokens to $e1$ and v_k places. For demonstration and clarity, subsequently in the paper, we always consider the semantics of a single inference.

A way to define the semantics of Petri nets is to compute the set of reachable *markings*, where a marking defines the number of tokens in each place at a given instant.

Definition 6 (Marking): Considering a Petri net with n places, a marking is defined as a vector $v \in \mathbb{N}^n$ giving the number of tokens $v[i]$ in the i -th place. This initial location of tokens is called the initial marking, this represents the starting state of the system. A final marking is a marking such that there is one token only in each final place and from which no transition can be fired any more.

Example 4: In the Petri net of figure 4, there are 24 states. There are 11 tokens in the initial making and there is a single token in the unique final marking (in the place out).

Property 1 (Initial marking and unique final marking): The unique initial marking is defined by the translation and consists of token(s) in the input tensor variable place and parameters-associated places. Because we consider feed-forward neural networks, there is unique *final marking*.

Definition 7 (Paths and semantics): A valid path starts from the initial marking m_i , lists a series of fireable transitions and ends in a final marking m_f , i.e. $m_i \xrightarrow{t_1} m_1 \xrightarrow{t_2} \dots \xrightarrow{t_l} m_f$. The semantics of a Petri net is the set of valid paths.

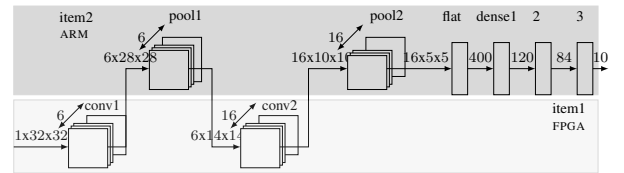
Property 2 (Possible executions of an NNEF description): Because we consider feed forward neural networks, the number of valid paths is finite and the valid paths correspond to all possible execution orders respecting the semantics of the ML model.

Semantics preserving code generation could lead to any implementation the path of which is valid. Full sequential code following the order of instructions of the NNEF file is one of them.

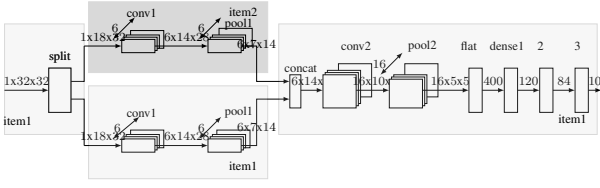
III. IS DISTRIBUTION NEEDED?

Because we consider highly distributed platforms, a designer may choose to split the ML model into several parts in order to accelerate the execution and reduce the execution time. In particular, it could lead to developing parts independently and on different *items* following the aeronautics standards [SAE10]. In such a case, there should be a formal description for each item that becomes the input specification for HW/SW item implementation. We identified 3 different needs for distribution to be addressed that we illustrate on the LUNET-5 example.

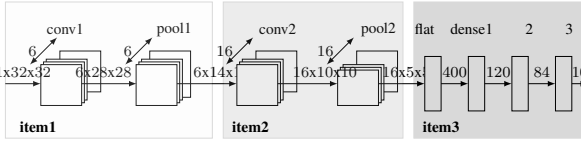
Remark 3: Note that if the designer considers its platform as a unique item, the NNEF description will be the specification. **Off-loading computation.** Let us consider for instance the implementation of the LUNET-5 on an ULTRASCALE+ (ZCU102) platform [Xil19] composed of several ARM cores, a GPU and an FPGA. Let us assume that we choose to execute the convolution on the FPGA and all other layers on one ARM. The idea will be to offload the input tensor of each convolution on the FPGA and retrieve the feature maps from the FPGA (see figure below).



Parallelizing the layers. A second type of distribution could be to refine the layers and exhibit more parallelism by distributing the computation of a layer across several items. It will be up to the designer to show the semantic preservation at this refinement level. Looking again at the LUNET-5, we can split the computation of the first two layers on two different items. Each item will do the convolution+maxpool on a part of the input image. To do so, the input image is split along the height and two sub-parts are executed on two different



items (see figure below). In order to keep the semantics for the convolution, some overlap exists between the two sub-images. **Pipelining the computation.** A third type of distribution is the pipelining of the DNN execution. In this case, each item is in charge of computing a specific layer (or a group of layers). The first item computes the first layer(s) on the first tensor input and sends its output to the second item that will compute the second group of layer(s) while the first item starts processing the second input tensor. This classical mechanism enables to reduce (e.g. for video processing) the frame rate while degrading the latency. The depth of the pipeline is the number of inputs that can be handled at the same time among the pipeline.



IV. NNEF EXTENSION FOR MULTIPLE ITEMS

The purpose of this section is to propose a manner to separate the specification of each item so that any execution of the items respecting the specification properly encode the global ML model. To that extent, we propose first to extend the syntax of NNEF to allow explicit parallelization and then to express the associated semantics with colored Petri nets.

A. Extension for item splitting

We first need to specify the item on which the description will be implemented. To that end, we enrich the graph definition with the keyword *graphitem* to provide the logical id of the HW or SW item.

Syntax1 GraphItem

```
<graph-definition> ::= <graph-declaration>
  <graph-declaration-item> <body>

<graph-declaration-item> ::= "graphitem"
  <identifier> <identifier> "(" (<identifier-list>)" "
  "->" "(" (<identifier-list>)" "
```

Semantics 1: The first *<identifier>* refers to the item id, the second *<identifier>* is the name of the local node and elements of the *<identifier-list>* will be input or output of the *graphitem*. The semantic of the *<graph-declaration-item>* is such that all instructions within the *<body>* are executed by the item.

We also need to *exchange data* between several items and ensure that those exchanged data are available before computation. To that end, we introduce a new type of variable,

namely *variablesync*. This references a variable that could be read from or write to another *graphitem*. We use a *fragment* to define this new type.

Syntax2 VariableSync

```
fragment variablesync<? = scalar>
  (shape: integer[] -> ( output: tensor<?> )
```

Semantics 2: Each *variablesync* is a shared variable with a unique writer and possibly multiple readers. Writer is in charge to transmit the variable via the instruction *send_var* and each reader can access this data via *get_var* instruction.

We then define new NNEF instructions to send or get data between several *graphitem*.

Syntax3 get_var

```
fragment get_var<? = scalar>(source : graphitem,
  data : variablesync) -> ( output: tensor<?> )
```

Semantics 3: *get_var* appears in each reader description. The output of this instruction is a local variable that gets the content of the shared variable and which is available for the caller item. *Source* is the item that writes and provides the shared variable the name of which is given by *data*.

Similarly, the writer must define the instruction to send a shared variable to other items.

Syntax4 send_var

```
fragment send_var<? = scalar>
  ( dest : graphitem[], data : scalar)
  -> ( output: variablesync )
```

Semantics 4: *send_var* appears in the writer description only. It takes as input the list of reader items and the name of the tensor that shall be sent. The output tensor is a global *variablesync* that will support the *synchronization*.

The rest of the NNEF syntax remains unchanged.

B. Splitting an NNEF description into multi-item descriptions

Initially, the DNN is described in a unique NNEF description as shown in section II. Such a description contains 3 types of instructions:

- definition of input tensor(s);
- definition of fixed parameters;
- variables computed by each layer.

A splitting consists in partitioning the last type of instructions among the items, adding the adequate definition(s) of tensors / fixed parameters and adding the adequate *send_var* / *get_var*. The composition of item descriptions shall respect the semantics of the full NNEF description.

Example 5: Let us consider the DNN of listing 2 with its associated Petri net (figure 5). Let us assume that the DNN is allocated on 3 items such that o_1, o_6, o_7 and *out* are computed on item 1; o_2, o_3 are computed on item 2 and o_4, o_5 are computed on item 3. Thus, the description on the items is given in Listing 3.

The union of the instructions of each item corresponds to the complete NNEF description with the additional *variablesync* and the communication instructions. Locally in the item, the

```

graph DNN(e1) -> (out)
{
  e1 = external...
  v1 = variable... 'variable1'; v2 = ...; v3 =
  ...; v4 = ...;
  v5 = ...; v6 = ...; v7 = ...; v8 = ...; v9 = ...;
  v10 = ...;
  o1 = conv(e1, v1, v2, ...;
  o2 = conv(o1, v3, v4, ...; o3 = conv(o2, v5, v6, ...;
  o4 = conv(o1, v7, v8, ...; o5 = conv(o4, v5, v6, ...;
  o6 = concat(o3, o5); o7 = flatten(o6, ...;
  out = gemm(o7, v9, v10...;
}

```

Listing 2. Complete DNN NNEF

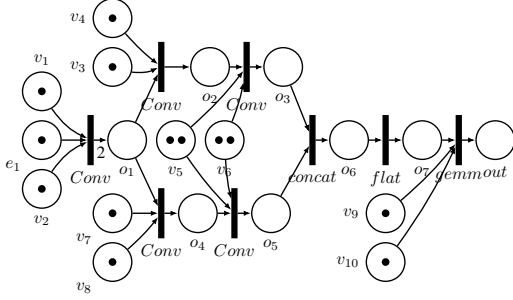


Figure 5. Petri associated to DNN of listing 2. Initial marking

pointers to the input tensor and fixed parameters must also be declared.

```

graphitem ITEM1 DNN1([e1, vsync2, vsync3]) -> ([
  vsync1, out])
{
  e1 = external...;
  v1 = variable... 'variable1'; v2 = ...; v9 =
  ...; v10 = ...;
  vsync1 = variablesync<scalar> (shape ...;
  vsync2 = variablesync ...; vsync3 =
  variablesync ...;
  o1 = conv(e1, v1, v2, ...;
  vsync1 = send_var ([ITEM2, ITEM3], o1);
  o3 = get_var (ITEM2, vsync2);
  o4 = get_var (ITEM3, vsync3);
  o6 = concat(o3, o4); o7 = flatten(o4, ...;
  out = gemm(o7, v9, v10...;
}

graphitem ITEM2 DNN2([vsync1]) -> ([vsync2])
{
  v3 = variable... 'variable3'; v4 = ...; v5 =
  ...; v6 = ...;
  vsync1 = variablesync ...; vsync2 = variablesync
  ...;
  o1 = get_var (ITEM1, vsync1);
  o2 = conv(e1, v3, v4, ...; o3 = conv(o2, v5, v6, ...
  vsync2 = send_var ([ITEM1], o3);
}

graphitem ITEM3 DNN3([vsync1]) -> ([vsync3])
{
  v5 = ...; v6 = ...; v7 = ...; v8 = ...
  vsync1 = variablesync...; vsync3 = variablesync
  ...;
  o1 = get_var (ITEM1, vsync1);
  o4 = conv(o1, v7, v8, ...; o5 = conv(o1, v5, v6, ...
  vsync3 = send_var ([ITEM1], o5);
}

```

Listing 3. NNEF for all items

C. Petri-based semantic

We define the execution model semantics of multi-item descriptions using coloured Petri nets [JK09]. We associate a colour to each item where the colour is set to the tokens and edges (on which the coloured tokens transit).

Translation 2: Let assume there are N items. We first apply the translation 1 for each item leading to N independent Petri

nets (P_i, T_i, V_i) , each with a unique and distinct colour. For the new instruction, the translation is extended as follows:

- a *varsync* does not generate any place;
- a *get_var* does not generate any transition;
- a *send_var* produces a transition *sync* with an incoming edge from the variable the content of which is transmitted.

The set of NNEF item descriptions generates a Petri net (P, T, V) which is roughly speaking the union of the N Petri nets (P_i, T_i, V_i) . More precisely,

- any input tensor or fixed parameter that is duplicated in the NNEF files appears in the Petri net of the item. Those duplicated places are merged. Because we use the same naming convention, $P = \cup P_i$;
- the initial tokens in the places are also merged leading to places with possibly multiple tokens and multiple colours;
- $T = \cup T_i \cup T'$ where T' are the transitions connecting places of one item to other items thanks to the *sync* transition. More precisely,
 - for each writer, there are k edges back from the *sync* label where k is the number of readers. The colour of the each arrow is the one of the reader and the number of tokens sent back corresponds to the number of time the shared variable appears in the reader item instructions;
 - for each reader, there is an edge from the writer place before *sync* to each transition requiring the shared data;
- As before, when a coloured token is present in a place, it means that the associated variable is available for the item identified by the colour and can be used by transition.

Example 6: The Petri net associated to the example 5 is given in figure 6. We present the initial marking with colored tokens. Each color represents the state of one item. Compare to the figure 4, we express here the multi items semantics with synchronizations.

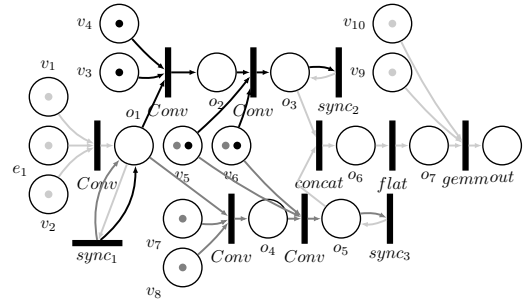


Figure 6. Semantic of the items synchronization

Property 3 (Equivalence between Petri nets): The semantics of the multi-items behaviour is equivalent to the complete original ML model.

Remark 4: It is important to explicitly describe the *send_var* and *get_var* either in the NNEF files but also in the Petri net because items are supposed to be independent and segregated. In particular, an item X is not allowed to access the memory space of an item Y and interfere with its execution. This is classical in aeronautics, see Arinc 653 specification. The XTRATUM hypervisor [CRM⁺09] is an example of *time*

and space partitioning hypervisor that provides communication with *sampling* and *queuing* ports (close to Arinc 653 requirements).

V. IMPLEMENTATION AND EXPERIMENTS

The previous sections showed how to fulfill the requirements 1 and 2 of the introduction. The purpose of this section is to give some hint of how a DO-178C compatible implementation process, as required per requirement 3, could be defined taking as input an extended NNEF specification. The considered target, a Jetson XAVIER TX system-on-chip, is composed of 6 Carmel ARM cores, a GPU, 2 deep learning accelerators (NVDLA) and other dedicated circuits. The use of a NVIDIA platform is mainly motivated by its availability and the ease to quickly deploy neural networks application. We will not discuss the adequation of GPU and CUDA implementation with DO-178C objectives because it is an open problem.

The *sync* implementation relies on barrier mechanism and each item NNEF description is manually coded. In order to validate the semantic preservation, we made some instrumentation to show that: (i) the execution trace is included in all possible execution traces defined by the Petri net; (ii) the numerical precision is kept; (iii) the measured execution time does not vary. We will use the multi-items example presented in example 5 as the specification. For our experiments, each item is allocated to one CPU and all the CUDA cores of the GPU (grouped in a CUDA stream). As a consequence, we do not guarantee a segregation between items (as they share the GPU) we instead focus on a way to implement parallel operations of neural networks with a static code scheduling while preserving the semantics. As for the Petri semantics, we only developed a code for a single inference. Nevertheless, it is easy to slightly modify the code by adding loops to handle several input tensors.

A. Get/Send specification

We chose to implement *get_var* and *send_var* with 1) global variables stored in the SRAM of the XAVIER and 2) the POSIX barrier mechanism of the *pthread* library. A barrier *b*, shared among several processes, will block them as long as not all of them reach *b*. Such a behavior is strictly included in the semantics of the *sync* transition within the Petri net. However, it is not the most efficient as it prevents the sending item to proceed until all the receiving items reach *b*, whereas the semantics of *sync* transition only requires a receiver to wait for the sender (not the sender to wait for all receivers). Nonetheless, the barrier mechanism is optimal for our example because no sender has to process any instruction before a further *get_var* or stop execution.

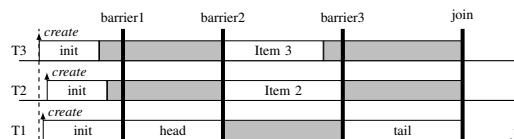
B. Manual code generation

There are C and PYTHON interpreters of the NNEF format [NNE18] but only for traditional CPU target. Consequently, no existing tool supports our syntax extension nor state-of-the-art GPU. Thus we developed the code for each item using C++

and CUDA using the CUDNN library. Basically the C++ code is executed by the ARM processor whereas CUDA allows the definition of kernels that are executed synchronously by all CUDA cores. The CUDNN library is built on top of CUDA for executing common neural networks layers.

1) *Software architecture*: Practically, each type of layer is implemented using a dedicated C++ class that inherits from the abstract *Layer* class that defines common attributes and methods to be implemented (*init()* and *forward()*) by child classes. In effect, *init* statically allocates tensors and CUDNN descriptors while *forward* launches the layer computation based on CUDNN for Convolution and max pooling layers. Each item contains one object implementing a static scheduler. More precisely, during the *init* phase, each item creates an object for each layer which are stored in ordered lists. Thus, items 2 and 3 need one single ordered list whereas item 1 needs two ordered lists (one for the first part and one for the second part). During *forward*, the scheduler calls in order the *forward* of objects stored in ordered lists.

2) *Scheduling*: We define one separate thread for each item allocated to one CPU + CUDA cores. More precisely, synchronizations between threads use *pthread_barrier_t* and associated APIs (*barrier_init* and *barrier_wait*). Barriers synchronize accesses to shared variables.



The execution sequence starts with the 3 threads creation on the CPUs and then reaches the first synchronization barrier. Then Item1 thread calls the *forward* method of layers of the head (until *send_var*) while Items 2 and 3 threads wait for the second synchronization barrier. After the second synchronization barrier, Items 2 and 3 threads call *forward* method of their layers while Item1 thread waits for the third synchronization barrier. After the third synchronization barrier, Item 1 thread calls *forward* method of layers of the tail. At last, threads join and exit.

C. Semantic preservation of the Petri net

The first analysis aims at verifying that all observed scheduling of layers on the XAVIER respects the Petri net semantics. Because we use a static scheduler, all schedules should behave as shown in section V-C which is included in the semantics of coloured Petri net of figure 6. For that, we logged each start/end of branches and layers and we stressed the robustness of the implementation by adding some temporal noises (sleep in the code).

All observed traces respected the schedule of section V-C with some timing variations. When observing the implementation with no noise, execution traces of Item 2 and 3 are interleaved on the GPU. When adding a wait of 1s at the beginning of Item 2 (just after barrier1), all layers of Item 3 were executed before those of Item 2.

D. Semantics preservation of the function

The second instrumentation mechanism aims at checking that the functional semantics of the DNN is preserved. We achieve this by re-implementing the NNEF specification in PYTORCH. Then we define 100 random vectors that we run both on the PYTORCH implementation and on the C++ implementation on the NVIDIA target. Finally, we compute the overall average error mean between both executions for the 100 runs.

We were not able to find the exact convolution algorithm of PYTORCH. We think that it exists a non documented heuristic that calls the best algorithm (considering execution time) depending of the convolution parameters and available hardware. According to the CUDNN documentation [NVI19b], it is possible to select the convolution algorithm among a list, but details of the implementation are not given. Thus, there may be a discrepancy between convolutions that we cannot fix. The average error mean is extremely small 1.10^{-7} for FLOAT32 using 3 CUDNN algorithms (namely gemm, Winograd and direct). Nevertheless numeric precision results for this experiment are in an acceptable range that is very close to the available numeric precision of floating point representation and this also is observed by other frameworks [SCGP22].

E. Measured Execution Time (MET)

One objective of the DO-178C that we did not mention until now is the capacity to estimate the Worst Case Execution Time (WCET). Due to the complexity of NVIDIA target, a formal demonstration using static analysis may be difficult. But at least, a good property is a low variation of the measured execution time among several executions. In our case, the generated code does not contain any IF-THEN-ELSE patterns or dynamic loop conditions. Thus, the variability is only linked to the hardware behavior. We measured the MET of the complete DNN and of the first convolution of Item 1 over 10 runs. We rely on the *nsys* tool from NVIDIA to get timing measurements.

	Mean(MET)	MIN(MET)	(MAX(MET))	STD(MET)
First Conv	324 2976 ns	322 688 ns	326 656 ns	1.45 ns
DNN	24 257 μ s	16 285 μ s	53 950 μ s	13 526 μ s

The MET of the first convolution is very stable with a very low jitter. The MET distribution of the DNN is large and to understand why, we need to investigate the low level behaviour. NVIDIA GPUs are black-boxes processors on which we cannot guarantee worst-case execution time [AA21], [ACR22].

VI. RELATED WORK

There are plenty of different formats but no consensus within the community. State-of-the-art frameworks like PYTORCH, or TENSORFLOW rely on custom black-box formats built on top of *protocol buffers* [Goo01] developed by Google. A *protocol buffer* is a structured binary format that is not human readable and requires conversion tools and template files to be interpreted. Thus, the syntax is not formally defined and specification of layers are only available through documentation website. For example, TENSORFLOW proposes

the *.h5* [HDF01] format and KERAS format [Ten15] builds on top of protocol buffer. Moreover the way to save a neural network is not unified among frameworks and may evolve every updated version with poor backward compatibility. When moving from caffe to caffe2 (known today as PYTORCH), the caffe [JSD⁺14] format was not supported anymore.

All previous formats were developed specifically for training frameworks (open source or proprietary) without any objective for sharing models. Their main purpose was to allow saving and reloading previous trained models without too much consideration on syntax and semantics. ONNX [BLZ⁺19] and NNEF [The22] were developed with the objective to be independent from frameworks. ONNX is still based on binary protocol buffers [Goo01] (so without a textual syntax) but is proposing a functional semantics through its github site. On the contrary NNEF is proposing a textual format with a syntax and a semantics that is formally defined in a specification. Unfortunately, the NNEF format suffers from a small community and tools supporting the format.

NNet [The22] format is an example of ad-hoc format. RELUPLEX neural networks examples [KBD⁺17] are in NNet. It is based on textual files but without definition of syntax and semantics. Import and export tools are provided, but its utilization for sharing neural networks between teams remains supremely painful.

Some other formats like [CE-17], [Apa18], [LAB⁺21a] tackle the need for intermediate representation between a neural network description and an implementation on a target. Especially this supports different optimizations passes like layers folding or low level tensor manipulation description like in LLVM [Lat02]. We consider that we are closest to programming language than from a neural network description format. Most of the time ONNX or NNEF are used as input like in [LAB⁺21b], [PBCPB20], [JBL⁺20].

Because DNN are data-flow, it is natural to translate them into *synchronous languages*. There are some works such as [LFG20] that proposed to encode them as Synchronous Dataflow Graphs or SCADE tool suite [CPPP18] which is currently developing a DNN libraries. Once the translation is done, it is then possible to reuse all the qualified code generation tools. This is complementary to our work as we could use the NNEF description to generate the SCADE program for instance. To the best of our knowledge, none of actual available neural network description formats propose solution for describing multi items implementation with concerns on sharing variables among them.

VII. CONCLUSION

We have proposed a formal extension of NNEF that takes into account the *execution model* of a description and allows for the modification of a description of a trained model to define traceable distribution and parallelisation optimizations that preserves the semantics while improving the execution time compared to a pure sequential approach. We have also proposed a code generation strategy based on barriers for exchanging data between items. As a future work, a working group has been set up to propose an ONNX aeronautics profile.

Acknowledgments

This work has benefited from the AI Interdisciplinary Institute ANITI, funded by the “Investing for the Future – PIA3” program Grant agreement ANR-19-P3IA-0004 and from the PHYLOG 2 project funded by the French government through the France Relance program, based on the funding from and by the European Union through the NextGenerationEU program.

REFERENCES

- [AA21] Tanya Amert and James H. Anderson. Cupid^{fl}: Detecting improper GPU usage in real-time applications. In *24th IEEE International Symposium on Real-Time Distributed Computing, ISORC 2021, Daegu, South Korea, June 1-3, 2021*, pages 86–95. IEEE, 2021.
- [ACR22] Michaël Adalbert, Thomas Carle, and Christine Rochange. PasTiS: building an NVIDIA Pascal GPU simulator for embedded AI applications. In *11th European Congress on Embedded Real-Time Systems (ERTS 2022)*, 2022.
- [Apa18] Apache. TVM, 2018. <https://tvm.apache.org/>.
- [BBD⁺17] Timothy Bourke, Léo Brun, Pierre-Évariste Dagand, Xavier Leroy, Marc Pouzet, and Lionel Rieg. A formally verified compiler for lustre. In Albert Cohen and Martin T. Vechev, editors, *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2017, Barcelona, Spain, June 18-23, 2017*, pages 586–601. ACM, 2017.
- [BLZ⁺19] Junjie Bai, Fang Lu, Ke Zhang, et al. Onnx: Open neural network exchange. <https://onnx.ai/>, 2019.
- [CE-17] CE-List. N2D2 frameworks, 2017. <https://github.com/CEA-LIST/N2D2>.
- [CFR⁺89] Ron Cytron, Jeanne Ferrante, Barry K Rosen, Mark N Wegman, and F Kenneth Zadeck. An efficient method of computing static single assignment form. In *Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 25–35, 1989.
- [CPPP18] Jean-Louis Colaço, Bruno Pagano, Cédric Pasteur, and Marc Pouzet. Scade 6: From a kahn semantics to a kahn implementation for multicore. In Hiren Patel, Tom J. Kazmierski, and Sebastian Steinhorst, editors, *2018 Forum on Specification & Design Languages, FDL 2018, Garching, Germany, September 10-12, 2018*, pages 5–16. IEEE, 2018.
- [CRM⁺09] Alfons Crespo, I. Ripoll, M. Masmano, P. Arberet, and Metge Jean-Jacques. Xtratun: An open source hypervisor for tsp embedded systems in aerospace. pages 31–, 05 2009.
- [EUR21] EUROCAE WG-114/SAE joint group. Certification/approval of aeronautical systems based on AI, 2021. on going standardization.
- [GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.
- [Goo01] Google. Protocol Buffers, 2001. <https://portal.hdfgroup.org/display/HDF5/Introduction+to+HDF5>.
- [GP23] Adrien Gaufrin and Claire Pagetti. Formal description of ml models for unambiguous implementation, 2023.
- [HDF01] HDF group. Introduction to HDF5, 2001. <https://www.hdfgroup.org/solutions/hdf5/>.
- [HSW89] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multi-layer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.
- [JBL⁺20] Tian Jin, Gheorghe-Teodor Bercea, Tung D. Le, Tong Chen, Gong Su, Haruki Imai, Yasushi Negishi, Anh Leu, Kevin O’Brien, Kiyokuni Kawachiya, and Alexandre E. Eichenberger. Compiling onnx neural network models using mlir, 2020.
- [JK09] Kurt Jensen and Lars M. Kristensen. *Coloured Petri Nets: Modelling and Validation of Concurrent Systems*. Springer Publishing Company, Incorporated, 1st edition, 2009.
- [JSD⁺14] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.
- [KBD⁺17] Guy Katz, Clark W. Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. *CoRR*, abs/1702.01135, 2017.
- [LAB⁺21a] Chris Lattner, Mehdi Amini, Uday Bondhugula, Albert Cohen, Andy Davis, Jacques Pienaar, River Riddle, Tatiana Shpeisman, Nicolas Vasilache, and Oleksandr Zinenko. MLIR: Scaling compiler infrastructure for domain specific computation. In *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, pages 2–14, 2021.
- [LAB⁺21b] Chris Lattner, Mehdi Amini, Uday Bondhugula, Albert Cohen, Andy Davis, Jacques A. Pienaar, et al. MLIR: scaling compiler infrastructure for domain specific computation. In Jae W. Lee, Mary Lou Soffa, and Ayal Zaks, editors, *International Symposium on Code Generation and Optimization (CGO)*, pages 2–14, 2021.
- [Lat02] Chris Lattner. LLVM: An Infrastructure for Multi-Stage Optimization. Master’s thesis, Computer Science Dept., University of Illinois at Urbana-Champaign, Urbana, IL, Dec 2002. [See http://llvm.cs.uiuc.edu](http://llvm.cs.uiuc.edu).
- [LFG20] Daniel Luenemann, Maher Fakh, and Kim Gruettner. Capturing neural-networks as synchronous dataflow graphs. In *MBMV 2020 - Methods and Description Languages for Modelling and Verification of Circuits and Systems; GMMITG/GI-Workshop*, pages 1–10, 2020.
- [NNE18] NNEF project. NNEF GitHub tools repository, 2018. <https://github.com/KhronosGroup/NNEF-Tools/>.
- [NVI19a] NVIDIA - Dustin Franklin. Introducing the Jetson Xavier Nx, 2019. <https://developer.nvidia.com/blog/jetson-xavier-nx-the-worlds-smallest-ai-supercomputer/>.
- [NVI19b] NVIDIA developer team. CudNN library documentation, 2019. <https://docs.nvidia.com/deeplearning/cudnn/api/index.html>.
- [Old86] Ernst-Rüdiger Olderog. Operational petri net semantics for ccsp. In *European Workshop on Applications and Theory in Petri Nets*, pages 196–223. Springer, 1986.
- [PBCPB20] Hugo Pompougnac, Ulysse Beaugnon, Albert Cohen, and Dumitru Potop-Butucaru. From SSA to Synchronous Concurrency and Back. Research Report RR-9380, INRIA Sophia Antipolis - Méditerranée (France), December 2020.
- [Pet77] James L Peterson. Petri nets. *ACM Computing Surveys (CSUR)*, 9(3):223–252, 1977.
- [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.
- [SAE10] SAE/EUROCAE. Aerospace Recommended Practices ARP4754a/ed-79a- development of civil aircraft and systems, 2010.
- [SCGP22] Iryna De Albuquerque Silva, Thomas Carle, Adrien Gaufrin, and Claire Pagetti. ACETONE: predictable programming framework for ML applications in safety-critical systems. In *34th Euromicro Conference on Real-Time Systems ECRTS 2022*, pages 3:1–3:19, 2022.
- [SZ06] Anton Maximilian Schäfer and Hans Georg Zimmermann. Recurrent neural networks are universal approximators. In Stefanos D. Kollias, Andreas Stafylopatis, Włodzisław Duch, and Erkki Oja, editors, *Artificial Neural Networks – ICANN 2006*, pages 632–640, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.
- [Ten15] TensorFlow. Keras Model, 2015. https://www.tensorflow.org/api_docs/python/tf/keras/Model.
- [The22] The Khronos NNEF Working Group. Neural Network Exchange Format – Version 1.0.5, 2022. <https://registry.khronos.org/NNEF/specs/1.0/nnef-1.0.5.pdf>.
- [WEE⁺08] Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat, and Per Stenström. The worst-case execution-time problem—overview of methods and survey of tools. *ACM Trans. Embed. Comput. Syst.*, 2008.
- [Xil19] Xilinx. ZCU102 Evaluation Board – User Guide, 2019. <https://docs.xilinx.com/v/u/en-US/ug1182-zcu102-eval-bd>.

Session We.1.B
Realtime Interference

Wednesday 12th June

10:00

—

Lauragais Room

A Refinement Method for Interference Analysis using the PHYLOG Modeling Language

Guillaume Brau, Eric Jenn
IRT Saint-Exupéry
Toulouse, France

Emmanuel Courty
IRT Saint-Exupéry &
Liebherr Aerospace
Toulouse, France

Kevin Delmas, Frédéric Boniol
ONERA
Toulouse, France

Abstract—Temporal interference may occur in multicore processor systems due to tasks running in parallel competing for shared resources such as buses or memories. This paper presents a model-based interference analysis based on the PHYLOG framework that intends to help in the certification process of multicore aeronautical systems. As PHYLOG does not define a clear modeling method, a refinement approach is proposed to model the system using the PHYLOG Modeling Language (PML). Our objective is to define a process that enables to build a model that is both precise and reliable so that analysis results are sound. The approach is finally validated on an industrial use case from the aerospace domain.

I. INTRODUCTION

In the last decade, multicore processors have become the norm in the general market. However, their use in critical real-time systems still represents a challenge since those systems have to meet strong temporal requirements with a high level of confidence and have to comply with regulatory requirements.

In order to demonstrate compliance with temporal requirements, upper bounds on execution times accounting for all contributions at hardware and software levels must be estimated. In the avionics domain, for instance, the AMC 20-193 [1] standard requires that temporal interferences which may occur when tasks running in parallel access shared resources (e.g., caches, memories, buses) are addressed. For example, the *MCP_Ressource_Usage_3* (RU3) demands that: “the applicant has identified the interference channels that could permit interference to affect the software applications hosted on the multicore processor cores, and has verified the applicant’s chosen means of mitigation of the interference”.

Towards that goal, model-based approaches can be used to help in the certification process of aeronautical systems that use multicore processors. These solutions build on a language that allows to describe the system and an analyzer to infer properties about it: one can e.g. combine the AADL language to describe the architecture of the system and Prolog to identify interferences [2], or use the LNT formal specification language to capture the system’s behavior and the CADP toolchain to detect interferences [3]. However, methodological aspects to use these tools remain mostly unaddressed and must be further defined in order to be used in an industrial context: *How to build a relevant model of the system? What to learn from it? How to use it in a certification process?*

In this paper, we focus on interference analysis using the PHYLOG approach [4]. In PHYLOG, the system architecture is to be described using the PHYLOG Modeling Language (PML), which model is then analyzed in order to identify interferences. Hence, the construction of the PML model is a critical point when applying the approach: *How to build the PML model?* In this paper, we propose to apply a refinement process in order to build an accurate and reliable model of the system. This process implements different types of refinements: structural (i.e., improving the description of the components of the architecture such as memories and buses), temporal (e.g., integrating data from the scheduling plan), etc. Our method is applied to an industrial use case from Liebherr, which is deployed on the AURIX TC399XE platform.

This paper is organized as follows: Section II deals with related work. We briefly introduce the use case in Section III. The refinement method is presented in Section IV with PML views implementing the method in Section V and application to the use case in Section VI. Finally, we discuss our approach (Section VII) and conclude with possible perspectives (Section VIII).

II. RELATED WORK

The interference problem is identified e.g. in [5] as “alterations of the processor’s behavior seen by software running on one core due to activities ordered by software running on other cores.”. Thus, certification authorities such as EASA and FAA notably set interference-related objectives for the certification of multicore systems [6], [1]. Focusing on the argumentation to reach these objectives, [4] propose to organize an argumentation strategy into a series of *evidences* (or sub-claim) towards the *claim* (the objective to demonstrate). They show in particular that the aforementioned RU3 objective involves identifying all interferences (*sub-objective 1*) and to classify their effects (*sub-objective 2*). We review some related work on these topics.

a) *Evaluation of interference effects*: A first class of works aims to evaluate the impact of contentions occurring within shared hardware resources on the applications execution times. For instance, [7] considers a multicore processor architecture composed of a single bus providing access to a shared memory, and it proposes a method to determine an upper bound on the number of bus requests that software tasks can generate

in a given time interval. Both [8] and [9] focus on measurement techniques based on dedicated stressing benchmarks and hardware monitors to characterize the architecture and the shared resources that can cause interferences between software applications.

b) Identification of interferences: Another class of works relies on a formal model of the architecture and a formal analysis method to explore the set of interference channels. A previous work by Brindejone et al. [10] proposes a way to characterize the interference behavior and to identify interference channels on a multicore processor. Their approach consists in identifying a set of test classes that completely covers the interferences that could occur in the architecture. However, they consider multicore architectures as black boxes, thus ignoring internal components of the architecture. In [2], the authors propose a tool ("Strange") that identifies interferences using an AADL structural model of the SoC that is first translated into a set of (Prolog) facts that is queried using a Prolog program. Several works have tried to circumvent the limitations of a pure structural representation of the SoC. In [3], for instance, the authors use LNT to capture the behavior of the Infineon TC275's crossbar arbiter and exploited the CADP toolchain to detect interferences. Two approaches have been investigated: PATCHCHECK, based on the detection of predefined patterns showing the manifestation of interference (i.e., the interleaving of a request concerning core Y in the sequence of transaction concerning core X), and SYNCHECK, based on the comparison of traces obtained in isolation and in contention. These two methods have been applied on a small part of the SoC, and their scalability has not been demonstrated.

Finally, note that the previous works either address sub-objective 1 (identification of interferences) or sub-objective 2 (evaluation of interference effects). In other words, both objectives are addressed *separately*, without stating whether they are related and how. In our approach both activities act in a complementary way: the PML model is used to identify the set of interference scenarios that must be evaluated; evaluation, in turn, enables to estimate the impact of the scenarios on the software and if this impact is compliant with the constraints defined by the applicant. Thus, the construction of the PML model is driven by the evaluation: the PML model must be refined until interference effects fulfill the constraints (see Section IV).

III. USE CASE

Hereafter, we successively present the software and hardware parts and the execution environment of the industrial use case developed by Liebherr Aerospace (referred to as LTS use case in the following).

a) Application: The software parts consist of two distinct legacy applications as well as a common board support package (BSP) software:

- the Integrated Air Management System (IAMS) that manages air bleed, air conditioning and cabin pressure control,
- the Power Electronics (PE) system that controls electrical motors,

- the common BSP software that deals with Inputs/Outputs (I/Os), e.g., GPIO, PWM, etc.

The IAMS and PE sub-systems do not coexist on the same hardware platform in the actual system. However, we deploy them on the same hardware target in order to (i) address different typologies of timing constraints (20 Hz for the IAMS, 20 kHz for the PE), (ii) increase the pressure on the use of shared resources provided by the platform, and (iii) investigate the integration of multiple heterogeneous applications on the same computation platform.

b) Hardware platform: The application software is integrated in a 6-cores Infineon AURIX TC399XE processor [11], which simplified architecture is depicted in Figure 1. The AURIX has been designed in order to minimize possible hardware interferences: each core has dedicated program (PSPR, PFLASH) and data (DSPR, DLMU) memories, 3 shared memories (LMUs) can be used by the cores through a crossbar (SRI), and a shared bus (FPI) is provided to access I/Os.

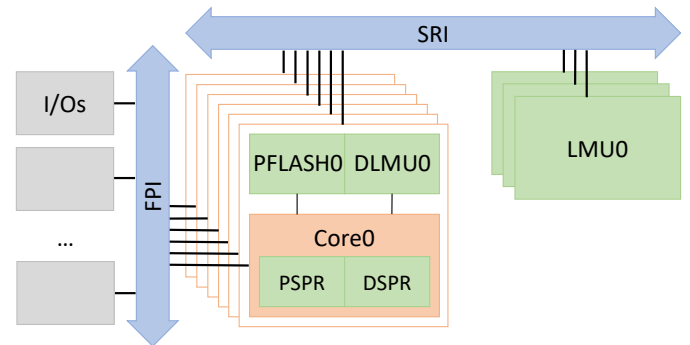


Fig. 1: AURIX TC399XE platform (simplified view).

c) ASTERIOS: The ASTERIOS toolchain and execution environment, provided by ASTERIOS Technologies [12], is used to integrate both applications and the common BSP on the AURIX platform and enforce the timing constraints.

Applications are developed using PsyC [13], a software architecture description language based on the synchronous Logical Execution Time (sLET) paradigm [14]. In this model, each new activation of a task is constrained by an earliest start date and a deadline specified by the user based on logical clock ticks. The IAMS application is implemented with two PsyC tasks (ag_iam and ag_cpcs), the PE application includes two tasks (ag_fast and ag_slow) and the BSP three tasks (worker_gpio, worker_adc and worker_pwm).

The PsyC design is then compiled, together with the user application code, into an executable binary that is executed by the ASTERIOS's real-time micro-kernel. At run time, the kernel relies on a static schedule generated by the ASTERIOS toolchain, called Repetitive Sequences of Frames (RSF), to enforce the tasks timing constraints and determinism (in multicore architectures, one RSF is generated per core). An RSF is divided into intervals, and each task may be given at most a frame within each interval for its execution (Figure 2). A frame corresponds to the CPU time allocated to a task, and is computed from the time budget provided by the user.

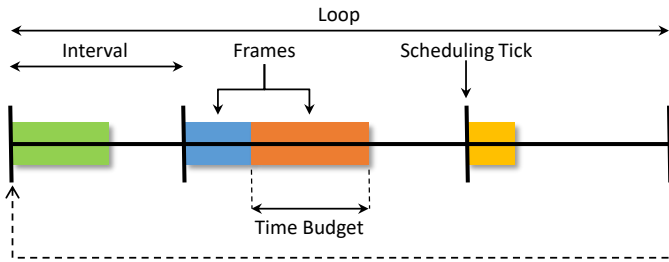


Fig. 2: Repetitive Sequences of Frames (RSF) in ASTERIOS.

IV. APPROACH

This section presents the refinement approach that is based on the PML language.

A. PHYLOG Modeling Language

The PHYLOG Modeling Language (PML) provides a set of constructs capturing the concepts identified in the AMC 20-193 and supporting interference analysis. Here, we briefly introduce the main elements of the language, further details can be found e.g. in [4].

a) *PML model*: A system can be described in PML in two main *views*: structural and behavioral.

The *structural* view describes the system architecture with:

- platform, i.e., the set of hardware components (cores, memories, communication resources, etc.) and physical connections between them,
- software, i.e., the software applications and their data,
- the allocation of the software on the platform : applications are allocated on cores and data on memories,
- routing aspects, e.g., paths from sources to targets.

The *behavioral* view details transactions (core accesses to shared resources, e.g. memory read/write) occurring on the platform and provides additional interference specifications.

b) *Semantics of a PML model*: Following [15], the platform in a PML model describes *initiators* (e.g., a core), *targets* (e.g., a memory component), and *transporter* (e.g., a bus). Applications are allocated to initiators and data to targets. Hardware components provide one or several *services* (e.g., store or load). When an application requests to access or write a data, it will trigger a *single transaction*, i.e., a sequence of components' services used to fulfill the request. For instance, to load a data from the *LMU0*, an application running on the *Core0* will trigger the single transaction: $Core_0^{ld} \cdot SRI^{ld} \cdot LMU_0^{ld}$ where C^{ld} stands for *load service of component C*. Several single transactions can be triggered by the applications executed by the initiators. A set of concurrent single transactions is called a *multi-transaction*. For instance, the multi-transaction $\tau = (Core_0^{ld} \cdot SRI^{ld} \cdot LMU_0^{ld} || Core_1^{ld} \cdot SRI^{ld} \cdot LMU_0^{ld})$ is the concurrent access by *Core0* and *Core1* to the *LMU0* through the *SRI*. The set of all possible multi-transactions, denoted \mathcal{T} , of a given PML model M is then all the possible concurrent single transactions that can be triggered by the applications executed by the initiators. An interference occurs when the simultaneous usage of a set of services may impact timing behavior. A typical

case is the simultaneous use of a single service by several single transactions in the multi-transaction. For instance, in the multi-transaction τ , the load services of the *SRI* and *LMU0* are used concurrently by the two cores. The purpose of the PML analyzer is then to identify efficiently the multi-transactions that may produce an interference.

B. Refinement Approach

In our approach, following AMC 20-193 objectives, a PML model is used as an appropriate abstraction of the system to (1) identify a set of interferences, and then (2) evaluate the effects of interferences.

a) *Interference identification*: Let M a PML model, interference identification is the analysis applied over the PML model $Id(M)$ that classifies multi-transactions \mathcal{T} described in M into \mathcal{I} , interfering multi-transactions, and \mathcal{F} , non-interfering multi-transactions.

As explained in Section IV-A, an interference in PML comes from the concurrent use of a service (or exclusive services) provided by a hardware component within a multi-transaction. An interference involves n (≥ 2) transactions, so we talk about n -ary interference (itf- n for short). The PML analyzer identifies the multi-transactions which may compete for services provided by the components of the platform (*interference scenarios*) and the components where those conflicts occur (*interference channels*). Complementary, non-interfering scenarios (non-interfering multi-transactions) are also provided. The PML analyzer relies on MONOSAT solver [16] to compute interferences.

b) *Evaluation*: Once interference scenarios have been identified, their effects on the software is evaluated in order to check compliance with the constraints defined by the applicant. For instance, the constraints may relate to real-time, requiring to evaluate response times in order to check that the tasks meet their deadlines (see Section VI-B for an example of evaluation).

c) *Refinement process*: The PML model being the main source of information to perform interference analysis, it shall satisfy the following properties:

- *Coverage*: the model enables to identify all interferences,
- *Precision*: the model enables to identify only interferences.

Coverage impacts safety, so it is explicitly required by the AMC 20-193. *Precision* impacts cost, as it may lead to unnecessary verification activities and over-design.

Assuming an initial model that would *satisfy coverage* and a refinement process that would *preserve coverage*, the approach consists in refining the model in order to increase precision *in order to meet the constraints*. In practice, and due to the difficulty to ensure coverage *by design*, additional verification and validation activities are also required. One way to ensure coverage is for example (i) to make reasonably conservative modeling assumptions and (ii) to verify these assumptions experimentally (see Section VI-C for an example of validation).

Let us define more precisely what is meant here by refinement and what are the objectives of the refinement process.

False-positives and false-negatives: A PML model is an abstraction – thus imperfect representation – of the system that may lead to identify false-positives and false-negatives. Let I (resp. F) the set of *actual* interfering (resp. non-interfering) multi-transactions that would be observed on the target, false-positives $\mathcal{FP} = \mathcal{I} \cap F$ are multi-transactions that are erroneously classified as interfering and, conversely, false-negatives $\mathcal{FN} = \mathcal{F} \cap I$ are multi-transactions that are erroneously classified as non-interfering. The presence of false-positives means that the model is imprecise, whereas the presence of false-negatives means that the coverage objective is not fulfilled.

Refinement (and validation): Refinement is a function $R(M_i) = M_{i+1}$ that adds details on the model, thus restricting the set of the possible behaviors and increasing the accuracy of analysis. Applied to the PML model, the consequence of refinement is that reported false-positives are reduced: $\mathcal{I}_{i+1} \subset \mathcal{I}_i$ and, consequently, $\mathcal{FP}_{i+1} \subset \mathcal{FP}_i$.

The objective is therefore to apply refinement so that the number of false-positives \mathcal{FP} is minimized (precision objective) in order to meet the constraints. In addition, validation of the model must ensure that all interferences are identified (coverage objective), i.e., $\mathcal{FN} = \emptyset$.

The refinement process is illustrated in Figure 3. Refinement of the PML model is represented on the horizontal axis: M_i is the model at phase i of the process (represented with a white circle in Figure 3). Interference identification is shown on the vertical axis: Id inputs the PML model M_i and outputs the set of interfering multi-transactions \mathcal{I}_i and non-interfering multi-transactions \mathcal{F}_i . Evaluation is illustrated in the gray rectangle in the bottom left-hand corner of Figure 3: the evaluation function Ev inputs identified interfering multi-transactions \mathcal{I}_i and enables to conclude whether the constraint is met (acceptable state) or not (failure state). If effects are not acceptable, the PML model can be refined in order to reduce \mathcal{I} and make the evaluation more precise: in Figure 3, if $Ev(\mathcal{I}_i)$ is not acceptable, M_i is refined into M_{i+n} and evaluation is applied on the refined \mathcal{I}_{i+n} set ($\mathcal{I}_{i+n} \subset \mathcal{I}_i$). The refinement process finishes when either (1) evaluated interference effects are acceptable or (2) the PML model cannot be refined anymore (in which case the system must be redesigned in order to mitigate interferences).

We explain in the following sections how the PML views can be refined (Section V) and show an application of our approach to the case study (Section VI).

V. REFINEMENT AND PML VIEWS

This section explains how refinement can be applied to the different PML views. Taking the LTS use case for instance, we describe the initial, coarse-grained, model and illustrate different refinements applied to the structural and behavioral views.

A. Baseline Model

A PML model is a Scala application in which each file describes an aspect of the architecture through a dedicated class/interface. A PML model is thus modular, making it possible to easily analyze variants of the system, e.g., models at different refinement stages.

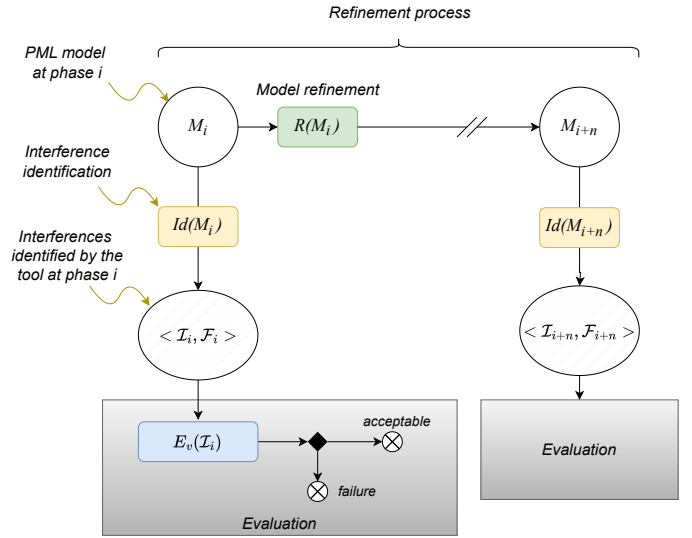


Fig. 3: Refinement process.

Modeling starts with the baseline model (M_0) that applies the most conservative modeling assumptions with information that can be easily found in the datasheet (e.g., block diagram of the platform). The initial model is as simple as possible with very little information on the parallelization capabilities of the system: no (or few) assumptions are made on the internal structure of the components, e.g., we make the assumption that the applications all execute at the same time, that transactions all take the same path, etc.

The initial model thus only describes basic structural views (platform, software and allocation) and transactions.

```
class core_unit(name: Symbol) extends Composite(name) {
  // core
  val core: Smart = Smart()

  // internal memories
  val pspr: Target = Target()
  val dspr: Target = Target()

  //connections (the core accesses its private memories)
  core link pspr
  core link dspr
}
```

Listing 1: Composite structure of a core unit.

a) *Platform:* The TC399XE platform used in the LTS use case embeds several core units (see Figure 1), which composite structure is described in Listing 1: a core unit includes a core (defined as an initiator with the "smart" keyword), internal memories (defined as targets), and connections between them. The full platform is described via the "AURIXPlatform" class (Listing 2) that enumerates the hardware components (core units, memories, buses, I/Os) and sets the physical links between them. The excerpt provided in Listing 2 shows the declaration of core_unit1 connected to DLMU1 (direct connection) and LMU0 (via the SRI). In the following, core X will denote the core component in core_unit X (core_unit X .core), the same

```

class AURIXPlatform(name: Symbol) extends Platform(name) {

  // core units
  val core_unit1 = new core_unit()
  // memories
  val dlmul: Target = Target()
  val lmu0: Target = Target()
  // buses
  val sri : SimpleTransporter = SimpleTransporter()

  // connections of the cores
  core_unit1.core link dlmul // direct connection to the dlmul
  core_unit1.core link sri // connection to the sri

  // connection of the lmu to the sri
  sri link lmu0
  [...]
}

```

Listing 2: Platform model of the AURIX TC399XE (excerpt).

```

trait AURIXCoarseSoftwareAllocation extends Configuration {
  self : AURIXCoarsePlatform =>

  //Tasks
  val ag_fast : Application = Application()

  //Data
  val code_ag_fast : Data = Data() //code
  val data_ag_fast : Data = Data() //data

  //Tasks allocation
  ag_fast use core_unit2.core

  //Data allocation
  code_ag_fast in core_unit2.pspr
  data_ag_fast in lmu0

  [...]
}

```

Listing 3: Software model of the LTS use case (excerpt).

notation will apply to designate `core_unitX.pspr` (PSPRX) and `core_unitX.dspr` (DSPRX).

b) Software and allocation: The software comprises applications (running on cores) and their data (stored in memories). The "AURIXSoftwareAllocation" class (Listing 3) first declares the tasks and the various ASTERIOS components (real-time kernel, etc.) together with their data (tasks code and data, communication data, etc.). The allocation of both applications (on cores) and their data (on memories and I/Os) is then provided. For instance, Listing 3 shows the declaration of one application `ag_fast`, executed on `core2`, and its data, stored in `PSPR2` (for its code) and `LMU0` (for its data).

c) Transactions and configuration: With the platform components and pieces of software defined, it is then possible to provide more information on the usage of resources. The *transaction library* lists transactions that can be performed by the tasks. A transaction declaration involves a task, a data and a type (e.g., read or write). For instance, Listing 4 shows different examples of transactions initiated by `ag_fast`: code loading and data read/write. One or more *configurations* to analyze can then be defined, a configuration being a particular set of transactions

```

trait AURIXTransactionLibrary extends TransactionLibrary {
  self : AURIXCoarsePlatform with AURIXCoarseSoftwareAllocation =>

  // tasks load code
  val ld_code_ag_fast : Transaction = Transaction(ag_fast read
    code_ag_fast)

  // tasks read/write data
  val wr_data_ag_fast : Transaction = Transaction(ag_fast write
    data_ag_fast)
  val rd_data_ag_fast : Transaction = Transaction(ag_fast read
    data_ag_fast)
  [...]
}

```

Listing 4: Transaction library (excerpt).

within the transaction library.

B. Refinement

Model refinement will consist in adding precision on both structural and behavioral views. Each refinement stage aims to reduce the set of interferences by removing false-positives.

We identify several types of refinements:

- *Structural refinement* is to improve the description of the components of the architecture (e.g., memories, buses),
- *Routing refinement* is to specify the paths from initiators to targets when several paths are possible,
- *Temporal refinement* is to provide temporal exclusions,
- *Quantitative refinement* is to describe resource usage.

a) Structural refinement: Structural refinement seeks to improve the description of the components of the architecture (e.g., memories, buses).

As seen in the baseline model, the SRI is modeled as a "black box", i.e., it is specified as a simple "transporter" (Listing 2). Observing that the SRI is a main contributor to interferences, we may "open" the black box and model the internal structure of the component. For example, in Listing 5, the SRI crossbar network is modeled as a composite: the SRI is made up of a set of input and output ports (modeled as transporters) where all inputs are connected to all outputs. Therefore, every input port is intended to serve a particular core and every output port is connected to a distinct memory.

b) Routing: Specifying routes may be useful, e.g., when multiple paths to a target are possible. For example, in the AURIX platform each core ($core_n$) uses a local PFLASH memory ($PFLASH_n$) that can be accessed through a direct link. Meanwhile, all PFLASHs can be accessed by all cores via the SRI. Thus, there exists 2 paths between $core_n$ and $PFLASH_n$: (1) one that uses the direct link, and (2) one through the SRI. As in fact transactions from $core_n$ to $PFLASH_n$ use the direct link, it is necessary to specify a route in order not to count them in the SRI. For this, the configuration is extended with routing constraints, e.g., to specify that the route for transactions issued by `core0` targeting `PFLASH0` uses the direct link between `core0` and `PFLASH0`.

c) Temporal refinement: Timing aspects such as tasks parameters (periods, time budgets, etc.) or scheduling affect interferences. Therefore, it would be interesting to integrate

```

class xbar(name: Symbol) extends Composite(name) {

  // input ports
  val i1: SimpleTransporter = SimpleTransporter ()
  val i2: SimpleTransporter = SimpleTransporter ()
  [...]

  // output ports
  val o1: SimpleTransporter = SimpleTransporter ()
  val o2: SimpleTransporter = SimpleTransporter ()
  [...]

  // connections (all inputs connected to all outputs)
  for {
    input <- Set(i1, i2, i3, i4)
  } {
    for {
      output <- Set(o1, o2, o3, o4, o5, o6, o7, o8, o9)
    } {
      input link output
    }
  }

  [...]
}

```

Listing 5: Structural refinement: composite structure of the SRI (excerpt).

temporal information, specified in the PsyC design or in the generated RSF, in the PML representation. While PML does not directly support such description of timing aspects, we can capture temporal information with two means: slices or exclusivity clauses.

In the first method, a temporal "slice" represents a remarkable execution unit whose transactions are encoded as a configuration. Each configuration is intended to be analyzed separately before the results from the different configurations are aggregated (duplicated interferences are counted once). The second way, which is the chosen solution in the current model, is to express (temporal) exclusions in the model. For example, by analyzing the RSF of the LTS use case, we can specify in PML that `worker_adc`, `worker_gpio`, `worker_pwm` (core1) and `ag_fast` (core2) are never executed at the same time.

d) Quantification: In some cases, a transaction can have a negligible impact on a service. In that case, one can assume that the concurrent usage of this service by other transactions will not result in a significant impact (e.g., in terms of execution times) on the application. This assumption can be specified in the PML model by identifying the transactions that do not affect specific services.

VI. APPLICATION ON THE USE CASE

This section shows an application of our approach (Section IV) and proposed refinements (Section V) to the LTS Use Case (Section III).

We illustrate the main activities – identification of interferences, evaluation and validation – with, for example, the schedulability constraint that tasks execution times (ET) must comply with time budgets (B) defined in the ASTERIOS application architecture (see Section III, Figure 2), i.e.,

$$\forall t, ET(t) \leq B(t) \quad (1)$$

A. Identification of Interferences

We build the baseline model and then apply structural, routing and temporal refinements as explained in Section V. The next paragraphs discuss the main results (interference scenarios and interference channels) for each refinement stage and, finally, summarize results for the overall process.

a) Baseline model: The PML analyzer enables to identify n -ary interference scenarios (i.e., interference scenarios involving n transactions). Interference channels are also identified. For instance, `<rd_data_ag_fast || wr_data_worker_adc>` denotes a 2-ary interference (or itf-2) that involves `rd_data_ag_fast` and `wr_data_worker_adc` transactions. Such interference occurs when tasks `ag_fast` and `worker_adc` attempt to load (resp. store) data at the same time in LMU0 accessed via the SRI, denoted by the interference channel `{ lmu0_load, lmu0_store, sri_load, sri_store }`. As the platform includes 6 cores, it is possible to compute up to 6-ary interferences: e.g., we count 8024 itf-2, 205142 itf-3, 3053207 itf-4 (see *Baseline* in Figure 4). LMU0 and PFLASH0, which are two of the main shared memories, and the SRI bus to access them are the main hardware components causing interferences, e.g., SRI is implicated in 6904 itf-2, LMU0 in 2589 itf-2 and PFLASH0 in 479 itf-2. I/Os and the FPI bus to access them are other components causing interferences (e.g., 1120 itf-2 for FPI, 480 itf-2 for I/Os).

b) Structural refinement: With the SRI description (*Structural refinement* in Figure 4), we observe that the number of interferences significantly decreases: -46% for itf-2, -66% for itf-3, -75% for itf-4, etc. Most of the interferences found in the SRI are eliminated, as it turns out that most of the memories serve a single core. Thus, only the SRI's ports that are used to access shared memories still experience interferences: o_9 connected to LMU0 (2589 itf-2) and o_2 connected to PFLASH0 (479 itf-2).

c) Routing: Routing has been specified for both the baseline model and the structural refinement model, i.e., with/without refinement of the SRI. In the first case (*BaselineRouting* in Figure 4), where SRI is described as a bus, we see that interference is reduced significantly: e.g., -19% for itf-2, -26% for itf-3 and -32% for itf-4. Many interferences are avoided due to transactions that are not counted in the SRI, e.g., we count -33% of itf-4 in the SRI. In the second case (*StructuralRouting* in Figure 4), we do not observe a reduction in the number of interferences itself, as the number of interferences is already reduced due to the description of the internal structure of the SRI. However, we can see the effect of routing on interference channels: conflicts now concentrate on PFLASH0 whereas interferences are reduced in the SRI, e.g., -26% of itf-4 in o_2 connected to PFLASH0.

d) Temporal refinement: As compared with structural refinement, we can see in Figure 4 (*StructuralRoutingTiming*) that temporal refinement discards e.g. 379 itf-2 (-9%), 17142 itf-3 (-25%) and 321775 itf-4 (-42%). For these scenarios,

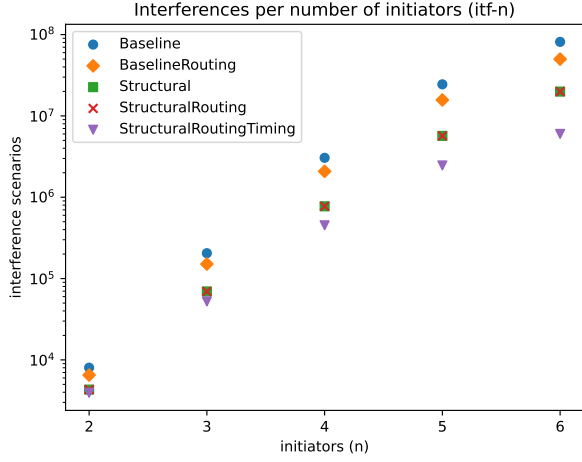
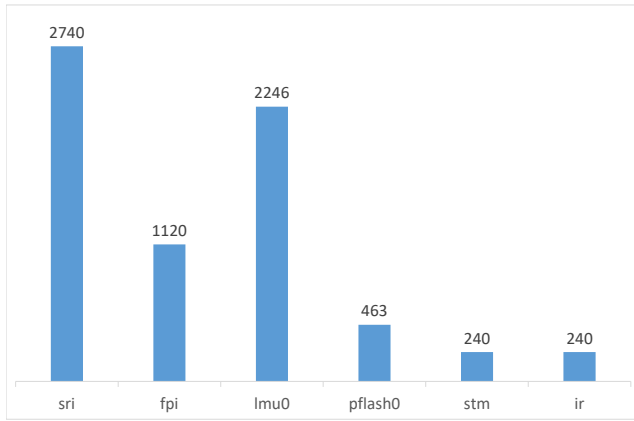
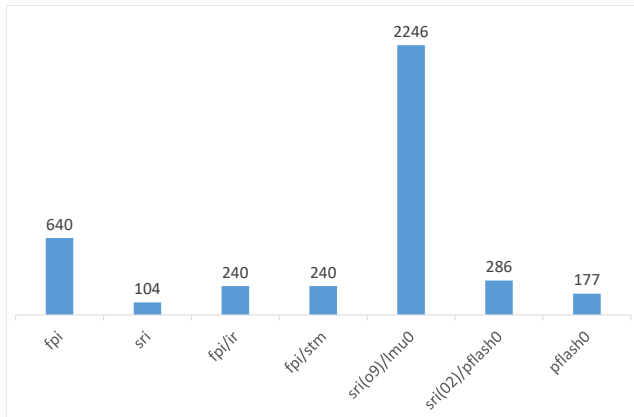


Fig. 4: Number of interference scenarios according to the number of initiators (itf- n) for different types of refinement (logarithmic scale).



(a) Itf-2 scenarios per hardware resource.



(b) Itf-2 scenarios per interference channel.

Fig. 5: Number of interference scenarios (itf-2) by (a) hardware resource, (b) interference channel.

exclusion of `ag_fast` and "workers" (`worker_gpio`, `worker_adc` and `worker_pwm`) transactions represents 52% of the total decrease, and exclusion of transactions between `ag_iams` and `ag_cpcs` 33%.

e) Summary: In total, 8976029 interference scenarios are found from the refined model: 3933 itf-2, 52272 itf-3, 451728 itf-4, 2460544 itf-5 and 6007552 itf-6. As compared to the baseline model, which displayed 109075037 interference scenarios, refinement enabled us to eliminate 92% (100099008) of interference scenarios : 51% of itf-2, 75% of itf-3, 85% of itf-4, 90% of itf-5, and 93% of itf-6. In terms of refinement steps, structural refinement, through the description of the internal structure of the SRI, has the greatest impact (83% of 100099008) before temporal refinement (17%), which specifies temporal exclusions.

We find different sources of interference (see e.g. Figure 5a for itf-2 scenarios) : memories with LMU0 and PFLASH0, inputs/outputs with STM and IR, and the buses to access them (SRI for memories and FPI for inputs/outputs). In terms of interference channels (Figure 5b for itf-2 scenarios), the SRI/LMU0 channel that is used by the cores to access the LMU0 shared memory is the main interference channel (e.g., 2246 itf-2 scenarios over 3933 itf-2 overall, that is to say 57% interference scenarios).

B. Execution Time Evaluation

After identifying the interference scenarios, we estimate execution times in order to check compliance with time budgets specified in the ASTERIOS application architecture (Equation (1)). Estimation of the execution time of a task is built on an analytical formulation of execution time that we compute with measures obtained on the target.

1) Analytical formulation of execution time: The evaluation of the execution time of a task must take into account all the interferences that the task may experience.

The execution time ET of a task t is given by:

$$ET(t) = ET_{iso}(t) + D_{itf}(t) \quad (2)$$

with ET_{iso} is the execution time in isolation (i.e., the execution time without contentions) and D_{itf} is the interference delay (i.e., the extra-time due to interferences). More precisely, the interference delay of task t is the time spent to access resources belonging to interference channels, which depends on the number of interfering accesses to every resource R in interference channels ($NA_{itf}(t, R)$) and the interfering time for each access ($DA_{itf}(t, R)$):

$$D_{itf}(t) = \sum_{\substack{R \in itf_channel(t) \\ NA_{itf}(t, R)}} DA_{itf}(t, R) \quad (3)$$

The worst-case execution time can therefore be expressed as the combination of the worst-case execution time in isolation

$WCET_{iso}$ and the worst-case interference delay WCD_{itf} :

$$WCET(t) = WCET_{iso}(t) + \underbrace{\sum_{R \in itf_channel(t)} WCD_{itf}(t, R)}_{WCD_{itf}(t)} \quad (4)$$

with the worst-case interference delay due to a shared resource R encompasses the worst-case number of interfering accesses to the resource $WCNA_{itf}$ and each access suffers a worst-case interference delay $WCDA_{itf}$:

$$WCD_{itf}(t, R) = WCNA_{itf}(t, R) \times WCDA_{itf}(t, R) \quad (5)$$

2) *Application*: For example, we apply Equation (4) in order to calculate worst-case execution times of `ag_fast` and `ag_slow` (parameters and results are summarized in Table I).

To do so, we first measure the execution time in isolation for each task using RVS¹ tool. For instance, Figure 6 describes the relative frequency of observed `ag_fast` execution times: observed BCET is 124 μ s, and observed WCET is 134.9 μ s. Therefore, we have $WCET_{iso}(\text{ag_fast}) = 134.9 \mu$ s

We then calculate the interference delay according to Equation (5). Access numbers (NA) are extracted from execution traces collected using TRACE32² tool. Figure 7 describes, for example, the frequency of accesses to LMU0 of `ag_fast` and `ag_slow`: the worst-case number of accesses (WCNA) is $WCNA(\text{ag_fast}, \text{LMU0}) = 1559$ and $WCNA(\text{ag_slow}, \text{LMU0}) = 4559$. In addition, hardware characterization allowed us to over-approximate access times, e.g., the maximum measured access time in contention was $WCDA_{itf}(\text{ag_fast}, \text{LMU0}) = 23.48$ ns (versus 7.53 ns in isolation).

The worst-case execution time is finally calculated according to Equation (4). The evaluation can be refined gradually, and the process stops when the calculated WCET meets the budgets (Equation (1)).

For example, as a first approximation (*level 1*), we may assume that each access to LMU0 results in an interference, i.e., for a task t , $WCNA_{itf}(t, \text{LMU0}) = WCNA(t, \text{LMU0})$ with $WCNA_{itf}$ is the worst-case number of interfering accesses and $WCNA$ is the worst-case number of accesses. Otherwise (*level 2*), we may calculate the worst-case number of interfering accesses to LMU0, i.e., for two tasks t_i, t_j accessing LMU0, $WCNA_{itf}(t_i, \text{LMU0}) = WCNA_{itf}(t_j, \text{LMU0}) = \min WCNA(t_i, \text{LMU0}), WCNA(t_j, \text{LMU0})$.

Evaluated WCETs are given in Table I together with the remaining margin to the budget. The stopping criterion is reached when the margin is positive (shown through green filled cells in the table) and unsuccessful when negative (orange filled cells). Therefore, we see that the stopping criteria is achieved by the first level of analysis for `ag_fast` and by the second level of analysis for `ag_slow`.

	ag_fast	ag_slow
Period (us)	500	2000
Budget (us)	250.56	300
BCET _{iso} (us)	124	221.5
WCET _{iso} (us)	134.9	224.7
level 1:		
WCNA _{itf}	1559	4559
WCD _{itf} (us)	36.6	107.0
WCET (us)	171.5	331.7
Margin with budget (us)	79.1	-31.7
Margin with budget (%)	31.6%	-10.6%
level 2:		
WCNA _{itf}	1559	1559
WCD _{itf} (us)	36.6	36.6
WCET (us)	171.5	261.3
Margin with budget (us)	79.1	38.7
Margin with budget (%)	31.6%	12.9%

TABLE I: Tasks parameters and estimation of execution times.

C. Validation

Validation of the model can be reached through tests performed on the real system. As the model is aimed at identifying interferences, validation can be achieved by checking that (i) when an interference is identified from the PML model, an interference can be observed on the real system, and (ii) when an absence of interference is identified no interference can be observed on the real system. Therefore, each interfering (resp. non-interfering) scenario is associated with a test scenario.

Remind that an interfering scenario involves transactions, initiated by tasks executing on different cores, contending for one or several (exclusive) service(s) provided by a same hardware component. Conversely, a non-interfering scenario is a scenario where transactions do not contend for services provided by hardware components.

In the following, we discuss the validation of scenarios with 2 transactions, this approach can be generalized to deal with scenarios with n transactions.

a) *Interference validation*: Let $\langle tr_A || tr_B \rangle$ an interference scenario, tr_A is initiated by $task_A$ hosted by $core_X$ and tr_B is initiated by $task_B$ hosted by $core_Y$ ($X \neq Y$), Res is the resource used to serve both tr_A and tr_B . To show that the interference scenario is valid, we can show that $task_A$ is sensitive to the service of tr_B by Res and that $task_B$ is sensitive to the service of tr_A by Res .

Consider for example interference scenarios in LMU0 involving `ag_fast`, executed on core 2, and `ag_slow`, executed on core 3. These interference scenarios can be validated by running `ag_fast` (resp. `ag_slow`) against co-runners on core 3 (resp. core 2) issuing transactions to LMU0. The objective of the co-runner is to act as the contending task, thus stressing the shared resource to maximize the bandwidth use and produce interferences. The expected outcome, if the interference scenario is valid, is an increase in the task execution time compared to the task behavior in isolation, as we can see for `ag_fast` in Figure 8 (`ag_fast` vs 1 contender lmu0).

b) *Non-interference validation*: Let $\langle tr_A || tr_B \rangle$ an interference scenario, tr_A is initiated by $task_A$ hosted by $core_X$

¹from Rapita Systems: <https://www.rapitasystems.com/products/rvs>

²from Lauterbach: <https://www.lauterbach.com/>

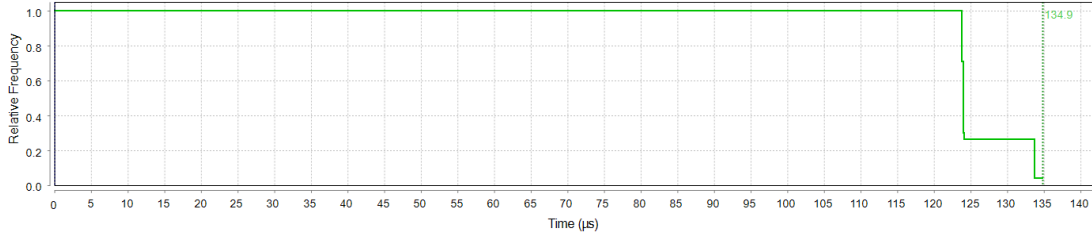
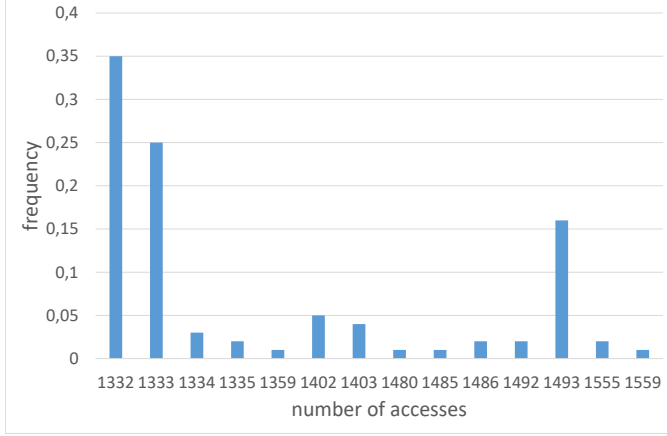
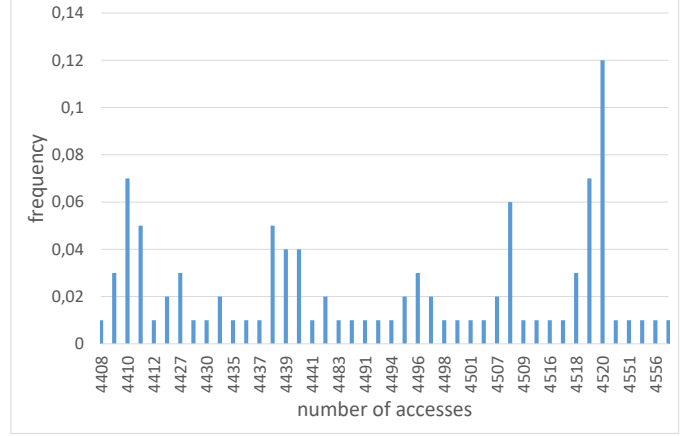


Fig. 6: Relative frequency of execution times of `ag_fast` (snapshot from RVS tool).



(a) `ag_fast`



(b) `ag_slow`

Fig. 7: Frequency of accesses to LMU0.

and tr_B is initiated by $task_B$ hosted by $core_Y$ ($X \neq Y$), Res_A is the resource that serves tr_A and Res_B is the resource that serves tr_B . To show that the interference scenario does not exist, we can show that $task_A$ is not sensitive to the service of tr_B by res_B and that $task_B$ is not sensitive to the service of tr_A by res_A .

Let us take the example of a non-interfering scenario `< rd_data_ag_fast || wr_data_ag_cpccs >`, `rd_data_ag_fast` is initiated by `ag_fast` (hosted by core 2) and targets LM0 while `wr_data_ag_cpccs` is initiated by `ag_cpccs` (hosted by core 4) and targets LMU1. The absence of interference can be shown by running `ag_fast` (resp. `ag_cpccs`) against co-runners on core 4 (resp. core 2) targeting LMU1 (resp. LMU0). The expected outcome, if the non-interfering scenario is valid, is no variation in the task execution time compared to the task behavior in isolation, as shown for `ag_fast` in Figure 8 (`ag_fast` vs 1 contender `lm1`, the slight variation is due to code instrumentation for measurement).

D. Summary

In this section, we have shown an application of our method (introduced in Section IV) to build a PML model. Refinement has been applied on the PML model with the objective to meet the budget constraint (execution times evaluated from the model must fulfill the budgets).

The process took place in 2 phases. We firstly applied model refinement (on the structural, routing and temporal views) in order to reduce interference scenarios to be later evaluated. In

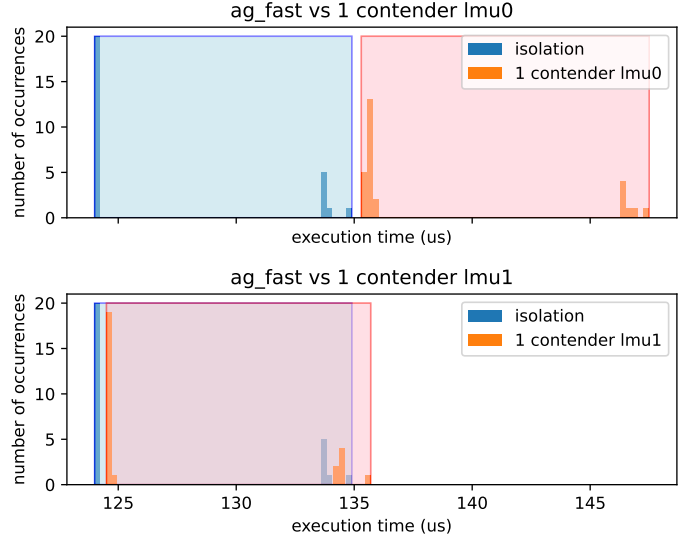


Fig. 8: Distribution of `ag_fast` execution times: in isolation (in blue) and vs contenders (in red).

the second phase, we evaluated execution times in order to check compliance with the budget constraint. The evaluation itself can be refined in order to minimize evaluation cost, e.g., we have shown two levels of evaluation based on measurement and the more precise (and costly) evaluation has only been applied when the first evaluation was not precise enough to conclude about the constraint. Validation of the model through tests has also been shown in order to confirm the modeling assumptions.

VII. DISCUSSION

This section provides some discussion on the refinement method presented in this paper. We also identify some limitations of the method and propose possible solutions.

a) Refinement path(s): In the previous section (Section VI), we have seen an example of application of our method. Note that if model refinement and evaluation were here performed sequentially, other paths could be possible. For example, the model could have been partially refined, then evaluated to check compliance with the constraint, refined again if necessary, re-evaluated, and so on until an abstraction that meet the constraint was found. Multiple paths are possible in practice, and choosing an "optimal" refinement path is based on practice and experience.

b) Tooling: The PML model captures data from various sources. Although the models described in this paper were done "by hand", a PML model could be automatically (or at least partially) built with the data extracted from design files, source code, configurations files, etc. For example, a script has been developed in order to extract temporal exclusions from the ASTERIOS's RSF. In a complementary way, the PML analyzer has also been extended in order to calculate metrics (e.g., interferences for each component) that can be used to drive the refinement process.

c) Reachability: When building a PML model, it may be the case that reaching the solution is not possible (e.g., modeling or evaluation effort is too high) or that this solution does not even exist (because the proposed design does not meet the constraint). In these cases, it is necessary to mitigate interferences (re-design the system) and model the system again. The results provided by the PML analyzer (list of interferences, interference channels) can be used in this task and the new design elements can be incorporated in the PML model.

d) Validation: Validation of the model is an important issue. For example, in Section VI-C, we proposed to address validation through tests: tests are used to corroborate interfering/non-interfering scenarios on the real system. However, this validation strategy is biased as it only enables to validate the elements that are captured in the model: it is thus possible to converge towards an incomplete model. A way to solve this problem would be to complete the validation strategy with benchmarks specifically devised to exhibit the common micro-architectural mechanisms that may be undocumented.

VIII. CONCLUSION

This paper dealt with the interference problem in multicore processor embedded systems. We extended the PHYLOG approach with a method to use the PML language. This method relies on different types of refinements (structural, routing, temporal, etc.) in order to build a precise and reliable model of the system and then perform interference analysis (i.e., identify interferences and evaluate their effects) in a sound way. We showed an application of this method on an industrial use case coming from the aerospace domain.

Future work could develop validation and evaluation aspects, investigate tools to help to build PML models such as Large Language Models (LLM) to deal with large datasheets, or integrate PML with ASTERIOS or Prelude [17] development toolchains to build interference-aware real-time applications.

ACKNOWLEDGMENT

This work has been done as part of the ARCHEOCS project funded by the French Research Agency (ANR) and the partners of the IRT Saint-Exupéry Scientific Cooperation Foundation and from the PHYLOG 2 project funded by the France Relance program and the European Union through the NextGenerationEU program. The authors would also like to thank ASTERIOS Technologies, especially Guillaume Phavorin, and Rapita Systems for their support in the use of ASTERIOS and measurement tools.

REFERENCES

- [1] "General Acceptable Means of Compliance for Airworthiness of Products, Parts and Appliances (AMC-20)," European Union Aviation Safety Agency, Tech. Rep. AMC-20, Amendment 23, 2022.
- [2] W.-T. Sun, E. Jenn, H. Cassé, and T. Carle, "Automatic Identification of Timing Interferences on Multi-Core Processor in a Model-Based Approach," in *Conférence d'informatique en Parallélisme, Architecture et Système*, 2019.
- [3] V. A. Nguyen, E. Jenn, W. Serwe, F. Lang, and R. Mateescu, "Using Model Checking to Identify Timing Interferences on Multicore Processors," in *Embedded Real Time Software and System Conference (ERTS'20)*, 2020.
- [4] F. Boniol, Y. Bouchebaba, J. Brunel, K. Delmas, T. Loquen, A. M. Gonzalez, C. Pagetti, T. Polacsek, and N. Sensfelder, "PHYLOG Certification Methodology: a Sane Way to Embed Multi-Core Processors," in *Embedded Real Time Software and Systems (ERTS 2020)*, 2020.
- [5] X. Jean, L. H. Mutuel, and R. Soulat, "Assurance of Multicore Processors: Limits on Interference Analysis," Federal Aviation Administration, Tech. Rep. DOT/FAA/TC-19/24, Mar. 2020.
- [6] "Multi-Core Processors Position Paper," Certification Authorities Software Team, Tech. Rep. CAST-32A, Nov. 2016.
- [7] D. Dasari and V. Nelis, "An Analysis of the Impact of Bus Contention on the WCET in Multicores," in *IEEE International Conference on High Performance Computing and Communication & IEEE International Conference on Embedded Software and Systems*, 2012.
- [8] 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.
- [9] S. Girbal, J. Bin, D. Gracia Perez, and A. Merigot, "Using Monitors to Predict Co-Running Safety-Critical Hard Real-Time Benchmark Behavior," in *Conference on Information and Communication Technology for Embedded Systems*, 2014.
- [10] V. Brindejonc and A. Roger, "Avoidance of Dysfunctional Behaviour of Complex COTS used in an Aeronautical Context," in *Congrès de Maîtrise des Risques et Sûreté de Fonctionnement*, 2014.
- [11] "Infineon AURIX TC3xx User Manual Part1," infineon, Tech. Rep. (V2.0.0), 2021.
- [12] V. David, A. Barbot, and D. Chabrol, "Dependable Real-Time Systems and Mixed-Criticality: Seeking Safety, Flexibility and Efficiency with ASTERIOS," *Krono-Safe*, Tech. Rep., 2017.
- [13] V. David, J. Delcoigne, E. Leret, A. Ourghanlian, P. Hilsenkopf, and P. Paris, "Safety Properties Ensured by the OASIS Model for Safety Critical Real-Time Systems," in *Computer Safety, Reliability and Security*, 1998.
- [14] F. Siron, D. Potop-Butucaru, R. de Simone, D. Chabrol, and A. Methni, "The Synchronous Logical Execution Time Paradigm," in *Embedded Real Time Software and Systems (ERTS2022)*, 2022.
- [15] Mutuel, Laurence H., X. Jean, V. Brindejonc, A. Roger, T. Megel, and E. Alepins, "Assurance of Multicore Processors in Airborne Systems," Tech. Rep. DOT/FAA/TC-16/51, 2017.
- [16] S. Bayless, N. Bayless, H. Hoos, and A. Hu, "SAT Modulo Monotonic Theories," in *AAAI Conference on Artificial Intelligence*, 2015.
- [17] C. Pagetti, J. Forget, F. Boniol, M. Cordovilla, and D. Lesens, "Multi-task Implementation of Multi-periodic Synchronous Programs," *Discrete Event Dynamic Systems*, vol. 21, no. 3, pp. 307–338, Sep. 2011.

Kryptonite++: Localizing Program Interference on Multi-core Embedded Systems

Nikhilesh Singh[†], Karthikeyan Renganathan[†], Chester Rebeiro[†], Jithin Jose[§], Ralph Mader[§]
[†]Indian Institute of Technology Madras, India, [§]Vitesco Technologies

Abstract—To address cost and energy constraints, various embedded system applications are increasingly adopting multi-core hardware. In applications that require real-time behavior, it is important to estimate the worst-case interference from other programs executing on the multi-core platform. Further, reducing the interference requires the knowledge of program regions that are most susceptible to interference. Existing solutions tend to overestimate the interference, while pinpointing interference hotspots remains an open problem.

In this paper, we present Kryptonite++, a framework to synthesize an environment for a given program on a multi-core embedded processor to estimate the worst-case program interference. Kryptonite++ builds this interfering environment using small code gadgets that are designed to hammer specific hardware modules. To arrange these gadgets, we use a greedy approach followed by a Reinforcement Learning algorithm. Kryptonite++ finally analyzes the interference patterns and the executed instructions to pinpoint the hotspots of interference in programs. We demonstrate Kryptonite++ on the automotive grade Infineon AURIX TC399 processor with a wide range of programs.

Index Terms—Multi-Core Embedded Systems, Worst-Case Program Interference, Real-time Safety, Reinforcement Learning.

I. INTRODUCTION

Cyber-physical systems have proliferated to a wide range of domains such as automotives [8], defense and medicine [1]. In recent years, due to financial and energy constraints, such systems are increasingly adopting multi-core processors. However, this transition introduces serious safety and security challenges, especially in applications that need real-time guarantees. For these applications, it is critical to determine the worst-case execution time (WCET) to ensure that the execution adheres to deadlines. The knowledge of WCET also aids in the optimal placement of program data in the presence of multiple independent memory modules in the hardware. Existing literature includes several solutions to precisely determine the WCET of programs on single-core processors [21]. However, the precise determination of WCET for a program executing on a complex multi-core processor remains an open problem and is provably NP-Hard [11].

The key factors affecting the WCET of a program on multi-core processors are (a) the inputs to the program and (b) the interference from other simultaneously executing programs. The program inputs determine the flow of instructions to be executed, influencing the execution time. On the other hand, simultaneously executing programs contend for shared hardware resources, resulting in the interference. For example, consider

two programs sharing a cache set. These programs can evict each other's cache lines, which increases the occurrences of cache misses, in turn, increasing the execution time of the programs.

While single-core WCET tools [21] can be used to evaluate the effects of the program inputs, the critical challenge lies in determining the Worst-Case Program Interference (WCPI) due to simultaneously executing programs. Most contemporary solutions statically analyze the control flow of the program under test (PUT) using an abstract model of the hardware to estimate interference [10, 11, 13, 15, 20]. These approaches, however, have two major limitations. First, the abstract model of the micro-architecture is typically restricted to a shared cache memory and the system bus [10, 11, 13, 15, 20]. However, typical multi-core systems are far more complex. There could be multiple memories, buses, and peripherals. Each of these units has different access latencies, which also depend on which core the program is executing (for example, Non-Uniform Memory Access [9]). Formally capturing such complex behavior is difficult. A second limitation of formal modeling is that analysts would need to completely know the micro-architectural details. However, not all details are documented, leading to oversimplified models. Thus most contemporary works tend to overestimate the worst-case values [21], resulting in imprecise estimates of interference, hampering WCPI analysis.

An orthogonal approach for WCPI determination is to estimate the worst-case interference as presented in Kryptonite [19]. Unlike existing works that formally model the multi-core architecture, Kryptonite performs dynamic analysis on the hardware to identify an environment that estimates the worst-case interference for a Program Under Test (PUT). Such an approach has two key advantages. First, Kryptonite can precisely capture the interactions between micro-architectural components and the PUT, as there is no abstraction involved. Second, it does not require tiny details of the hardware, which are often proprietary.

In a typical application, the access to resources need not be uniform [18]. Thus, some regions of code in the PUT can be more susceptible to interference than others. However, the WCPI estimated by Kryptonite is for the entire PUT and does not provide any fine-grained information about specific regions of high interference. Pinpointing these hotspots would provide deep insights about the causes of inference and help application development and testing. In this paper, we extend Kryptonite to provide (a) the currently synthesized

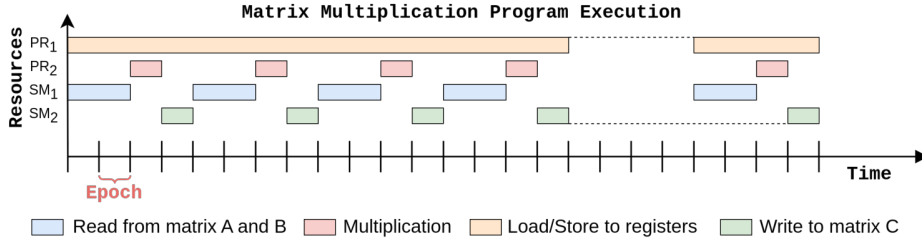


Fig. 1: The access pattern of a program performing matrix multiplication with shared modules SM_1 and SM_2 , and private modules PR_1 PR_2 . To aid in pinpointing the interference hotspots, Kryptonite++ segments the executions into different epochs.

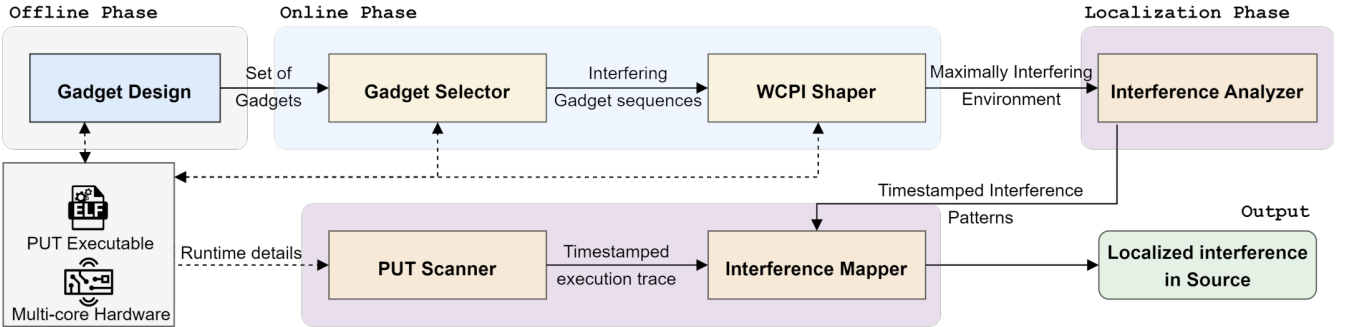


Fig. 2: The Kryptonite++ framework. It works in three phases, namely, offline, online, and localization. The offline phase uses the hardware specifications to design gadgets that can hammer different shared modules. The online phase uses these gadgets as the building blocks to synthesize the maximally interfering environment by performing runtime measurements for the program under test (PUT). First, the Gadget Selector designs interfering gadget sequences which are then fine-tuned to generate the final output using Reinforcement Learning by the WCPI Shaper. In the localization phase, the Interference Analyzer segments the interference patterns into small time periods called epochs, while the PUT Scanner segments the PUT instructions in the corresponding epochs. Finally, the Interference Mapper collates the per-epoch observations to output the hotspots of interference in the PUT in the presence of the maximally interfering environment.

WCPI environment for a given PUT, and (b) the localized information about the interference behavior during the runtime of the PUT. Further, we map the regions of high interference to the source code, which can be used by the development team to introduce modifications reducing susceptibility to interference. Following are our major contributions.

- We present Kryptonite++, a dynamic analysis framework to estimate the Worst-Case Program Interference (WCPI), and further locate the hotspots of interference for PUT executing on multi-core hardware.
- Kryptonite++ works by building an environment that induces an estimation of the worst-case interference on the PUT using specialized code snippets, called *gadgets*, that hammer specific hardware modules.
- The gadgets are selected and organized using a greedy algorithm and Reinforcement Learning (RL). Further, the PUT execution and the maximally interfering environment are segmented into *epochs*. Kryptonite++ maps the instructions executed by the PUT to interference patterns observed in the corresponding epoch to identify the hotspots of interference in different regions of the PUT.
- We demonstrate the working of Kryptonite++ on popular

automotive hardware, TriBoard v2.0 hardware with a 6-core Infineon AURIX TC399 microcontroller on a wide range of applications including AES and the Mälardalen benchmark suite [5].

II. BACKGROUND

Multi-core processors are composed of hardware modules that are either private to each CPU core or shared across cores. Programs executing on a CPU core access private and shared modules at different time instances during their run time. For example, Fig. 1 describes the run time access patterns for a matrix multiplication program where the input matrices are stored in shared memory unit SM_1 while the result is stored in another shared memory unit SM_2 . During execution, the program uses private hardware resources PR_1 and PR_2 to store intermediate results and perform ALU operations. The access patterns to the shared and private hardware resources are different and are characteristic of the program. The WCET for such a program is influenced by the (a) program inputs and (b) the interference that occurs due to other programs executing on the multi-core processor. Of the two factors that decide WCET, estimating the Worst-Case Program Interference (WCPI) is a challenge. The interference can only occur due to the shared modules that are present (*i.e.* SM_1 and SM_2), and not the

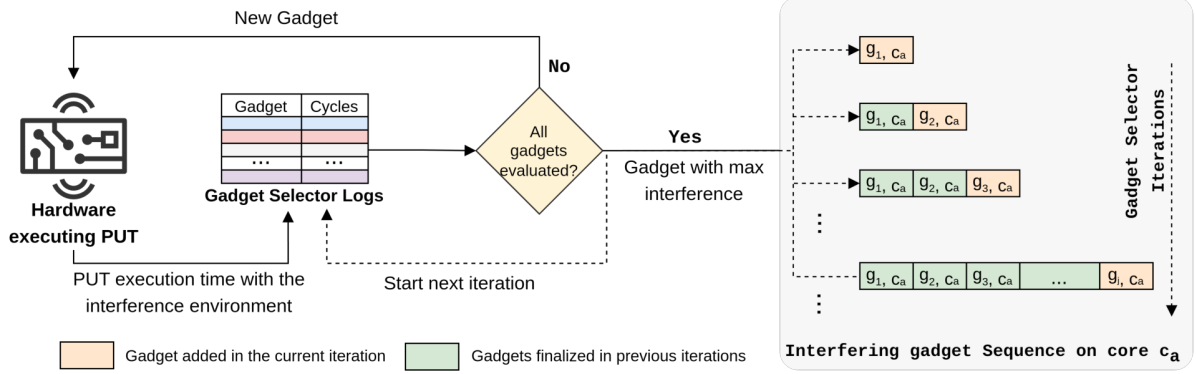


Fig. 3: The Gadget Selector. On every CPU core, each iteration evaluates all the gadgets and adds the gadget that results in the highest execution time of the PUT to the interfering environment. After each addition, the Gadget Selector starts the next iteration, ultimately crafting a gadget sequence.

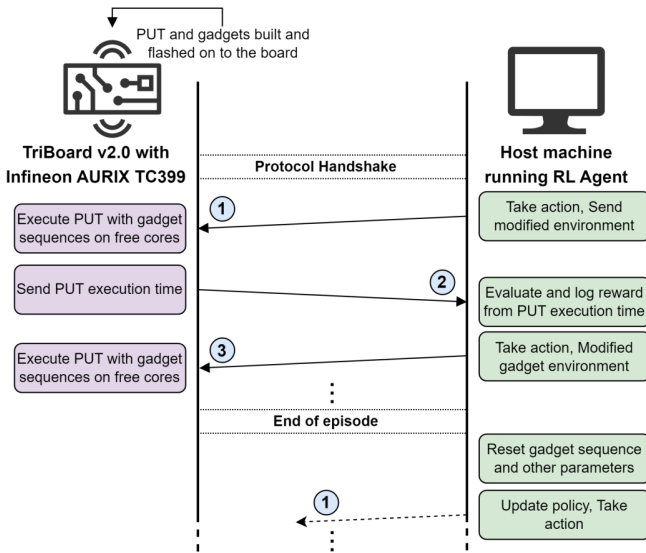


Fig. 4: The hardware-in-the-loop setup used by Kryptonite++ to train the Reinforcement Learning agent in the WCPI Shaper.

private modules (*i.e.* PR_1 and PR_2). The rest of the paper discusses Kryptonite++, which is designed to estimate the WCPI of a program and identify the hotspots of interference.

III. THE KRYPTONITE++ FRAMEWORK

In this section, we discuss different components of the Kryptonite++ framework, as shown in Fig. 2, used to design the maximally interfering environment on the free CPU cores and locate the interference hotspots. Consider a multi-core system with a set of n processors executing the Program Under Test (PUT). During execution, the PUT accesses different shared and private hardware modules in the system at different instances during its execution (Fig. 1). If the PUT executes on $(n - m)$ CPU cores, such that $n > m$, then assuming no preemption, interference can be caused by

the programs that execute on the remaining m CPU cores and utilize the shared hardware modules. We denote these free CPU cores by the set $\mathcal{C} = \{c_0, c_1, c_2, c_3, \dots, c_{m-1}\}$. In our experiments, we primarily use PUTs that execute on one CPU core ($m = (n - 1)$).

A. Gadget Set Design

Listing 1: A generic structure of a gadget.

```

void a_generic_gadget(int iterations){
  for(i=0; i< iterations; i++){
    operation #1;
    operation #2;
    ...
    operation #n;
  }
}

```

We define a *unit* as the smallest entity within a shared module in a multi-core processor that can independently cause contention. Each unit in the hardware can support multiple operations. For instance, a cache set can be defined as a unit with load and store as the possible operations. Fundamental to Kryptonite++ are small pieces of code called gadgets, designed to hammer one or more such units that are within a shared module or across modules. The hammering is done for a configurable duration represented by the *iterations* parameter. For example, in a shared cache memory, the smallest unit that can cause contention is a cache set. Thus, we design gadgets to hammer one or more cache sets using different combinations of the load and store applications. Listing 1 describes the generic structure of a gadget with n operations.

Similarly, Listing 2 describes an example of a gadget targeting the load operation in the 15-th (0xe) cache set in a 4-way set associative cache. The four load instructions executed in a loop hammer the cache set, creating contention with other programs that share the same cache set. In our evaluation, we typically begin with *iterations* set to 64 for each gadget. This empirically selected value ensures that the overheads due to

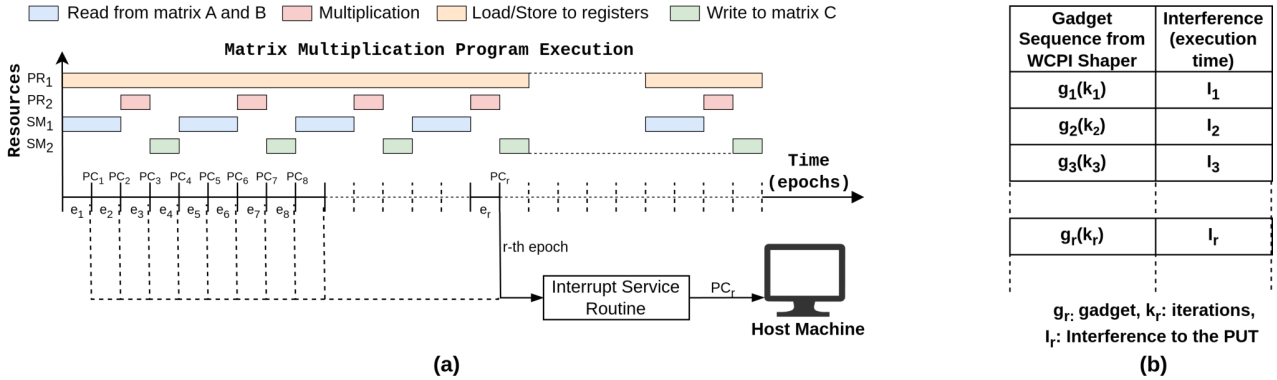


Fig. 5: (a) Introducing timestamps in PUT execution using interrupts at the end of each epoch. At the end of the r -th epoch, interrupts are triggered to all the CPU cores, pausing the PUT as well as the gadget execution. The Interrupt Service Routine (ISR) then collects the program counter (PC_r) value from the CPU core executing the PUT and sends it to the host machine. Finally, the execution of the PUT and the gadgets resumes in synchronization for the next epoch. (b) Different gadgets in the sequence on a CPU core with their corresponding interference on the PUT. The Interference Mapper converts the duration of each gadget’s execution into epochs to map the interference behavior with different sections of the PUT.

non-interfering instructions (such as loop updates or function calls) are kept to a minimum while ensuring that the fine-grained execution phases of the PUT are captured.

Listing 2: An example gadget targetting the cache set mapped by the 4 set bits 0xe (bits 9-12 from LSB) looping for iterations times.

```

void cache_set_gadget(int iterations){
    for(i=0; i< iterations; i++){
        // the set bits are 0xe
        load 0xdeafbeef;
        load 0x00ff0ead;
        load 0xfffffe00;
        load 0x12345eba;
    }
}

```

B. Gadget Selector

Given a Gadget Set \mathcal{G} determined using the techniques in the previous section and a set of free CPU cores \mathcal{C} in the hardware, the goal of the Gadget Selector is to design an environment that maximizes the interference with the PUT. This environment comprises the set of gadget sequences $\mathcal{L}(c_i)$ for each CPU core $c_i \in \mathcal{C}$ so as to maximize the PUT execution time. We construct such a sequence of gadgets on each core using an iterative greedy approach, as shown in Fig. 3. Initially, all gadget sequences, $\mathcal{L}(c_i)$, $c_i \in \mathcal{C}$, are empty. For the addition of the j -th gadget on a CPU core, all other cores must have at least $(j - 1)$ gadgets. Upon ensuring this, the order of CPU cores in which the j -th gadget is added to the corresponding gadget sequence is selected randomly. This approach ensures an independent distribution of gadgets on each CPU core. To add the j -th gadget in the gadget sequence $\mathcal{L}(c_i)$, Kryptonite++ identifies the gadget that, when added to the j -th position in the gadget sequence, results

in the maximum execution time of the PUT with a given environment.

The final interfering environment from the Gadget Selector has two major limitations. First, in this step, the duration of execution for each gadget is not optimized. This is done to reduce the search space for the greedy algorithm. Instead, we empirically fix a value for the number of iterations and use it for every gadget in \mathcal{G} . Thus, the duration of a gadget can exceed or fall short of the ideal duration, preventing the maximization of interference. Second, the interfering environment is limited to a finite set of gadgets, which can be inadequate for some execution patterns of a PUT. We address these limitations in the next component, called the WCPI Shaper.

C. WCPI Shaper

The WCPI Shaper addresses the limitations of the Gadget Selector and fine-tunes the interfering gadget sequence to provide the maximally interfering (WCPI) environment using a Reinforcement Learning (RL) algorithm. The algorithm uses an *agent* that learns the optimal series of modifications to the interfering environment over several timesteps so as to maximize the interference.

At timestep t the RL agent observes the interference environment formed by gadget sequences, represented as state S_t . The agent performs an *action* $a_t = (c_i, j, move)$ at timestep t , which performs a modification *move* on the j -th gadget on core c_i . Thus, the action a_t changes the state of the agent from state S_t to S_{t+1} , by either changing a gadget in the sequence or the duration of execution of the gadget. The PUT is executed with the modified environment, and based on its execution time, the agent accrues a scalar value called *reward* that quantifies feedback about the action a_t when in state S_t . Over several steps, as shown in Fig. 4, the agent learns the optimal sequence of actions that maximize the rewards, and in turn, the interference.

D. Interference Analyzer

The maximally interfering environment from the WCPI Shaper applies to the entire PUT and does not provide fine-grained information about hotspots of interference in the PUT. The identification of these hotspots requires two key pieces of information: **(a)** the interference patterns during the runtime of the PUT, and **(b)** the instructions executed by the PUT corresponding to those interference patterns. First, the Interference Analyzer executes the maximally interfering gadget sequence with the PUT to determine the variations in interference during the runtime of the PUT. To this end, we divide the PUT runtime into *epochs*, which can be defined as a minimum time period for observation. In each epoch, the rate of increase in the execution time of the PUT represents the effective interference incurred by the PUT. An epoch with a larger increase in the execution time implies that the instructions executing during the epoch are more susceptible to interference.

Once the interference pattern is determined, the next challenge is to identify the specific instructions of the PUT executing during each epoch. We address this by designing periodic interrupts that are configured to trigger at the end of every epoch, as shown in Fig. 5. These timer-based interrupts get triggered on all CPU cores, pausing the execution of the PUT as well as the gadgets. On the CPU core running the PUT, the interrupt service routine (ISR) logs the program counter (PC) value, which is the address of the last instruction in the PUT before the interrupt. This address is transmitted to the host machine from the hardware board. The ISR on all the other CPU cores wait until the transmission is complete, and thus the resumption of the execution of the PUT and the gadgets is synchronized. Similarly, the PC is logged for each subsequent epoch, and as a result, we get a timestamped execution trace of the PUT along with timestamped variations in the interference.

E. PUT Scanner and Interference Mapper

The goal of the PUT Scanner is to trace the logged instruction addresses of the PUT back to code regions. We achieve this by parsing the disassembly of the given PUT and dividing it based on the logged addresses. The final challenge in Kryptonite++ is mapping the interference patterns with these code regions of the PUT. While the trace of the PUT is uniformly divided into the duration of an epoch, the duration of each interference pattern depends on the corresponding gadget and the associated number of iterations, as shown in Fig. 5. The Interference Mapper resolves this issue by measuring the execution time of all the gadgets in the sequence. We use these timing measurements to identify the gadgets executing in each epoch. This enables the mapping of the interference patterns to corresponding epochs and, in turn, to the PUT's code region. Thus, along with the WCPI estimate of the PUT and a realizable interfering environment, Kryptonite++ also identifies the hotspots of interference in the PUT.

IV. IMPLEMENTATION

We perform the evaluation of Kryptonite++ on a setup including a host machine (Windows 10, Intel i7-7700) running different components of Kryptonite++. The host is connected to the TriBoard v2.0 with an Infineon AURIX TC399 microcontroller through a UART port. Fig. 6 presents an overview of the relevant units present in the Infineon AURIX TC399 board. Prior to the start of the online phase, we build all the gadgets in \mathcal{G} using the HighTec Free Entry toolchain that uses the *tricore-gcc v4.9.3.0* compiler. We flash the PUT and all the gadgets on the evaluation board using the Miniwiggler debugger. Each core has a dedicated CCNT register that records the elapsed CPU clock cycles. The register can be accessed at specific addresses in the memory. We read the value of the CCNT register at the start and the end of the PUT's execution to determine its execution time.

We also establish a UART-based communication protocol between the Infineon AURIX TC399 microcontroller board and the host machine. After initialization, the Gadget Selector or the WCPI Shaper can specify a sequence of gadgets to execute on each of the CPU cores along with the PUT. Fig. 4, for example, describes the protocol for communication between the WCPI shaper and the Triboard through UART. The execution time for the PUT is measured at the board and transmitted to the host machine. The RL agent in the WCPI Shaper is implemented based on the Proximal Policy Optimization (PPO) [17], which is a model-free algorithm, i.e., it does not make any assumptions about the underlying hardware or the PUT.

The interrupts on the Infineon TC399 board are implemented using the drivers in AURIX Development Studio. The Infineon board maintains different stacks for the ISR and the regular program execution. The Interference Analyzer collects the program counter values from the PUT executing core using the PCXI register, which contains the address of the previous execution stack. The collected addresses are transmitted to the host machine using the UART communication channel. The duration of each epoch to trigger an interrupt is $5\mu s$ which is 1500 clock cycles on the Infineon AURIX TC399 board.

V. EVALUATION AND RESULTS

Listing 3: An example PUT that performs AES encryption with T-tables present in different memory units of the Infineon AURIX TC399 board.

```
Ifx_Cpu_Setup(); // set the required data;
AES_Exp_Key_LMU0(); // key expansion, with
    T-tables in LMU0
AES_Exp_Key_dLMU0();
AES_Exp_Key_dLMU1();
AES_Exp_Key_LMU1();
AES_perform_LMU0(); // Encryption and
    Decryption, with T-tables in LMU0
AES_perform_dLMU0();
AES_perform_dLMU1();
AES_perform_LMU1();
```

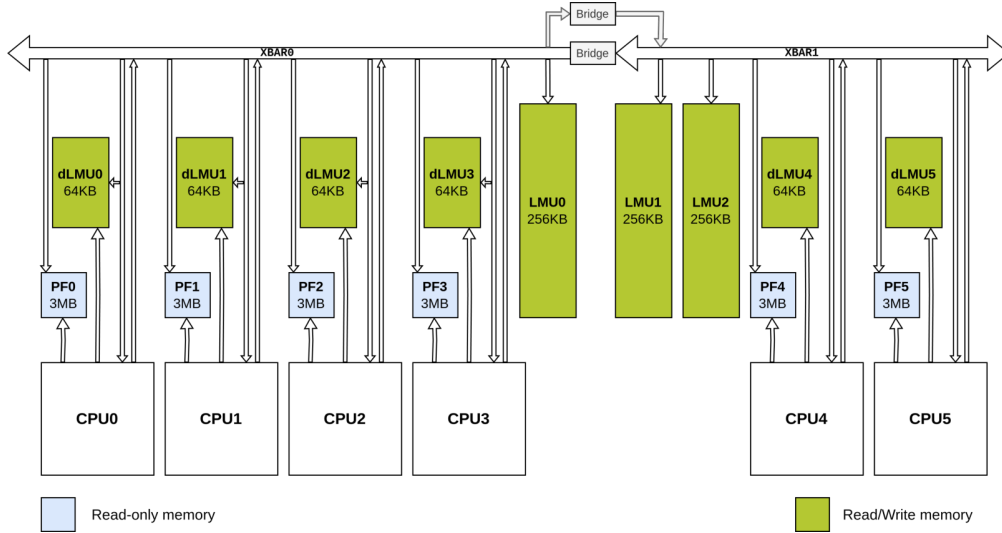


Fig. 6: Relevant layout section of Infineon AURIX TC399 microcontroller on the TriBoard hardware used for evaluation in Kryptonite++. The shared modules consist of six data Local Memory Units (dLMU), three Local Memory Units (LMU), and six Program Flash (PF) modules. Considering one operation in the gadget loop (Listing 1), we have 24 gadgets: read/write gadgets for each dLMU (6×2), read/write gadgets for each LMU (3×2), and read gadgets for each PF (6×1). Similarly, considering a read operation followed by a write in the gadget loop, we have 135 gadgets: 15 possibilities for the read operation (6 dLMUs, 3 LMUs, and 6 PFs) and 9 possibilities for the write operation (6 dLMUs and 3 LMUs).

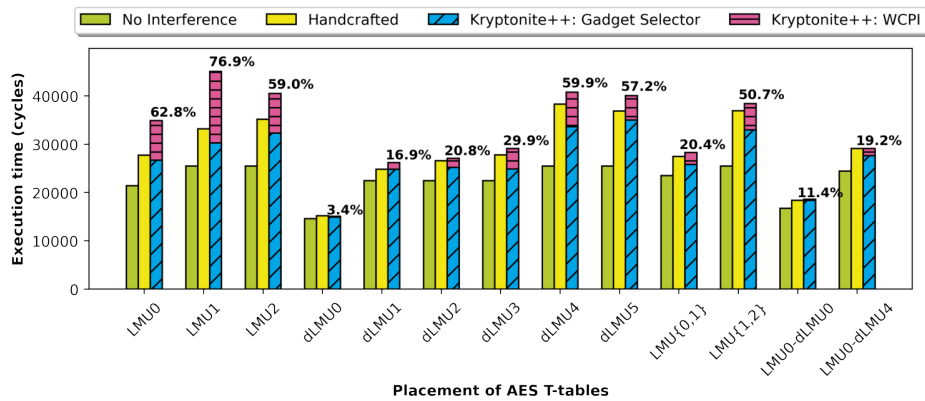


Fig. 7: The execution time (in clock cycles) of a PUT running AES encryption with different locations of the T-tables on the Infineon AURIX TC399 microcontroller. The bars compare the execution time in four cases: no interference to the PUT, interference with an environment handcrafted with the knowledge of the PUT, interference with the Gadget Selector output in Kryptonite++, and interference with final output from WCPI Shaper in Kryptonite++. The numbers on the top of the bars represent the percentage increase in PUT’s execution time with the maximally interfering environment given by Kryptonite++.

Fig. 7 shows the maximal interference for different implementations of AES using the Gadget Selector and the WCPI Shaper. The implementations vary in their placement of AES T-tables across different memory units shown in Fig. 6. To benchmark the WCPI estimates, we design handcrafted gadget sequences, with complete knowledge of the AES program and its data access patterns. We observe that Kryptonite++ consistently outperforms a handcrafted arrangement of gadgets which assumes the knowledge of the PUT internals. We observe that the LMUs have a higher interference compared

to other memory units. Further, the interference when the T-tables are in the dLMU (dLMU0) associated with the CPU core executing the PUT (CPU0) is minimal, as such requests are serviced via a dedicated bus (Fig. 6).

To discuss the localization of interference, let us consider the PUT described in Listing 3. This PUT consists of an AES implementation with various locations of the T-tables. The `Ifx_Cpu_Setup()` function ensures the placement of data in associated memory units and sets up the UART communication channel. This is followed by the key expansion functions

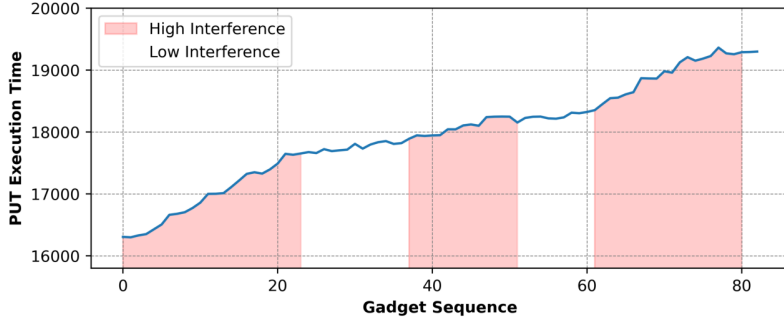


Fig. 8: The execution of the PUT with respect to gadgets in the WCPI gadget sequence. For example, the PUT execution time corresponding to the i -th value on the X-axis represents the interference to the PUT with the first i gadgets in the WCPI gadget sequence. We observe that the general trend of the curve is monotonically increasing, but there are variations in the slope when observed closely. This is because, though the interference to the PUT generally increases with more gadgets, the distribution of the interference is not uniform across the PUT’s execution. This observation implies that some sections of the PUT are more susceptible to interference than others. Further, there can be mild disturbances due to the inherent noise in execution time measurements.

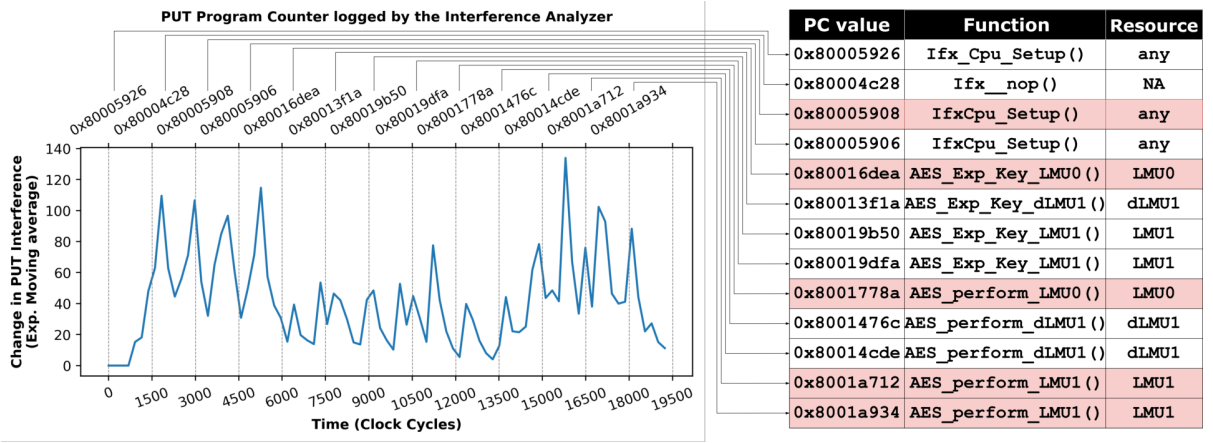


Fig. 9: A closer look at the distribution of interference shown in Fig. 8. The exponential moving average of the change in the execution time of the PUT with runtime (in clock cycles). Timer interrupts are triggered by the Interference Analyzer every epoch (1500 clock cycles), and the program counter associated with the PUT is logged. The output of the PUT scanner maps the logged instruction addresses to functions in the PUT.

for different implementations of AES with the T-tables present in LMU0, dLMU0, dLMU1, and LMU1, respectively. Finally, the PUT performs a set of AES encryption and decryption instances using T-tables present in the given memory units.

Without any interference, this PUT takes 16305 clock cycles. With the WCPI environment from Kryptonite++, the execution time increases by 18.4% to 19298. However, the distribution of interference is not uniform during the execution of the PUT, as shown in Fig. 8. For instance, till the 23rd gadget, the addition of each gadget increases the PUT’s execution time at a constant rate to 17652 clock cycles. After this, the execution time of the PUT increases quite slowly with the addition of newer gadgets for a while and then again increases sharply (shaded red in Fig. 8).

With the interference pattern of the PUT derived from Fig. 8, the Interference Analyzer now executes the PUT with

the WCPI environment with interrupts to log the program counter values at every epoch of 1500 clock cycles. Fig. 9 presents the exponential moving average (EMA) of the change in the execution time of PUT with runtime, given as

$$EMA_p = (1 - \alpha)EMA_{p-1} + \alpha\delta_p ,$$

where α is a smoothing factor ($\alpha = 0.5$) and δ_p is the increase in PUT execution time due to the p -th gadget in Fig. 8. We represent the runtime in clock cycles elapsed in place of the number of gadgets in the sequence executing simultaneously with the PUT. After an epoch, an increase in this moving average implies that the PUT section executing during the epoch incurs a high interference. On the other hand, a decrease in the moving average indicates a lower susceptibility of the associated PUT region to interference.

The top axis of Fig. 9 gives the program counter value for the CPU core executing the PUT as the timer interrupts occur every epoch. These values are instruction addresses in the PUT executable. The PUT Scanner parses the PUT disassembly and outputs the associated function within which the instruction with the given address is located. Thus, the interference behavior during an epoch can be mapped to a function in the PUT. We observe that the functions utilizing LMUs are the hotspots of interference for the given PUT. In contrast, the functions associated with dLMUs, and in particular dLMU0, have minimal susceptibility to interference. With access to the source code, the Interference Mapper can further provide the exact lines in the source code with high interference. For example, in the given PUT, line number 871-1094, which coincides with the function `AES_perform_LMU1()`, are deemed as hotspots of interference by Kryptonite++.

To ensure adaptability, we also test Kryptonite++ with the Mälardalen [5] benchmark suite, which consists of popular benchmarking applications in the real-time safety community. Table I presents the WCPI estimate for several benchmarks using the LMUs for data along with the functions identified as hotspots of interference by Kryptonite++. We observe that programs such as *bbsort* and *compress*, which are memory intensive, face more than 50% increase in the execution time. The interference hotspots are functions such as `BubbleSort()`, which performs bubble sort on arrays in LMUs, and `write-bytes()`, which writes to arrays in the LMUs involved with data compression. On the other hand, programs such as *fft1* (Fast Fourier Transform) are compute-heavy, resulting in minimal interference and no hotspot functions.

VI. RELATED WORK

Existing works attempt to determine WCPI by statically modeling the PUT and the interference in the multi-core platform [15, 10, 11, 13, 20] or estimating WCPI by creating interfering environments [2, 3, 4, 6, 12, 16]. While these solutions provide different techniques to determine WCPI, none of them can pinpoint the different regions in the PUT resulting in interference. Table II compares these state-of-the-art solutions with Kryptonite++.

Static approaches determine upper bounds for interference using statistical methods [13, 20], algebraic techniques [15, 10, 11] and machine learning [2, 4]. These bounds provide formal guarantees and can be tuned for different access patterns. However, statically determined WCPI bounds are low in precision as they are prone to overestimation [15, 21, 12]. Further, these solutions are limited in scale as they consider an abstract representation of the hardware, which is difficult to model for multi-core systems. It also requires complete micro-architectural details, which are not always available.

An orthogonal approach is to estimate WCPI by executing the PUT with other interfering programs [3, 6, 12, 16]. Each free CPU core in the processor executes a program that stresses a targeted shared module potentially increasing interference with the PUT. While such approaches result in more precise WCPI estimates compared to static techniques, they have the

following limitations. First, since each CPU core stresses a single shared module, these techniques do not scale well with an increase in the number of shared hardware modules. Second, the stress programs are pre-determined and cannot be easily adapted to different execution phases of the PUT (Fig. 1).

In a recent work, Li *et al.* [7] present PolyRhythm, which attempts to maximize interference using similar stress programs. Unlike [6, 16], which stress different components from each available CPU core, the PolyRhythm tool periodically selects the program with the highest interference using Reinforcement Learning and executes it on all the cores for a fixed duration. This enhances the scalability of the solution as compared to [6, 16], since PolyRhythm is not constrained by the number of free CPU cores available. However, it has the following limitations. **(a)** Since an interfering program is executed for a pre-determined duration, there is no further fine-tuning possible. **(b)** Executing the same stress program on each CPU core need not result in optimal interference, especially for a real-time PUT with heterogeneous and non-uniform accesses to different hardware modules. **(c)** At the end of each duration, a context switch needs to be performed on all the CPU cores causing short periods of no interference. **(d)** Finally, to perform the switching, PolyRhythm relies on an underlying Operating System (OS), which might not be present in many real-time deployments. Kryptonite [19] addresses these limitations by providing a function-level granularity for interference without the requirement of an OS to perform context-switching. While this approach provides safe and precise WCPI estimates, similar to prior work, the output of Kryptonite does not provide any information about the interference behavior within different regions of the PUT.

Unlike existing techniques, Kryptonite++ automatically creates an interference environment that is fine-tuned to match the execution phases of the PUT. Further, it provides a region-wise distribution of interference for the PUT to aid software development teams in iterative improvements. Since the interference environment in Kryptonite++ is independent of the number of shared hardware modules, it can scale to complex hardware designs and PUTs.

VII. DISCUSSION

Explainability of results in Kryptonite++. For a given PUT on multi-core hardware, we provide the WCPI estimate and environment along with hotspots of interference. However, the explainability of the interference, as presented in the description of the hotspots in Table I requires manual efforts. We can further extend Kryptonite++ with program analysis techniques to also provide explanations about the observed interference in terms of the PUT’s operations.

Empirical dependence on gadgets. The WCPI environment determined by Kryptonite++ depends on the gadgets synthesized for the hardware. It is challenging to formally guarantee the completeness of the gadget set as the determination of the actual worst-case interference in a multi-core system is

TABLE I: The WCPI estimate of different programs in the Mälardalen [5] benchmark suite along with the functions that are hotspots of interference as marked by Kryptonite++. We also provide a description of the operations performed in these hotspot functions.

Program	WCPI estimate	Hotspot Functions	Description of hotspot functions
bbsort	58.7%	BubbleSort()	Bubble sort on an array in LMU0.
binary_search	18%	binary_search()	Binary search on an array in LMU0.
cnt	18%	Initialize(), Sum()	Read/Write to a matrix in LMU0.
compress	50.2%	initbuffer(), writebytes()	Read/Write to a buffer in LMU0 and LMU1.
crc	22.8%	icrc()	Read/Write to a buffer for arithmetic operations.
edn	32.6%	jpegdet(), iir1(), latsynth()	Vector (of size 200) multiplications (read/write) and array handling in LMU0.
fdct	28.1%	fdct()	read/write on integer array elements.
fft1	5.7%	None	All arithmetic operations, limited data handling form LMU0.
fir	30.8%	fir_filter_int()	Read/Write on two arrays length 700 in LMU0 and LMU1.
insertsort	17.2%	main()	Insert sort algorithm on an array in LMU0.
matmult	26.7%	Multiply()	Matrix multiplication (read/write) in LMU0.
minver	9.4%	mmul()	Multiplication of scalar and a matrix (read/write) in LMU0.

TABLE II: An overview of different existing approaches for WCPI determination on multi-core systems (*IS-WCET: Interference-Sensitive Worst-Case Execution Time, WCPI: Worst-Case Program Interference*).

Paper	Analysis	Precision	Scalability	Fine-grained Tunability	Interference Localization	Output
Nagar and Srikant [10]	Static	Low	✗	✓	✗	IS-WCET bounds
Nagar and Srikant [11]	Static	Low	✗	✓	✗	IS-WCET bounds
Pellizzoni <i>et al.</i> [14]	Static	Low	✗	✓	✗	WCPI bounds
Skalitis [20]	Static	Low	✗	✓	✗	IS-WCET bounds
Potop-Butucaru [15]	Static	Low	✗	✓	✗	IS-WCET bounds
Nowotch [13]	Static	Low	✗	✓	✗	IS-WCET bounds
Courtaud [2]	Dynamic	High	✗	✓	✗	WCPI predictions
Griffin <i>et al.</i> [4]	Dynamic	High	✗	✓	✗	WCPI predictions
Nowotch and Paulitsch [12]	Dynamic	High	✗	✗	✗	WCPI estimates
Fernandez <i>et al.</i> [3]	Dynamic	High	✗	✗	✗	WCPI estimates
Radojkovic <i>et al.</i> [16]	Dynamic	High	✗	✗	✗	WCPI estimates
Iorga <i>et al.</i> [6]	Dynamic	High	✗	✗	✗	WCPI estimates
Li <i>et al.</i> [7]	Dynamic	High	✓	✗	✗	WCPI estimates
Kryptonite [19]	Dynamic	High	✓	✓	✗	WCPI estimates and environment
Kryptonite++ (this paper)	Dynamic	High	✓	✓	✓	WCPI estimates, environment and localized regions of interference

an NP-Hard problem [11]. We empirically observe that the 135 gadgets designed for Infineon AURIX TC399 (Fig. 6) consistently provide safe and precise WCPI estimates as compared to existing works.

Dependence on interrupt frequency. The Interference Analyzer in Kryptonite++ triggers timer interrupts every epoch to log the program counter associated with the PUT. The epoch duration and, thus, the granularity of scanning the PUT is dependent upon the frequency of interrupts. However, the maximum frequency of timer interrupts is dependent on the hardware, which can limit the granularity of sections of PUT that are mapped with given interference patterns.

VIII. CONCLUSION

Kryptonite++ provides a dynamic mechanism to first estimate the Worst-Case Program Interference (WCPI) and then

identify the hotspots of interference for a program on multi-core hardware. To this end, it selects gadgets targeting different modules and arranges them to form interfering sequences. It then performs a fine-tuning of these sequences using a Reinforcement Learning algorithm to output the Worst-Case Program Interference (WCPI) environment for a given program. In subsequent phases, it analyzes the maximally interfering environment along with the program under test (PUT) to pinpoint the instruction sequences that predominantly influence interference. This information is critical to application developers in improving the isolation of programs sharing the same hardware resources. Beyond WCPI estimation and localization for embedded real-time systems, such a framework opens up multiple avenues of research, such as the analysis of isolation across virtual machines on hypervisors or the identification of security vulnerabilities via shared hardware.

REFERENCES

- [1] Jon Perez Cerrolaza, Roman Obermaisser, Jaume Abella, Francisco J. Cazorla, Kim Grüttner, Irune Agirre, Hamidreza Ahmadian, and Imanol Allende. Multi-Core Devices for Safety-Critical Systems: A Survey. *ACM Comput. Surv.*, 53(4), Aug 2020.
- [2] Cédric Courtaud, Julien Sopena, Gilles Muller, and Daniel Gracia Pérez. Improving Prediction Accuracy of Memory Interferences for Multicore Platforms. In *IEEE Real-Time Systems Symposium, RTSS 2019, Hong Kong, SAR, China, December 3-6, 2019*, pages 246–259. IEEE, 2019.
- [3] Mikel Fernández, Roberto Gioiosa, Eduardo Quiñones, Luca Fossati, Marco Zulianello, and Francisco J. Cazorla. Assessing the suitability of the NGMP multi-core processor in the space domain. In *Proceedings of the 12th International Conference on Embedded Software, EMSOFT 2012, part of the Eighth Embedded Systems Week, ESWeek*, pages 175–184. ACM, 2012.
- [4] David Griffin, Benjamin Lesage, Iain Bate, Frank Soboczenski, and Robert I. Davis. Forecast-based interference: modelling multicore interference from observable factors. In Enrico Bini and Claire Pagetti, editors, *Proceedings of the 25th International Conference on Real-Time Networks and Systems, RTNS*. ACM, 2017.
- [5] Jan Gustafsson, Adam Betts, Andreas Ermedahl, and Björn Lisper. The Mälardalen WCET Benchmarks: Past, Present And Future. In Björn Lisper, editor, *10th International Workshop on Worst-Case Execution Time Analysis, WCET 2010, July 6, 2010, Brussels, Belgium*, volume 15 of *OASICs*, pages 136–146. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2010.
- [6] Dan Iorga, Tyler Sorensen, John Wickerson, and Alastair F. Donaldson. Slow and Steady: Measuring and Tuning Multicore Interference. In *IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS*, pages 200–212. IEEE, 2020.
- [7] Ao Li, Marion Sudvarg, Han Liu, Zhiyuan Yu, Chris Gill, and Ning Zhang. PolyRhythm: Adaptive Tuning of a Multi-Channel Attack Template for Timing Interference. In *IEEE Real-Time Systems Symposium, RTSS*, pages 225–239. IEEE, 2022.
- [8] Ralph Mader, Gerd Winkler, Thomas Reindl, and Nirad Pandya. The Cars Electronic Architecture in Motion: The Coming Transformation. In *42nd International Vienna Motor Symposium*, 2021.
- [9] Zoltan Majo and Thomas R. Gross. Memory System Performance in a NUMA Multicore Multiprocessor. In *Proceedings of the 4th Annual International Conference on Systems and Storage, SYSTOR '11*. ACM, 2011.
- [10] Kartik Nagar and Y. N. Srikant. Precise shared cache analysis using optimal interference placement. In *20th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS*. IEEE Computer Society, 2014.
- [11] Kartik Nagar and Y. N. Srikant. Fast and Precise Worst-Case Interference Placement for Shared Cache Analysis. *ACM Trans. Embed. Comput. Syst.*, 15(3), 2016.
- [12] Jan Nowotsch and Michael Paulitsch. Leveraging Multi-core Computing Architectures in Avionics. In *2012 Ninth European Dependable Computing Conference*, pages 132–143. IEEE Computer Society, 2012.
- [13] Jan Nowotsch, Michael Paulitsch, Daniel Buhler, Henrik Theiling, Simon Wegener, and Michael Schmidt. Multi-core Interference-Sensitive WCET Analysis Leveraging Runtime Resource Capacity Enforcement. In *26th Euromicro Conference on Real-Time Systems, ECRTS*, pages 109–118. IEEE Computer Society, 2014.
- [14] Rodolfo Pellizzoni, Andreas Schranzhofer, Jian-Jia Chen, Marco Caccamo, and Lothar Thiele. Worst case delay analysis for memory interference in multicore systems. In Giovanni De Micheli, Bashir M. Al-Hashimi, Wolfgang Müller, and Enrico Macii, editors, *Design, Automation and Test in Europe, DATE*, pages 741–746. IEEE Computer Society, 2010.
- [15] Dumitru Potop-Butucaru and Isabelle Puaut. Integrated Worst-Case Execution Time Estimation of Multicore Applications. In *13th International Workshop on Worst-Case Execution Time Analysis, WCET*, volume 30 of *OASICs*, pages 21–31. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2013.
- [16] Petar Radojkovic, Sylvain Girbal, Arnaud Grasset, Eduardo Quiñones, Sami Yehia, and Francisco J. Cazorla. On the evaluation of the impact of shared resources in multithreaded COTS processors in time-critical environments. *ACM Trans. Archit. Code Optim.*, 8(4):34:1–34:25, 2012.
- [17] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms. *CoRR*, abs/1707.06347, 2017.
- [18] Timothy Sherwood, Erez Perelman, Greg Hamerly, Suleyman Sair, and Brad Calder. Discovering and Exploiting Program Phases. *IEEE Micro*, 23(6):84–93, 2003.
- [19] Nikhilesh Singh, Karthikeyan Renganathan, Chester Rebeiro, Jithin Jose, and Ralph Mader. Kryptonite: Worst-Case Program Interference Estimation on Multi-Core Embedded Systems. *ACM Trans. Embed. Comput. Syst.*, 22(5s):149:1–149:23, 2023.
- [20] Stefanos Skalistis and Alena Simalatsar. Worst-Case Execution Time Analysis for Many-Core Architectures with NoC. In *Formal Modeling and Analysis of Timed Systems - 14th International Conference, FORMATS Proceedings*, volume 9884 of *Lecture Notes in Computer Science*, pages 211–227. Springer, 2016.
- [21] Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat, and Per Stenström. The Worst-Case Execution-Time Problem—Overview of Methods and Survey of Tools. *ACM Trans. Embed. Comput. Syst.*, 7(3), May 2008.

Session We.1.C

Benchmarking

Wednesday 12th June

10:00

–

Pastel Room

An Evaluation Bench for the Exploration of Machine Learning Deployment Solutions on Embedded Platforms

Eric Jenn^{*}, Floris Thiant[†], Theo Allouche^{§†}, Halima Bouzidi[¶], Ramon Conejo-Laguna^{*†}, Omar Hlimi^{*†}, Cyril Louis-Stanislas[‡], Christophe Marabotto^{*†}, Smaïl Niar[¶], Serge Tembo-Mouafo^{*†}, Philippe Thierion^{||†}

^{*} IRT Saint Exupéry [†] IRT System-X [‡] Viveris Technologies [§] ATOS

[¶] Université Polytechnique Haut de France, ^{||} Ampere

Keywords: Machine learning, Embedded systems, Embarcability, MLOps

Abstract—Finding the most efficient deployment of a Machine Learning (ML) model requires setting up multiple combinations of ML tools and hardware targets, running series of experiments, and evaluating relevant parameters (latency, memory usage, etc.). All these operations are complex, sometimes tedious, and always time consuming. Therefore, in order to facilitate this Design Space Exploration process, we propose an evaluation bench that (i) integrates the necessary software and hardware resources (tools, boards) to deploy a variety of ML models, and (ii) provides a uniform and abstract API to exercise and evaluate multiple deployment solutions. This paper defines more precisely the end-users needs, describes the architecture of the bench and illustrates its application on a use case.

Index Terms—Machine learning, edge computing, deployment

I. INTRODUCTION

A typical embedded system has to comply with a set of strict requirements about size, weight and power (SWaP), response times, reliability, certification, and – eventually – cost. Satisfying all of these constraints presents a significant challenge when deploying complex Machine Learning (ML) algorithms. Finding the combination of ML framework, backend, and hardware target that satisfies all of these constraints at an acceptable cost is a complex optimization problem that is currently solved by combining return of experience, analysis, and, essentially, experimental exploration. Considering that the approach is essentially empirical, being able to evaluate a large set of solutions in a fast and reproducible way is crucial.

In this context, our work aims to facilitate and partially automate the process of exploring the design space. The central challenge is to reconcile a high level of automation [7], a wide diversity of hardware targets and toolchains, and the need for repeatability, controllability, observability, and mastery of the hardware and software components involved in the deployment process

This work is done in the frame of the Con fiance.ai program¹ led by IRT SystemX whose objective is to foster the adoption of Artificial Intelligence by providing methods, environments,

tools, and components to design and industrialize trustworthy AI-based critical systems. The evaluation bench presented hereafter is one component of a larger environment developed in the program that includes in particular Model-based engineering methods for the development, verification and validation of those systems, software libraries to address some crucial issues such as robustness or explicability, etc.

To achieve the objectives of efficient Design Space Exploration, we propose an environment integrating (i) a set of preinstalled and configured Machine Learning (ML) implementation tool chains, (ii) a set of hardware targets including low-end microcontrollers, high-end System-on-Chip (SoC) fitted with multiple cores, GPUs, ASIC ML accelerators or FPGAs, (iii) an API to setup and run experiments and collect results, (iv) means to generate automatically candidate deployment workflows, (v) means to predict performances of hardware targets not present in the bench or not available, (v) documentation.

The paper is organized as follows: Section II presents the typical use cases and operational needs of the evaluation bench. Related work is detailed in Section III. The architecture of the bench is presented in Section IV and its usage is illustrated on a typical use case in Section V. Section VI concludes the paper and presents perspectives for further developments.

II. USE CASES AND OPERATIONAL NEEDS

The typical use case for the evaluation bench is the situation where (i) the ML model has been designed and its ML performance assessed on a development platform (e.g., a PC running TensorFlow), and (ii) the hardware target and associated deployment workflow remains to be chosen considering all the other constraints than ML performance. Finding the optimal deployment of a ML model is then an optimisation problem with multiple degrees of freedom. Often, the set of possible solutions to this problem is strongly reduced due to domain-specific constraints such as the compliance with certification objectives, the compatibility with specific environmental conditions, etc., or company-specific constraints such as the prescription or prohibition of specific ML frameworks

¹See www.confiance.ai.

or hardware targets. Nevertheless, the set of potential solutions remains usually large.

For each solution, the user has to select a toolchain compatible with his/her input model, install and configure it for a given hardware target, implement and deploy the model on the target, execute it and perform measurements. Developers are usually familiar with these activities for a limited set of software and hardware combinations. Difficulties arise when new tools or new platforms are to be evaluated since, then, the user will have to tackle the usual issues of model/toolchain and toolchain/OS compatibility, toolchain configuration, etc. In addition, if it makes sense to maintain 3, 4 or 5 different tools or targets on a local setup, installing, maintaining and analyzing 10 or 20 or more tools and targets becomes a real issue.

To address this general use case, the evaluation bench shall have capability to

- Give access to a large variety of hardware targets covering a wide range of CPU and GPU architectures, FPGA, AI ASIC accelerators, execution environments, ML deployment tool chains
- Carry out evaluation experiments in as many configuration as necessary without having to setup the relevant environments, deal with dependencies, etc.
- Leverage results from past experiments.

Those capabilities rely on several main features:

- A set of hardware boards with their execution environment
- A set of preinstalled ML deployment tool chains (e.g. TensorFlow, TensorFlow lite, TIDL, Vitis AI, finn, etc.)
- A set of ML models (aka "model zoo") to be used for evaluation purposes
- detailed information about the transformations performed by the tool chains
- Means to generate implementation and deployment workflows for the available hardware targets and toolchains according to user-defined constraints
- Means to execute selected workflows in an automatic manner
- Means to replace a hardware target by a surrogate model for inference time and memory usage estimations
- Means to control the state of the target boards (i.e., power-on/off, reset).

III. RELATED WORK

With the ever growing use of ML models in operations, a lot of work has been done to industrialize the ML deployment workflow in order to guarantee consistent, reliable and repeatable results. This practice, referred to as MLOps, is based on the DevOps practices used for the development and maintenance of non ML software systems, and extended to cover the development of the data necessary to the training phase. Note that in the context of this study, focus is placed strictly on the inference phase.

MLOps is already supported by dedicated services and frameworks provided by the major Cloud providers, such as

Amazon with SageMaker², Google with TFX³, or Microsoft with Azure⁴, etc. All these services are dedicated to a specific cloud-hosted execution platforms and do not target embedded systems. For those systems with limited memory, processing, and energy resources, specific practices have been developed under the name of Tiny-MLOps [1] (from tiny-ML [11] and MLOps).

Tiny-MLOps faces several main challenges. First, the capabilities of the target hardware are extremely large, ranging from small microcontrollers with low processing capabilities (e.g., one core at a few hundred MHz), limited communication capabilities (e.g. buses such as USB, SPI, I2C, etc.), limited flash and RAM memory, and no operating system, to high-end SoCs with multi-core CPUs, GPUs, AI accelerators, Ethernet connectivity, megabytes of RAM and flash, and running Linux. Second, those targets are supported by a very large range of ML frameworks including the usual ones based on Python (e.g., TensorFlow, PyTorch, etc.), and the proprietary ones tuned for some specific hardware (e.g., Vitis-AI for AMD's MPSoCs, STM32 CubeAI for STMicro STM32 chips, etc.). The crux lies in establishing the appropriate pairing between the implementation chain and the hardware target, as this synergy is essential to satisfy all performance objectives, including functional (e.g., object detection and classification capability) and non-functional (e.g., response time) requirements.

MLOPS	TinyMLOps	Model Serving	Benchmarking
Amazon SageMaker ■	Edge Impulse ■	Tensorflow Serving ■ ■	MLPerf Inference Tiny/Edge ■
Activeeon ProActive AI Orchestration ■ ■	Amazon SageMaker Edge ■ ■	BentoML ■	MLBench ■
Google Vertex AI ■	MicroEJ ■	Nvidia Triton Inference Server ■ ■ ■	Our evaluation bench ■ ■ ■ ■ ■

- Optimization control (compilation/inference)
- Methodology/Benchmarks
- Support of high-end devices
- Support of low-end devices
- Extensibility/customization

TABLE I
COMPARISON OF OUR EVALUATION BENCH WITH SOLUTIONS ALLOWING TO DEPLOY OR BENCHMARK ML MODELS ON HARDWARE.

Table I presents four categories of solutions to deploy/benchmark ML models on hardware targets:

- 1) MLOps solutions provide a fully managed service for building, training, and deploying machine learning models at scale. They offer a wide range of built-in algorithms and frameworks for model training, as well as features for automatic model tuning, monitoring, and scaling.

²<https://aws.amazon.com/fr/sagemaker/>

³<https://www.tensorflow.org/tfx>

⁴azure.microsoft.com

- 2) Tiny-MLOps solutions are designed for deploying ML models to tiny/edge devices by providing optimized inference runtimes for resource-constrained environments, and provide a service for building, training, and deploying ML models on tiny/edge devices.
- 3) Model serving solutions provide a flexible and scalable architecture for serving models via gRPC or REST APIs. They support model versioning, model management, and monitoring capabilities, simplifies the deployment process by containerizing models along with their dependencies and various ML frameworks and cloud platforms
- 4) Benchmarking solutions provide standardized benchmarks and metrics for evaluating inference performance on edge devices. They help comparing the efficiency of different hardware platforms and software frameworks, includes benchmarks for training and inference tasks across various hardware and software configurations, and offers a comprehensive set of metrics and tools for performance analysis.

As shown on the previous table, several frameworks target MLOps/TinyMLOps for low-end (MCU) and high-end devices (CPU, GPU, FPGA) by providing automate pipelines, but they lack the capability to fine control the optimization process thus limiting the transparency in the implementation. Benchmarking solutions such as the MLPerf Tiny Benchmark [2] leave to each user the choice of deployment pipeline, thus limiting automation and reproducibility. Our environment tries to reconcile the objectives of automation and fine-grained control. In addition, it also integrate predictive models making it possible to estimate performance metrics without requiring the actual deployment and execution of the model on the hardware target.

Since the beginning of this project, several solutions have been proposed that target a similar objective, i.e., provide an environment to evaluate easily different implementation configurations. For instance, ST Micro’s STM32Cube.AI Developer Cloud⁵ allows a user to upload a model, select optimizations, generate the implementation model, upload it on an actual target board in the STM32 family, and get the performance results. This is partially similar to what we propose, but this solution is strictly restricted to STM products.

Finally, it is worth noting that our work is different to that of Hardware Network Architectural Search (HW-NAS), which is focused on finding the best neural network architecture for one or multiple hardware targets, considering multiple criteria such as latency, memory footprint, or energy consumption. Research on HW-NAS are very large [3] and increasing. We can mention for instance MCUNet [8] that proposes addresses the system-algorithm co-design for microcontroller targets. Our work starts when the model has been designed, but integrating HW-NAS capabilities is actually one perspective for the development of the bench.

⁵See <https://stm32ai-cs.st.com>.

IV. TEST BENCH ARCHITECTURE

In this section, we present the main components of the evaluation bench: (i) the bench infrastructure component, (ii) the performance prediction component, (iii) the workflow generation component, (iv) the hardware targets component, (v) the documentation component.

A. Bench infrastructure component

The evaluation bench is composed of a set of components encapsulated in docker containers⁶ providing services accessible via gRPC⁷ remote procedure calls.

Services concern (i) the management of the ML experiments lifecycle (e.g., set up experiments, implement and deploy model, run inferences, collect experimental results, etc.) and (ii) the management of the state of the hardware infrastructure (e.g., power-up/down of boards, monitor voltage or temperature, etc.). They are accessible using a Python API that masks the heterogeneity of the execution platforms (from high-performance targets running Linux, possibly with Android, to bare metal microcontrollers).

In order to leverage existing standards for ML deployment, we integrate the open-source Nvidia Triton Inference Server⁸ that implements the Predict Protocol⁹), a gRPC inference API independent of any specific ML framework and model server that is optimized for CPUs and GPUs. We plan to extend this inference server to handle unsupported hardware or execution provider implementation (also called *backends* such as PyTorch, TensorFlow, TensorRT, etc.). In a way similar to the Predict Protocol, we provide an independent API for each class of services used in a ML pipeline (e.g “Deployment Protocol”, “Conversion Protocol”, “Quantization Protocol”, etc.), but without being restricted to a limited set of hardware targets and their associated software tools.

Fig.1 gives an overview of the main components of the evaluation bench architecture. (Fig. 1):

- 1) **Front-end** (Python/Jupyter notebook): a user-level Python API used to create or reuse a workflow. A workflow is represented by a DAG (Directed Acyclic Graph) in which nodes are calls to the services provided by “Executors” (see below).
- 2) **Orchestrator** (Airflow¹⁰): a component in charge of executing a DAGs. It relies on two sub-components: a **Configurator** providing the configuration data required by the Executors and a **Controller** managing the state of components (*init*, *config*, *start*, *stop*, etc.)
- 3) **Request Interpreter**: an Airflow service relying on the gRPC remote procedure call protocol, which build dynamically the gRPC operator corresponding to services interfaces (i.e. Python abstract class) in accordance with the serialization/deserialization protobuf¹¹ schemas.

⁶www.docker.com

⁷<https://grpc.io/>

⁸github.com/triton-inference-server/server

⁹github.com/kserve/kserve/blob/master/docs/predict-api/v2/

¹⁰<https://airflow.apache.org/>

¹¹<https://protobuf.dev>

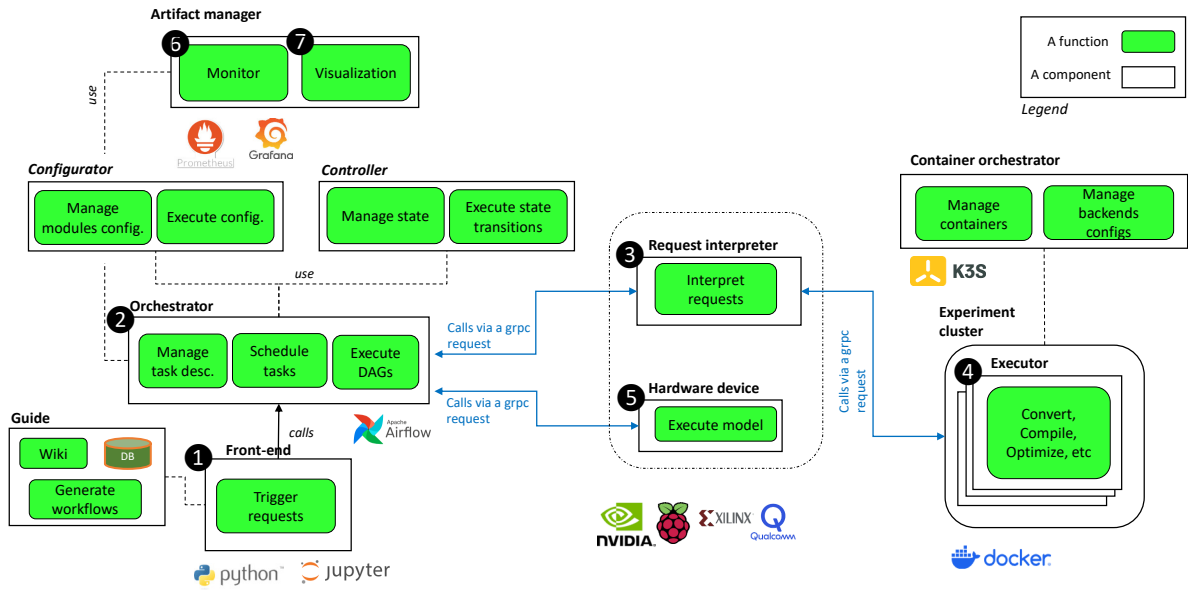


Fig. 1. Evaluation bench software architecture

- 4) **Executors** (Docker): containers embedding implementing the elementary services used in a workflow (e.g., model conversion, model quantization, model implementation, etc.). Executors are grouped into clusters providing isolation and managed by the Container Orchestrator (K3s¹², a lightweight Kubernetes¹³) under control of the main Orchestrator
- 5) **Hardware devices**: target HW boards on which models are deployed and run
- 6) **Monitoring** (Prometheus¹⁴): a component that collects and stores experiment hardware/ML metrics), and monitors the platform (system health, behavior, and performance). It uses Prometheus HTTP endpoint or raw text files for low-end targets
- 7) **Visualization** (Grafana¹⁵): a component that provides a dashboard to monitor the environment and the execution of the experiments.

B. Configuration component

Aged by a dedicated component, the Configurator, which reads yaml files and supplies all components with the configuration parameters they require, enable dual operation: function calls with parameters, in line with grpc client-server architecture, or, for greater flexibility, direct parameter retrieval by a developer wishing to quickly integrate a new toolchain, without necessarily having to implement all the necessary services. In addition, this component, responsible for storing configuration data in a database, also holds a set of global variables representing the state of the system (e.g. : $flag_run ==$

$result_of_is_an_experiment_running, board_1), flag_available == is_available(board_2))$ avoiding over-querying, therefore minimizing interference/sources of variability on hardware behavior. The configuration files ingested by the Configurator are in yaml format, as illustrated on Fig. 2, representing the environment to build and deploy, the target to reach, the ml application to run, the measurement to perform, or the full automatized workflow to execute.

The services of the benchmarking environment are implemented by components packaged in the form of Docker containers. To facilitate their management, and in keeping with a TinyMLOps approach, the container orchestrator K3s was chosen, in particular because of the specific nature of our platform, which has to interact with hardware connected to physical servers located in different geographical locations. According to the K3s/Kubernetes terminology, these components/dockers are called Pods, a group of one or more containers with shared storage and network resources, and a specification for how to run the containers. All the Pods required to perform an experience, i.e. to execute the actual services using grpc calls, are inter-connected via a logical sub-network to mask the complexity of network addressing. For this purpose, we have superimposed the concept of services to Kubernetes Service, a method for exposing a network application that is running as one or more Pods in the experiment cluster. In our case, this means exposing the grpc server running in the pod, whose ip address provided by the bench configuration file is translated into a Kubernetes Service name.

C. Orchestration component

As mentioned in the previous sub-section, the user-generated DAG (also called ExpDAG) interacts with the

¹²<https://k3s.io/>

¹³<https://kubernetes.io/>

¹⁴<https://prometheus.io/>

¹⁵<https://grafana.com/>

```

config_files:
- !include config_services.yml
- !include config_envs.yml
- !include config_workflows.yml
- !include config_targets.yml
- !include config_components.yml

applications:
- application: !application
  identifier: app_1
  model_name: densenet_onnx
  model_folder: $MODELS_REPO
  data_name: mug.jpg
  data_folder: $MLTB_DATASET/images
  scaling: INCEPTION
  shape: [3, 224, 224]
  dformat: FORMAT_NCHW
  dtype: FP32

measurements: # i.e metrics
- measurement: !measurement
  identifier: measure_1
  mname: nv_inference_compute_infer_duration_us

optimizations:
- optimization: !optimization
  identifier: optimization_1
  otype: quantization
  calib_dataset: $MLTB_DATASET/calib_dataset.py
  calib_data_set_path: $MLTB_DATASET
  strategy: static
  value_type: int8

exps:
- exp: !exp
  identifier: exp_1
  name: my_exp1
  application: app_1
  workflow: workflow_1
  target: testbench_1
  measurements: [compute_infer]
  optimizations: [optimization_1]

```

Fig. 2. Configuration of testbench

Configurator to retrieve the parameters needed to build the Airflow task to be executed, i.e. a `grpc` task (Airflow `Grpc` Operator). The building of user request, and the full process leading to the generation of the new `grpc` request is illustrated on Fig. 5.

This DAG, which generically applies a sequence of controllable transitions via the application of a schema (`init`, `config`, `start`, `stop`, `deinit`), will eventually be integrated into the Configurator to generate as much as possible of the implementation enabling the DAG to be executed under constraints, i.e. the preceding schema or extended according to the desired granularity (e.g. `:` for all optimization services of type `quantization`, apply the following control schema : `call_static_quantization` \rightarrow `call_recalibrate`).

Fig.1 illustrates the execution workflow corresponding to configuration file on Fig.2

D. Performance Prediction component

The bench integrates a variety of hardware boards, but there is usually only one instance of a given reference. So, once the board is used by some user, it becomes unavailable for other users during a certain time. To alleviate this limit, the bench integrates a component to *predict* performance indicators (latency, memory usage, etc.) without using the actual hardware. This feature, based on the work presented in [4], also allows the evaluation of a deployment solution when the actual target hardware exists but is not yet installed in the bench. It could also be useful to get a first estimation of performance or resource usage when implementing and deploying the model is time consuming. This could be the case, for instance, when implementing the model requires model conversion, code generation and compilation, or gate-level synthesis (e.g., using FINN [10]). As of today, this approach has been only used on the NVIDIA Jetson family.

The performance prediction process is depicted on Fig.3: a set of *prediction models* is trained using a dataset composed of (i) feature values extracted from a collection of state-of-the-art and synthetic models, and (ii) time and memory usage measured on hardware targets (NVIDIA Jetson Nano and AGX Orin). The set of features includes the number of FLOPS, the total number of layers, the number of fully-connected layers, batch-normalization layers, convolutional layers, etc. (see [4] for details). In order to obtain the best predictions, several prediction models have been used including ridge polynomial regression, support vector regression (SVR), multi-layer perceptrons, random forest and XGBoost (XGB). The best prediction results are given below for the two hardware targets.

Target	Inputs	Time		Mem.	
		Algo.	Acc.	Algo.	Acc.
Nano	1016	SVR	92.44%	SVR	91.12%
AGX Orin	1159	XGB	88.09%	SVR	96.12%

Currently, the inputs of the prediction model are the *features of the model* and not the *feature of the hardware targets* such as the number of cores, clock frequency, RAM speed, memory hierarchy, etc. Therefore, it does not allow inferring the performances of a model for a hardware target that has not been used during the training phase. This is an extension to be considered, but obtaining large datasets will be difficult without resorting to synthetic, simulated architectures.

E. Workflow generation component

In order to automate the exploration of multiple deployment solutions, our environment provides the ability to generate *workflows* automatically from the definition of the workflow *input* (a ML model) and *outputs* (a set of measurements), and a set of *constraints*. Constraints may concern the hardware target (e.g., the country of origin, the number of cores, the architecture of the cores, etc.) or the toolchain components (e.g. prevent workflows that use TensorFlow or, conversely, only use workflows that use TensorFlow lite, etc.). In order to simplify the end-user’s task, the constraints are not expressed in Prolog, but using a dedicated configuration file (YAML). This file contains “positive” and “negative” constraints (“or” “obligation” and “prohibitions”). For instance, the following excerpt of a constraint definition file states that the input model is a Tensorflow model (line 2) using three different operators (line 4-5), and the output metrics are “latency” and “accuracy” (line 7). In addition, it also indicates some constraints about the converter components, frameworks, delegates and backends that may use in the workflow.

```

1 # Model constraints
2 model_framework : 'tensorflow'
3 model_ops_list :
4   ['tensorflow_Conv2D', 'tensorflow_DepthWiseConv2D',
5    'tensorflow_ReLU6']
6 # Metric constraints
7 metrics : ['latency', 'accuracy']
8 converters : ['tf2tf1']

```

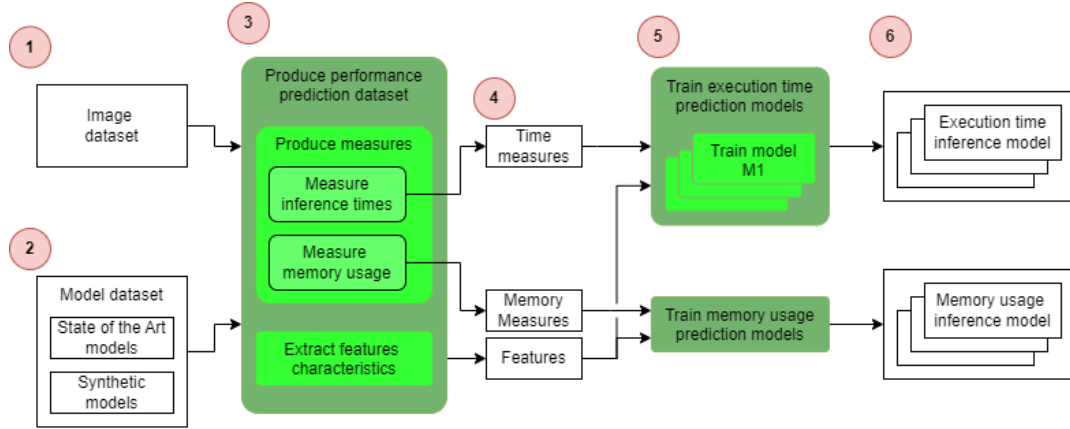


Fig. 3. Performance prediction workflow

```

9 frameworks : ['tflite']
10 delegates : ['xnnpack', 'gpu_delegate',
11             'hexagon_delegate']
12 backends  : ['cpu_kernels', 'opengl',
13             'hexagon_skels']

```

The set of candidate workflows is produced automatically by combining transformations according to compatibility rules between the outputs of a transformation and the inputs of the next transformation in the workflow, and the user-defined constraints. Combined with the other features of the environment, this capability allows a user to generate a set of candidate workflows and iterate automatically on each of its element using standard Python constructs.

A workflow is a sequence of calls to gRPC services. “Converting a model”, “building the binary file that implements a model” and “deploying a model” are examples of functions implemented by services. A service takes a set of inputs and produces a set of outputs. Each service is described by a specific Prolog fact, as shown below for the `do_convert` service of the “TensorFlow Lite for microcontroller” component.

```

1 is_activity([tfilm, do_convert], [pc, linux],
2           [tf2tfl, Method],
3           [[model, [tflow, Opss ], [pb_file, pc]],
4           [[model, [tflite, Opsd]], [fbs_file, pc]]) :-
5   Quant =
6     [quantize_int8, quantize_int16,
7     quantize_uint8,
8     quantize_uint16,
9     quantize_dynamic_range],
10    member(Method, Quant),
11    are_all_equivalent(tflow, Opss, tflite,
12                      Opsd).

```

In this example, the service is executed on a pc running linux. This operation has a parameter (Method) that describes the type of quantization to be done. Quantization may be `quantize_int8`, `quantize_int16`, etc. The input is a model of the type `tflow` (tensorflow) that uses a certain set of operators (Ops) denoted by `Opss` (Ops Source) that is stored in a protobuf file (`pb_file`) stored

on a pc. The component outputs a tflite model with another set of operators denoted by `Opsd` (Ops Destination). The relation between `Opss` and `Opsd` is expressed by the `are_all_equivalent` clause that checks that the set of operators in `Opsd` is “compatible” with those of `Opss`.

Usually, operators realizing the same functions have the same name in different toolchains. When this is not the case, correspondence is described globally using “dummy operators”. For instance, the following predicate states that a GPU operator is compatible with a dummy macro operator if the operator belongs to the list of GPU operators.

```

1 is_equivalent(Gpud_Op, ['openc1_MOp']) :-
2   are_operators(gpu_delegate, L),
3   member(Gpud_Op, L).

```

Building a workflow boils down to finding a sequence of activities accepting a given *model* as input and producing a given *measure* as output. The workflow is a linear graph of input/output compatible activities, as defined by the following clause.

```

1 is_edge([[C1, S1], E_1, P_1, In_1, Out_1],
2         [[C2, S2], E_2, P_2, In_2, Out_2]) :-
3   is_activity([C1, S1], E_1, P_1, In_1, Out_1),
4   is_activity([C2, S2], E_2, P_2, In_2, Out_2),
5   are_IO_compatible(S1, Out_1, S2, In_2).

```

Example: Fig. 4 gives two examples of workflow generated automatically for a model `mdl` and a measure `mea`. The first workflow targets the ESP32 micro-controller. In this case, the model needs first to be converted into a C program that is built and deployed on the board before inferences can be executed and results can be obtained. In the second workflow, the model can directly be deployed without conversion, then executed and measures be obtained.

F. Hardware infrastructure component

Fig.5 shows a picture of the actual bench with a close view of a rack. Each rack hosts a set of target boards with their accompanying monitoring boards. A monitoring board provides the capability to control the board’s power supply

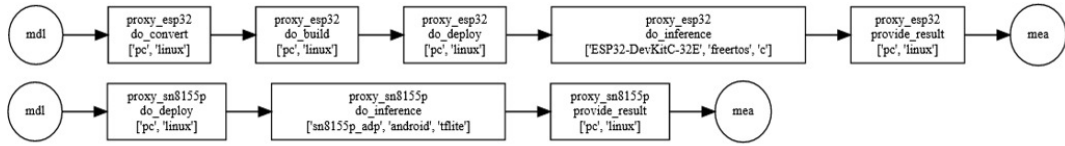


Fig. 4. Example of automatically generated workflows

(power-on/off), to reset it, and to monitor its power supply voltage and current. This allows to power on board only when needed and to reset them to a known state between experiments. In addition, each rack is fitted with a simple HMI to display and control the rack state.

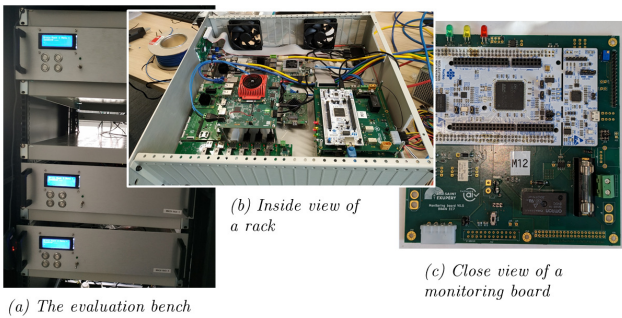


Fig. 5. The test bench hardware

G. Documentation component

Providing appropriate information about the software frameworks and hardware targets is another objective of the evaluation bench. To facilitate informed decisions on deployment strategies, a precise representation of how frameworks interact with target boards is essential. Towards that goal, the bench hosts a dedicated Wiki providing documentation about the hosted hardware boards and toolchains/frameworks. Part of the Wiki's content is generated from a database of boards and tools. This database is also used when setting up experiments (see Sect.V).

The Wiki serves as a “front-end” to the documentation publicly available on the Internet, but focus is placed on documenting the transformations performed by the ML implementation tool chains. For instance, a section of the Wiki is dedicated to the TDA4VM target. It provides data about its architecture, the supported ML frameworks, the previous performance logs, etc. Two deployment solutions, TIDL and OpenVX, are currently documented. The documentation of the OpenVX toolchain is structured as follows (the same applies for the other toolchains):

- Overview: This section provides a general introduction to both OpenVX and TIOVX frameworks, offering readers a comprehensive understanding of their functionalities.
- Toolchain analysis: Here, readers gain insight into the workflow of TIOVX, understanding the steps involved

in implementing a model. A detailed guideline is provided, offering clear instructions on how to execute the model implementation process. Following this, a semantic preservation study is conducted to ensure the implemented model aligns identically with the reference model (ONNX model).

- Limitations: Here, the limitations of the TIOVX toolchain are described, offering a transparent view of its potential challenges.
- A Frequently Asked Questions table (F.A.Q): A table of frequently asked questions is established to provide developers with quick and helpful answers to common issues.

The “Toolchain Analysis” section contains an overview of its key points. The diagram in Fig.6, extracted from the Wiki, shows the sequence of steps to implement and execute the model on the TDA4VM target. The workflow is divided into a host and a target part. Concerning the host, for instance, the process includes generating, registering, and implementing custom kernels, which in the case of ACAS-Xu model, represent different operators (e.g., QuantizeLinear, QGEMM, etc.) Additionally, it demonstrates how to develop the OpenVX application and integrate the kernels onto it. Lastly, the compilation process is detailed to ensure successful application compilation.

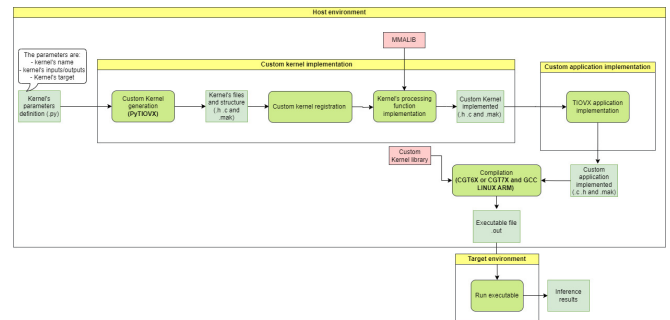


Fig. 6. TIOVX workflow description

Section “Toolchain Analysis” also addresses the important question of the preservation of the semantic of the model during the implementation and deployment process. In this section, we analyze the impact of the transformations involved in this process (incl. compilation, optimization, compression, etc.) on the semantics of the input model. For instance, during the experiments carried out on the deployment of the ACAS-XU model [6] on the TD4VM board, some differences were observed between the outputs of the model executed on the

development environment (the host PC with ONNX runtime) and on the embedded target (the TDA4VM). To resolve this issue, the ONNX Runtime implementation of the QGEMM operator was examined to identify the root causes of the differences. Based on this study, modifications were made to our QGEMM code to align it with the ONNX Runtime implementation. This ReTex is an example of what is collected in the bench Wiki.

V. THE EVALUATION BENCH AT WORK

A. Use Case scenario

A typical Use Case for the testbench is the following: the user needs to choose the best (hardware target, toolchain) combination to deploy a ML model that complies with a set of requirements about ML, timing and resource usage performances, and constraints about acceptable hardware targets and toolchains.

In order to make this choice, the user needs (i) data provided by the documentation, retex and analysis and (ii) data provided perform a series of experiment

- 1) The user uses the Python API to consult the list of hardware targets and deployment toolchain available on the bench.
- 2) S/he consults the documentation available on the bench to make a first selection of targets and tools.
- 3) S/he extracts the list of hardware target compliant with a certain criterion (e.g., country of manufacturing, core architecture, memory size, etc.) using the bench database API.
- 4) S/he creates the configuration file corresponding to his/her experiment (e.g., name of the model to be deployed)
- 5) S/he uses the workflow generation tool to generate a set of candidate workflows from the model to the metrics
- 6) S/he uses Python to loop over all possible workflows and obtain performance measures for each of them, i.e., (i) reserve the hardware resources necessary to execute the workflow, (ii) setup the hardware resources (power supply), (iii) execute the workflow, (iv) collect the results
- 7) S/he uses Python to display the results and select the most efficient implementation.
- 8) S/he selects the optimal solution.

This scenario has not been *partially* exercised on the implementation of the ACAS-XU¹⁶ [9] system proposed in [5]. This implementation uses 45 neural fully-connected neural networks (with identical structures and different parameters) corresponding to a certain discretization of the input domain. This case study has been used to validate the implementation and deployment components for the TDA4VM, GPU and FPGA targets. In addition, it has also been used as a typical example of a semantic preservation analysis (see Sec. IV-G). Hereafter, we consider a generic usage scenario that correspond to the expected usage for the ACAS-XU.

¹⁶Airborne Collision Avoidance System (ACAS) for Unmanned Aerial Systems (UAS).

The design space exploration (DSE) for the ACAS-XU involved 3 hardware targets and associated toolchains: the TDA4VM fitted with several DSPs using the TIDL and TIOVX toolchains, the Xilinx Crya fitted with an UltraScale+ SoC (CPU+FPGA) using the VITIS-AI toolchain, and the NVIDIA Jetson Nano using the CUDA toolchain. To perform one experiment “by hand”, one would have to buy each board, setup each environment. This would take several weeks and a significant effort struggling with the hardware, with the configuration of tools, with library dependencies, etc. By simply providing readily available hardware and tools, the effort for design space exploration is significantly reduced. The capability to automate the execution of experiments (so: to reproduce them easily) also contributes to the reduction of the DSE effort since several configurations can be easily described and executed via a simple Python script.

B. Scenario execution

Figs. 7 to 11 show the Jupyter notebook script that implements the scenario. The end-user interacts with the bench mainly using a simple Python API. This solution, which does not rely on some fancy GUI, has been chosen because Python is the language usually used to develop the models and our bench is to be used right after the model development. In this context, a unique notebook can cover all phases, from the development of the model to its implementation, deployment, and evaluation on different targets. Details about the operations performed via the API are given hereafter.

Fig. 7 shows how the database provided by the evaluation bench is used to select a hardware board according to existing performance data obtained on a “standard” model – here, a ResNet-50 model. Then, the performance prediction component (Sect.3) is used to get a first estimation of performance data on the actual target model. This phase can be executed without the cost of deploying and executing the model on the actual hardware, and without requiring an exclusive access to this resource (that is currently unique in the bench).

```
# database access
import database

# get available boards
boards = database.get_mysql_db_data('select ref from brds;')

# set hardware constraints
hw_constraints = database.get_mysql_db_data(
    'select brd_ref from perfs where FPS >= 64;')
# -> return 'Jetson Nano'

# example of previous benchmark performed with Jetson Nano
# on a Resnet-50
perfs = database.get_mysql_db_data(
    "select execution_time from perfs "
    "where ml_model_ref='ResNet-50' and brd_ref='Jetson Nano;'"
# -> return value is 0 because now experiment has been performed

# check if performances prediction is available on this model
pred_client = PredictionClient('service-prediction')
exec_time = pred_client.predict(model_name='ResNet-50',
                               input_size=(32, 32, 3),
                               target='Jetson Nano',
                               perfo='execution_time')
```

Fig. 7. First platform selection and performance estimation

Fig. 8 shows how the automatic workflow generation component is used to propose a set of workflows, i.e., a set of

sequences of service calls translating a specification model to an executable model.

```
# workflow generation
# set new constraints returning several results
constraints = database.get_mysql_db_data(
    "select brd_ref from perfs where FPS >= 5 and "
    "ml_model_ref='ResNet-50' and execution_time < 100;")
# -> return several boards (gpu, fpga, mcu) and then specific services
# to perform a fully deployment and benchmark

# generate the corresponding workflow which produce the sequencing of
# couples (component, service) to perform an experiment
ret, workflows = conf_client.gen_wf(constraints)
ret = conf_client.set_workflows(workflows)
```

Fig. 8. Generate workflow associated to user constraints

In order to execute experiments, the test bench must be configured. This phase is performed by the script of Fig. 9 that loads the configuration from a YAML configuration file.

```
# testbench configuration
from configurator_client import ConfiguratorClient
# testbench configuration
CONFIG_FOLDER = os.environ['MLTB_CONFIG_FILES']
my_config = CONFIG_FOLDER + '/' + "config_exp.yml"

conf_client = ConfiguratorClient('service-configurator')
ret = conf_client.load_parse_conf(my_config)

ret, services = conf_client.get_services()
ret, targets = conf_client.get_targets()
ret, envs = conf_client.get_envs()
```

Fig. 9. Configure the testbench

Fig. 10 shows the script used to execute the series of workflows created previously. In the first loop, each experiment corresponding to a specific workflow is created, associated with a workflow and appended to a set of experiments. In the second loop, all experiments are executed by triggering the execution of the associated DAG. Note that an experiment may be executed without using the experiment orchestrator (based on Triton), by simply calling the basic services provided by the environment components (11).

```
# experiments creation for automatic execution
exps = [] # list of airflow DAGs
for i in enumerate(len(workflows)):
    exp = experiment.Exp('exp_' + str(i), conf_client)
    exp.load_workflow(workflows[i])
    exps.append(exp)
for exp in exps:
    exp.run(ipynbname.name()) # trigger the DAG
```

Fig. 10. Automatic execution of an experiment

The last excerpt of the Jupyter notebook, given on Fig. 12, shows the access to the measures collected during the experiment.

Several levels of visualization of the results associated with the experiment are available:

- 1) system level (Fig.13): allow to analyze the performances of the system, by providing some clues on potential bottlenecks (e.g. : data transfer time comparing to whole experiment execution time)
- 2) target level (Fig.14): allow to analyze the target's ability to hold the inference charge

```
# experiments creation for manual execution
exp = experiment.Exp('exp_1', conf_client)
# connect to target
ret0, srv0 = exp.init_service('connect_target')
ret0 = exp.config_service(srv0)
ret0, tid = exp.start_service(srv0)
# deploy Nvidia inference server to target gpu
ret1, srv1 = exp.init_service('deploy_env')
ret1 = exp.config_service(srv1)
ret1, tid = exp.start_service(srv1)
ret2, srv2 = exp.init_service('triton_infer')
ret2 = exp.config_service(srv2)
ret2, tid = exp.start_service(srv2)
exp.run(ipynbname.name())
```

Fig. 11. Manual execution of an experiment

```
# collect measures
import measurement
measures = measurement.get_measures('execution_time', conf_client)
# -> return all the execution times
```

Fig. 12. Collecting of measures

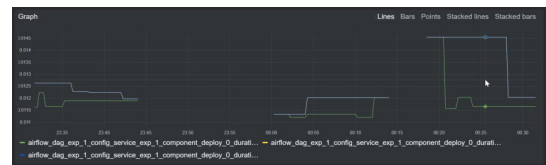


Fig. 13. System performances tracking

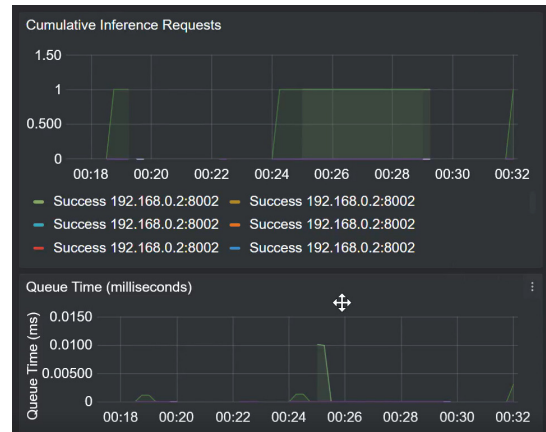


Fig. 14. Inference server/engine performances and history

- 3) ML level (Fig.15): allow to analyze the performances of ML model execution



Fig. 15. ML inference performances

VI. CONCLUSION AND PERSPECTIVES

In this paper, we have presented an environment aimed at facilitating the evaluation of ML models deployed on embedded targets. The environment provides a set of ready-to-use tools

and hardware targets with the accompanying infrastructure to define and orchestrate experiments using a simple Python API. We think that this environment can significantly reduce the effort to perform early design explorations. We also consider that there is an opportunity to put in common the element of analysis of the tool chain components, i.e., to expose and document in a standard way what are the transformation performed by the tools, what the properties preserved, etc.

Perspectives for future work are numerous. Currently, two separated benches have been developed on two different sites (one at IRT Saint Exupery in Toulouse and one at IRT System-X in Palaiseau). In the future, access to these boards will be possible transparently, wherever their actual physical location. We hope this bench to be opened to other partners in order to share resources and allow any participating members to have access to any of the software or hardware resources.

Among the other features that could be added to the bench. First, the capability to implement an actual optimization process besides what is done today, which is limited to generating all possible deployment solutions and evaluate them successively. In addition, intelligible explanations of the choice of candidate workflows or the optimal workflow will be provided to the user in the final version of the generator. Second, we will consolidate the integration of the evaluation bench in the ML system development environment built in the Confiance.ai programme. This environment provides guidance and components (incl. concept definitions, process models, argumentation models, ML libraries, etc.) to develop, verify and validate ML systems. The list of components can be found on the programme's web site. Eventually, the user will be able to design his/her ML system development process according to the Confiance.ai recommendations, develop his/her ML model using techniques enforcing important properties related to trust such as robustness, explainability, etc., and deploy and evaluate it on actual hardware targets, all these steps being done in a consistent environment. For the moment, we have considered that the model to be deployed was either provided by the user or taken from a "model zoo", but integrating a HW-NAS component in the bench to provide an "holistic" optimization would be an interesting extension.

ACKNOWLEDGEMENTS

This work has been supported by the French government under the "France 2030" program, as part of the SystemX Technological Research Institute. We would like to thank our shepherd for his many insightful comments and suggestions.

REFERENCES

- [1] Mattia Antonini, Miguel Pincheira, Massimo Vecchio, and Fabio Antonelli. Tiny-MLOps: a framework for orchestrating ML applications at the far edge of IoT systems. In *2022 IEEE International Conference on Evolving and Adaptive Intelligent Systems (EAIS)*, pages 1–8, Larnaca, Cyprus, May 2022. IEEE.
- [2] Colby Banbury, Vijay Janapa Reddi, Peter Torelli, Jeremy Holleman, Nat Jeffries, Csaba Kiraly, Pietro Montino, David Kanter, Sebastian Ahmed, Danilo Pau, et al. Mlperf tiny benchmark. *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, 2021.

- [3] Hadjer Benmeziane, Kaoutar Maghraoui, Ouarnoughi Hamza, Smail Niar, Martin Wistuba, and Naigang Wang. *Hardware-Aware Neural Architecture Search: Survey and Taxonomy*. August 2021. Pages: 4329.
- [4] Halima Bouzidi, Hamza Ouarnoughi, Smail Niar, and Abdessamad Ait El Cadi. Performance Modeling of Computer Vision-based CNN on Edge GPUs. *ACM Transactions on Embedded Computing Systems*, 21(5):1–33, September 2022.
- [5] Mathieu Damour, Florence De Grancey, Christophe Gabreau, Adrien Gauffriau, Jean-Brice Ginetet, 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 Sujun, and Friedemann Bitsch, editors, *Computer Safety, Reliability, and Security*, pages 34–48, Cham, 2021. Springer International Publishing.
- [6] Christophe Gabreau, Marie-Charlotte Teulières, Eric Jenn, Augustin Lemesle, Dumitru Potop Butucaru, Floris Thiant, Laurent Fradin, Pierre De la Ruffie, Lucas Fischer, and Cristian Maxim. A study of an ACAS-Xu exact implementation using ED-324/AR6983. In *12th European Congress on Embedded Real Time Software and Systems (ERTS 2024)*, Toulouse, France, June 2024.
- [7] Sam Leroux, Pieter Simoens, Meelis Lootus, Kartik Thakore, and Akshay Sharma. Tinyml: Operational challenges for widespread edge ai adoption, 2022.
- [8] Ji Lin, Wei-Ming Chen, Yujun Lin, Chuang Gan, and Song Han. Mcunet: Tiny deep learning on iot devices. *Advances in Neural Information Processing Systems*, 33, 2020.
- [9] Michael P. Owen, Adam Panken, Robert Moss, Luis Alvarez, and Charles Leeper. ACAS Xu: Integrated Collision Avoidance and Detect and Avoid Capability for UAS. In *2019 IEEE/AIAA 38th Digital Avionics Systems Conference (DASC)*, pages 1–10, San Diego, CA, USA, September 2019. IEEE.
- [10] Yaman Umuroglu, Nicholas J. Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees Vissers. FINN: A Framework for Fast, Scalable Binarized Neural Network Inference. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 65–74, February 2017. arXiv:1612.07119 [cs].
- [11] P. Warden and D. Situnayake. *TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-low-power Microcontrollers*. O'Reilly, 2020.

Multi-core WCET Analysis Using Non-Intrusive Continuous Observation

Daniel Kästner¹, Gernot Gebhard¹, Markus Pister¹, Simon Wegener¹, Christian Ferdinand¹,
Albert Schulz², Martin Sachenbacher², Martin Leucker³, and Alexander Weiss^{2*}

Abstract

For safety-relevant real-time applications, worst-case execution time (WCET) bounds have to be determined in order to demonstrate deadline adherence. For timing predictable microprocessors, worst-case execution time guarantees can be computed by static WCET analysis. Hybrid WCET analysis is a solution for covering effects from accesses to interference channels of multi-core processors. In this article we present a seamless approach for hybrid WCET analysis that tightly couples the tools TimeWeaver and CEDARtools. We will describe the underlying concepts, illustrate the tool workflow, and discuss the application of our approach to meet the timing requirements of the EASA AMC 20-193 guidance.

Keywords: DO-178C, multi-core, AMC 20-193, static analysis, real-time tracing, timing predictability, functional safety, certification

1 Introduction

In real-time systems the overall correctness depends on the correct timing behavior: each real-time task has to finish before its deadline. All current safety standards require reliable bounds of the worst-case execution time (WCET) of real-time tasks to be determined.

Until recently, current industry practice—in particular in the automotive domain [23]—oftentimes still relied on end-to-end measurements. However, the timing information obtained with them is only determined for one concrete input, but due to caches and pipelines, the timing behavior of an instruction depends on the execution history. Hence, one needs to take each possible hardware state into account. 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 are not necessarily identical to the timing behavior

of the original software. Moreover, the results of end-to-end measurements are hard to interpret, as they are not related to particular parts of the code but only to the whole program.

One safe method for timing analysis is static program analysis by Abstract Interpretation which provides guaranteed upper bounds for the WCET of a task. Static WCET analyzers are available for complex processors with caches and out-of-order pipelines, and, in general, support single-core and multi-core processors. A prerequisite is that reliable 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 integrates 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. Increasing the number of measurements for each snippet increases the chance to catch the possible worst-case state for each of them without the need to trigger the worst-case initial hardware state for the whole task. When the snippets cover the complete program under analysis, a worst-case path can be computed. The probe effect can be avoided by leveraging the embedded trace unit (ETU) of modern processors, which allows a fine-grained observation of a core's program flow. These traces are usually analyzed offline, but new FPGA-based approaches allow to analyze them online, enabling continuous non-intrusive runtime monitoring of embedded software.

For multi-core systems, the main challenge for WCET analysis is the interference generated by other cores running in parallel. AMC 20-193 [9] covers means to bound and mitigate these effects. When static WCET analysis is performed, the maximum costs of possible interference must be included in the result, for example with the help of a WCRA (worst-case resource accesses) analysis that gives safe upper bounds for shared resource accesses. In a second analysis step, the bounds are multiplied with the maximal interference delays of these shared resources, giving the maximal interference costs for a particular shared resource. For hybrid WCET analysis, the picture is simpler: all observable interference is already contained in the measurements, and no extra analysis step is needed.

In this article, we describe TimeWeaver, a hybrid WCET analysis tool, its coupling with CEDARtools to exploit state-of-the-art runtime monitoring, and its use in the context of AMC 20-193.

*The TRISTAN project, nr. 101095947 is supported by Chips Joint Undertaking (CHIPS-JU) and its members Austria, Belgium, Bulgaria, Croatia, Cyprus, Czechia, Germany, Denmark, Estonia, Greece, Spain, Finland, France, Hungary, Ireland, Israel, Iceland, Italy, Lithuania, Luxembourg, Latvia, Malta, Netherlands, Norway, Poland, Portugal, Romania, Sweden, Slovenia, Slovakia, Turkey and including top-up funding by the German Federal Ministry of Education and Research (funding IDs 16MEE0273 and 16MEE0276).

¹AbsInt Angewandte Informatik GmbH, Germany
{info@absint.com}

²Accemic Technologies GmbH, Germany
{schulz, msachenbacher, aweiss@accemic.com}

³University of Luebeck, Germany
{mleucker@isp.uniluebeck.de}

2 EASA AMC 20-193 Objectives

EASA Amendment AMC 20-193 [9], published in 2022, discusses means and defines objectives for the demonstration of compliance with the applicable airworthiness specifications for airborne systems and equipment that contains multi-core processors (MCPs). In the following we will briefly summarize its main contents and motivate the methodology outlined in this article with respect to AMC 20-193 verification obligations.

AMC 20-193 applies to systems with two or more activated cores not executed in lockstep mode for which the item development assurance level (IDAL) of at least one relevant software application is A, B, or C. One of the basic motivations of the amendment is to determine and mitigate inter-core interference, since “*interference between the software applications or tasks executing on an MCP could cause safety-critical software applications to behave in a non-deterministic or unsafe manner, or could prevent them from having sufficient time to complete the execution of their safety-critical functionality*”. Indeed, interference delays can have a huge impact on the memory access latencies. Nowotsch et al. [21] measured maximal write latencies of 39 cycles when only one core of the P4080 [11] was active, and maximal write latencies of 1007 cycles when all eight cores were running.

The guidance formulated by AMC 20-193 is structured in six stages, (i) *planning*, (ii) *setting of MCP resources*, (iii) *interference channels and resource usage*, (iv) *software verification*, (v) *error detection and handling, and safety nets*, and (vi) *data to complement the accomplishment summaries*.

The *planning* stage provides guidance to enhance the system development and verification processes in a way that will enable the hardware and the software hosted on the MCP to satisfy the functional, performance, and timing requirements of the system. Amongst others, Objective *MCP_Planning_1* requires that the applicant specifies the MCP they intend to use, the number of the active cores, and the software architecture hosted on the MCP. In particular, the applicant has to “*identify whether or not the MCP platform will provide robust resource partitioning and/or robust time partitioning*”. Objective *MCP_Planning_2* demands to describe the planned use of the shared resources, taking into account the time interference possibly caused by the usage, as well as the planned means to verify the usage, e.g., the tools and techniques planned for WCET analysis / timing verification.

Stage (ii) is concerned with the configuration of the MCP. According to objective *MCP_Resource_Usage_1*, the applicant has to determine and document any settings that may affect the system’s ability hosted on the MCP to satisfy the functional and non-functional requirements. In stage (iii), objective *MCP_Resource_Usage_3* requires identifying and mitigating interference channels, while objective *MCP_Resource_usage_4* demands to identify and allocate resources and to verify that the demands for resources of the MCP and the interconnect do not exceed the available resources.

In the *software verification* stage, objective *MCP_Software_1* aims at providing assurance that the time bounds defined for the system are not violated. If the platform provides robust

resource partitioning and robust time partitioning, the WCET of software applications may be determined separately. In this context, tools for computing static WCET guarantees, e.g., aiT WCET Analyzer [10] are applicable. If no robust resource and time partitioning can be guaranteed, the WCET has to be determined with all software components on all cores executing in the intended final configuration. This is the topic on which this article concentrates: we will present an efficient methodology for hybrid WCET analysis that allows interference-aware WCET bounds to be computed non-intrusively and that provides feedback on the trace coverage obtained.

Verification goal *MCP_Software_2* demands to verify that the data and control coupling between all SW components has been exerted, and that it is correct. Here, two aspects are needed: the feasible data and control coupling has to be determined, and the data and control coupling coverage achieved by the requirements-based testing has to be determined. An approach for sound data and control coupling analysis based on the static analyzer Astrée has been presented in [16]; the coverage information needed may be complemented by the trace data of the hybrid WCET analysis presented in the following sections.

Stage (v) is concerned with error detection and mitigation. Effects of any failure that may happen inside the MCP needs to be detected and handled according to the safety goals of the system hosted on the MCP. “Safety nets” may provide a fail-safe containment for these failures. Finally, stage (vi) requires the applicant to provide a description of how the objectives of AMC 20-193 are satisfied.

From a timing analysis point of view, the most crucial factors addressed by AMC 20-193 are the selection of the MCP; the identification, documentation, and assessment of interference channels; and the availability of robust resource and robust time partitioning. All these affect the suitability of the applied methods for timing verification. For example, a MCP providing timing predictability [5, 7, 30] and timing compositionality [12] allows for a fully static analysis. The effects of resource conflicts can then be bounded with a worst-case resource accesses (WCRA) analysis.

The availability of robust resource partitioning directly influences the choice of the analysis strategy, as formulated in objective *MCP_Software_1*. Robust partitioning can either be achieved via dedicated hardware features, or via a suitable software architecture. For example, the privatization of shared resources can prevent resource access conflicts and hence, provide robust partitioning. Ways to implement privatization of shared resources include TDMA-based resource scheduling [24] and runtime resource capacity enforcement [21].

However, robust partitioning may be hard to achieve for unpredictable or undocumented features of a MCP, as the same features that prevent the design and implementation of precise static timing analyses often also make the analysis and mitigation of interference channels difficult. Consider for example a shared cache with pseudo-random replacement: The mitigation of such an interference channel may need some kind of software-based cache partitioning [26] which complicates the software architecture and may conflict with other development goals, for example the use of specific operating systems or soft-

ware libraries.

Micro-benchmarks that intentionally drive contention on shared resources [28] (sometimes called “stressors” or “daemons”) are useful for determining and assessing interference channels, i.e., for platform characterization. They may uncover undocumented interference channels and measure the impact of resource conflicts on the timing behavior, i.e., they may help to determine the interference cost of a resource conflict for that specific interference channel. However, their applicability for measurement-based timing verification is limited. In case robust partitioning is available for the system hosted on the MCP, it will by definition prevent any adverse effects of the stressor on the software tasks for which the timing needs to be verified. If no robust partitioning is available, it will lead to overly pessimistic timing behavior being observed that not necessarily reflect the real timing behavior of the system. Moreover, the use of a stressor during measurement-based timing verification contradicts objective *MCP_Software_1* of AMC 20-193: the WCET has to be determined with all software components on all cores executing in the intended final configuration.

Instead, in accordance with AMC 20-193, we propose to apply hybrid timing analysis for commercial-of-the-shelf (COTS) multicore processors for which robust partitioning cannot be guaranteed. The measurements should be performed in the intended final configuration without any artificial generation of contention. Quite the contrary, the software architecture should prevent any unneeded interference, if possible.

3 Hybrid WCET Analysis

The goal of non-intrusive trace-based WCET analysis is to observe execution times of tasks and interrupt service routines (ISRs) including the timing interference due to concurrent execution and multi-core resource conflicts, while avoiding the probe effect.

The solution which is implemented in the hybrid WCET analysis tool TimeWeaver [1] combines static context-sensitive path analysis with non-intrusive 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. The probe effect is completely avoided since no code instrumentation is needed. The trace information can be provided out-of-the-box by embedded trace units of modern processors, like Nexus IEEE-ISTO 5001™ [13], Infineon MCDS™ [14], or ARM CoreSight™ [3], as used for example in the NXP Layerscape LX2xxx. These trace protocols allow the fine-grained observation of a program execution and assign timestamps to specific program points during execution. Thus, the traces contain an execution time measurement for each trace segment stretching out between two consecutive trace points. 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 by employing path extrapolation [18]. Thus, the coverage of the in-

put traces on the analyzed code is an important metric that influences the quality of the computed WCET estimates. ARM, PPC, RH850, and TriCore/AURIX are already supported; a TimeWeaver version targeting RISC-V is currently in development, exploiting the open trace interface developed as part of the European Chips JU TRISTAN [27].

3.1 Structure of TimeWeaver

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 (see Figure 1). Most steps in this tool chain are shared with aiT WCET Analyzer which provides a fully static analysis targeting timing-predictable processors [10].

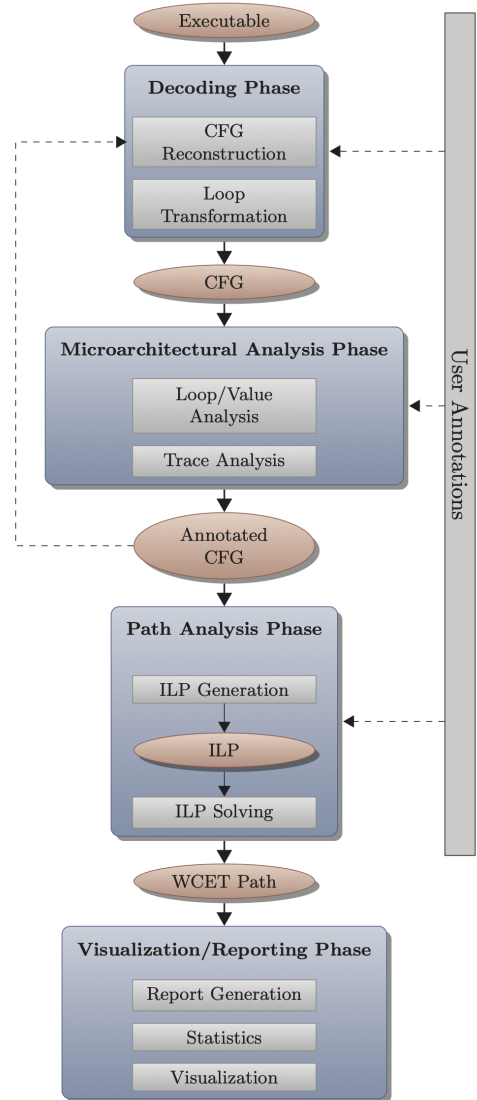


Figure 1: Structure of TimeWeaver. The analysis proceeds in four key stages: decoding, loop/value analysis, trace analysis, and path analysis.

In the decoding phase, the instruction decoder reads and disassembles the input executable(s) into its individual instructions. Architecture specific patterns decide whether an instruction is a control-flow related instruction (e.g., call, branch, return) or just an ordinary instruction. This knowledge is used to reconstruct the basic blocks of the control-flow graph (CFG). Then, the control flow between the basic blocks is reconstructed. In most cases, this is done completely automatically. However, if a target of a call or branch cannot be statically resolved, either the user can write some annotations to guide the control-flow reconstruction, or TimeWeaver can be instructed to extract the targets of unresolved branches or calls from the input traces. To this end there is a feedback loop between the CFG reconstruction and the trace analysis step.

In the next phase, several microarchitectural analyses are performed on the reconstructed CFG starting with the combined loop and value analysis. It determines possible values of registers and memory cells, addresses of memory accesses, as well as loop and recursion bounds. Based on this, statically infeasible paths are computed, i.e., parts of the program that cannot be reached by any execution under the given configuration. This is important because each detected infeasible path increases the trace coverage. Such paths are pruned from further analysis. If the value analysis cannot compute a loop bound or if the computed bound is not precise enough, users can specify custom bounds by means of annotations which are used by the analysis. Loop bounds can also be extracted from the traces.

After value analysis, the analyzer has annotated each instruction in the control-flow graph with context-sensitive analysis results. This context-sensitivity is important because the precision of an analysis can be improved significantly if the execution environment is considered [25]. For example, if a routine is called with different register values from two different program points, the execution time in both situations might be different. Depending on the context settings, this is taken into account leading to higher precision in the analysis result.

In the trace analysis step the given traces are analyzed such that each trace event is mapped to a program point in the control-flow graph. This mapping defines the trace points and trace segments between them and is not only necessary for the whole analysis but also ensures that the input trace matches the analyzed binary. 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.

Afterwards, a CFG which combines the results of value analysis and traced execution timings (both context-sensitive) is available. This graph is the input for the next step, the path analysis phase. Here, the trace segment times alongside the control-flow graph are used to generate an integer linear program (ILP) formulation to compute the worst-case execution path with respect to the traced timings. At this point, the recorded times for each pair of trace segment and analysis context get maximized. The implicit path enumeration technique (IPET) used by TimeWeaver allows to construct WCET estimates for paths that have not been observed themselves during measurements

but are only created during path extrapolation. Thus, not every path needs to be explicitly observed in the traces, greatly reducing the number of measurements that need to be taken.

TimeWeaver also computes the timing contributions of each function and uses debug information to map this information back to the source code. Thus, TimeWeaver allows to have an in-depth look where time is spent and helps to uncover hotspots. These are often unexpected. For example, the accidental use of 64-bit integer division on a 32-bit architecture leads to the inclusion of software routines for this arithmetic operation. Changing the underlying integer type in the source code reduces this overhead.

3.2 Quality of Measurements

Besides the WCET estimate itself, TimeWeaver also gives guidance concerning the quality of the measurements by computing several coverage metrics. In principle, path coverage is the best coverage criterion. Achieving 100 % path coverage means that every possible path through a program has been tested. However, both computing path coverage as well as trying to reach full path coverage is computationally extremely expensive, as there are exponentially many paths for the number of branches in a program. Thus, TimeWeaver employs path extrapolation to reduce the burden of having each path measured at least once. The traces are cut into segments, i.e., the path between two consecutive trace points. These segments may span several basic blocks in the CFG. The ILP formulation of the path analysis allows to construct the longest path in the CFG based on the trace segments even if this path has not been observed directly in the traces.

TimeWeaver computes the following coverage metrics: block/instruction coverage, edge coverage, and flow coverage. The metrics are computed at the machine code level; a mapping to the source code level is available. For each basic block in the CFG, TimeWeaver reports whether it has been covered by measurements and if yes, how often. This information is also used to compute the instruction coverage. Paths for which infeasibility has been proven need no measurements, so associated blocks are excluded while computing coverage. This makes it easy to detect missing tests that are needed to trigger specific execution scenarios.

Checking the number of measurements for each basic block allows to assess the confidence in the measured timings. To support this, TimeWeaver also reports for each trace segment the minimum, maximum, and average observed execution times, plus the standard deviation. The same information is also computed for all traces, making outliers easily detectable. Moreover, loops for which the analyzed worst-case iteration count has not been measured are also reported.

Some basic blocks are reachable from multiple predecessors, so full block coverage does not ensure that each way a block can be reached has been observed. However, this is an important metric for timing analysis as many performance-enhancing features of modern processors take the execution history into account. Hence, TimeWeaver additionally computes the edge coverage and the flow coverage for each block. Full edge coverage means that each possible combination of a block and its

predecessors has been observed. Flow coverage improves on this metric by taking the successors into account, i.e., each possible combination of predecessor \rightarrow block \rightarrow successor needs to be observed to reach full flow coverage. Flow coverage helps to uncover hidden dependencies in the measurements.

Most implementations of the various trace protocols do not emit a trace message for each branch but only if the branch target has been computed, or if the branch history buffer is full. The program flow is captured by recording single bits for taken/not-taken branches. Thus, there might be some control-flow joins in the CFG for which no trace point exists, preventing the path extrapolation at these program points. To aid the path extrapolation, special code patterns can be used to force the ETU to emit a trace message. For example, the ETU can be configured to emit a trace message for each branch-and-link instruction on the PowerPC architecture. Together with the trace point at return instructions, this feature ensures that trace segments do not cross routine boundaries. Another possibility is the usage of lightweight hardware-supported instrumentation to enforce trace points at specific locations.

Many performance-enhancing features like branch prediction, caches, pipelining, etc. take the execution history into account. Thus, the hardware state influences the timing behavior of a code snippet. The longer the observation period, the greater the likelihood of capturing the WCET situations for each segment in the trace. Although it cannot be guaranteed that the WCET situation for each trace segment occurred during tracing, it is much more likely than trying to trigger the WCET situation for the whole task or ISR. Moreover, since the path extrapolation combines maximum trace segment times that might be mutually exclusive in reality, the resulting WCET estimate is usually larger than the maximum observed execution time of that path, adding some kind of safety margin. However, long observation periods result in large trace files. Hence, a balanced approach is favorable. The key here is not to analyze just any trace sequences, but those with relevant anomalies such as particularly long execution times or specific execution prefixes (see Figure 2). How exactly these sequences are identified is explained in Section 4. A virtually unlimited observation period (typically spanning a few hours to a few days)—as provided by Accemic’s CEDARtools—can significantly enhance the statistical relevance and, as a result, increase confidence in the results of the WCET analysis. Thus, we coupled TimeWeaver with CEDARtools, see Section 5.

3.3 Qualification Support

AbsInt provides Qualification Support Kits (QSKs) to assist the automatic qualification of its tools up to the highest criticality levels. The QSKs aim at demonstrating the correct functioning of the tool in the operational context of the tool user with respect to the relevant tool-influencing parameters like options, code constructs, provided external information for the analyzers, etc. The QSK consists of the following parts: specification of the tool functional requirements, test cases and test case procedures, requirements trace data (traceability matrix), test suite and execution framework, and tool lifecycle data. The tool lifecycle data demonstrate development in accordance to safety

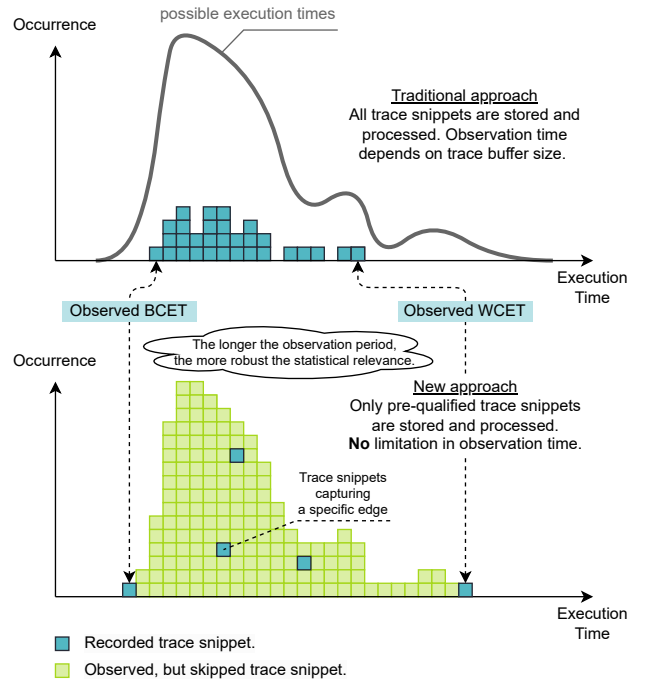


Figure 2: Comparison of traditional trace analysis approaches with the new online analysis implemented by CEDARtools that allows for a virtually unlimited observation period.

standards. The QSKs enable the qualification of TimeWeaver in accordance to domain-relevant safety standards like DO-178C / DO-330, ISO-26262, IEC-61508, EN-50128 / EN-50657, and more. More details on the tool qualification strategy of AbsInt can be found in [17].

4 Embedded Trace

Embedded trace [22] is a method for non-intrusively monitoring processors, providing valuable insights into their program execution at the machine code level. By leveraging *embedded trace*, engineers can gain a deep understanding of processor operations without disrupting the normal execution of software or hardware. This technology is implemented through specific hardware structures that are tightly coupled to the CPU(s) and are used to capture data about the execution of code. The major advantage of using embedded trace is that the embedded trace unit (ETU) does not influence the CPU when monitoring the program flow, i.e., from the application’s point of view it is not possible to tell whether the ETU is active or not.

Figure 3 provides an overview of the individual elements and the trace data flow of a processor equipped with embedded trace. We refer to these elements by their encircled identifiers in the following detailed explanation.

There are several options for transferring the trace data generated by the ETU to an external tool. The trace data can be temporarily stored inside the processor in special embedded trace buffers (ETB) (A) or in the system memory (B). Trace data can also be output via dedicated embedded trace interfaces (ETI) (C1) or via system interfaces (C2) to avoid the limited observa-

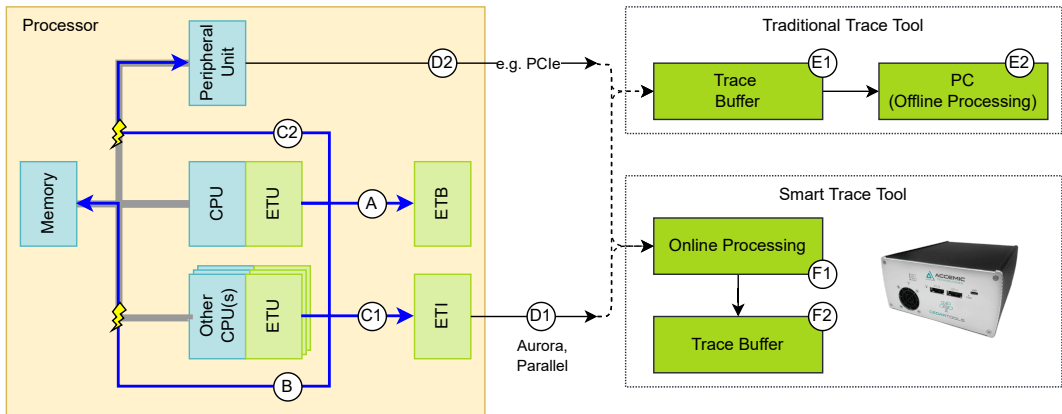


Figure 3: Elements of embedded trace. The target system producing trace data is shown on the left, while the off-chip processing of trace data is shown on the right.

tion time caused by the memory limitations. The direct output of the trace data via ETI is technically the most elegant solution, since it does not interfere with the processor. However, implementing an interface with the required high bandwidth is expensive, so some processor designs compromise by storing the trace data in the main memory (*B*) or outputting it via a system interface (*C2*). However, this is also done at the expense of the desired non-intrusiveness—the required bus operations interfere with the application (indicated by the yellow flash symbols in Figure 3). To allow the use of trace information outside the SoC, trace data is output either directly using dedicated trace interfaces (parallel [20] or high-speed serial like Aurora [31]) (*D1*), or via fast system interfaces (usually PCIe, sometimes USB) (*D2*).

This trace data is then received by an external trace tool. There exist simple trace buffers and advanced smart trace tools. Trace buffers (*E1*) typically contain several GiB of memory where the received trace data is temporarily stored before the decoding and further processing in a PC (*E2*). The disadvantage of this approach is the limited size of the trace buffer, which limits the observation time and thus contradicts the requirement of being able to observe and analyze a system for as long as possible. This problem is addressed by a new generation of smart trace tools, which continuously process incoming trace data (*F1*) and perform real-time reconstruction of the control flow, including timing information. Scalable parallelization with FPGA-based hardware acceleration enables control-flow reconstruction for processors with more than 2 GHz operating clock (for example the Layerscape[®] LX2160A). Control-flow reconstruction is also supported for applications running on multitasking operating systems such as Linux or VxWorks[®], and for applications using dynamically loaded libraries. Based on the live reconstructed control flow and a set of filters, the system can now qualify and store relevant trace snippets in a segmented buffer with a capacity of several GiB (*F2*). This smart architecture has been implemented in CEDARtools, which are presented in Section 5.

A crucial requirement for trace-based WCET analysis is precise timing information about the observed program execution. However, a high-frequency output of timestamps could signifi-

cantly inflate the required trace bandwidth. Furthermore, modern trace protocols like Intel PT, Arm Coresight ETM v4, and Nexus Branch History Trace focus on bandwidth optimization and hence, emit trace data not for each executed instruction, but only for the execution of conditional and computed branches. In addition, it cannot be relied upon to generate a timestamp for every jump in every case, but rather to output a timestamp only after a group of jumps. Some trace protocols support setting a minimum cycle count (Arm[®] Cortex[®]-A53: 4 CPU clock cycles [4]) at which a new timestamp is sent for a branch command. For other protocols, the sending of a new timestamp is linked to the execution of an indirect jump [13]). Depending on the monitoring requirements (accuracy, acceptable intrusiveness) and the capabilities of the embedded trace unit implemented in the processor, a suitable trace strategy can be found for each application. Balancing the capture of sufficient timing information while optimizing trace bandwidth is a critical challenge when utilizing *embedded trace* for WCET analysis.

An enhancement of the *embedded trace* approach, tackling the two issues discussed before, is currently being developed within the TRISTAN project [27]. In a processor-side trace subsystem for the RISC-V architecture, the application is empowered to communicate relevant trigger points to the trace unit through minimally intrusive instrumentation. Ideally, dedicated static analysis tools can automatically identify these trigger points, and the compiler can insert the appropriate instrumentation into the binary code as non-functional instructions. This instrumentation performs deterministic access to special CPU registers, leaving the registers relevant to the application unaltered. From the application’s point of view, it is equivalent to a sequence of `nop` instructions, which consumes only a few CPU clock cycles and does not access the system bus. In most cases, it is therefore justifiable to leave such minimally intrusive and deterministic instrumentation in the release code. The identified trigger points could, for example, involve measuring the time between instruction A and instruction B, storing the trace sequence from 1 ms before A to 1 ms after B for every case where the time between A and B reaches a new maximum and discarding the previously stored trace sequence. This foreknowledge of the code segments relevant for WCET measure-

ments, known at design time, can also be used for precise and bandwidth-efficient control of the output of timing information (CPU clock cycles, wall clock time) by the CPU. Instead of sending timestamps indiscriminately as it was done previously, precise timing information can now be embedded into the trace data stream, ensuring that no redundant information is transmitted while also preventing any gaps in the timing data.

More complex functional tests can be executed on the system traces going beyond simple code coverage or timing measurements. Runtime verification [19] is a formal dynamic method that considers actual runs of a system, and checks properties on streams of events—i.e., system traces—using so-called monitors constructed from high-level specifications. Given such a specification formulated in an appropriate specification language, a monitor is synthesized that runs in parallel to the execution of the system. It accepts exactly the traces of the system adhering to the specification. All other executions are identified as failures. Some level of system resilience can be achieved by so-called runtime reflection [19], which aims to devise mitigation actions in the case of failures to restore some of the system’s functionality. This is particularly useful for safety-critical systems for which no safe state exists, and thus, being fail-safe is not sufficient but being fail-operational is necessary. The same techniques can also be used to implement complex trigger conditions for trace recording.

The smart trace approach has been implemented in the form of software tools and specialized hardware, and demonstrated in a number of pilot applications. Theoretical and practical aspects of it have been described in several previous publications [22, 15, 29, 6, 8].

5 TimeWeaver/CEDARtools Coupling

Nowadays, in many recent computing platforms for embedded systems, trace data of the program execution is provided by the target system via dedicated, often already existing processing interfaces. Existing trace tools are logging such trace data in a file for offline analysis. This allows to analyze the cause of complex error patterns even after a system has been released. However, this allows only post-mortem analysis and due to the sheer data volume of traces (several GiB/s even for medium-sized processors), the time span that can be observed is limited.

As an alternative, we present with CEDARtools [2] an approach for dynamic analysis and inspection of embedded systems that is based on the idea of on-the-fly analysis of trace data at run-time. We argue that the approach offers several advantages compared to existing static and dynamic analysis methods. The trace analysis capability can be fully separated from the target system; in our case, through specialized high-performance hardware (FPGAs) that can keep up with the speed of trace generation. This means that no instrumentation of the system under scrutiny is necessary, and its original, unaltered behavior can be observed (non-intrusiveness). This is especially important for precise timing analysis in real-time applications, and for analyzing systems with non-deterministic compute architectures and parallelism. In addition, in our solution the on-the-fly processing of traces obliterates the need

for storing (most or all of) the trace data¹, thus enabling long-term or even continuous monitoring to catch also sporadic rare events that are otherwise hard to track down. With a capacity of several GiB, the trace memory of CEDARtools is large enough to store relevant sequences for evidentiary purposes. Both simple and complex triggers can be defined in a high-level specification language [15] and make it possible to precisely identify relevant parts of the trace data stream. Hence, only those relevant parts of the trace stream need to be stored in a (partitioned) ring buffer. For example, each time a new maximum execution time a task is observed, the corresponding trace data replaces the old measurements, ensuring that a trace of the currently observed worst-case scenario is stored for later offline analysis. Furthermore, the efficiency of the analysis is significantly increased by monitoring not only one point of interest, but up to 32 of them simultaneously.

The coupling between TimeWeaver and CEDARtools is depicted in Figure 4. It consists of three main components: the target system on the left, the FPGA-based CEDARtools trace box in the middle, and the offline hybrid timing analysis on the right side of the figure.

The binary executable of the software under analysis is loaded onto the target system. The test engineer decides which parts of the software system are of particular interest for the timing analysis, for example tasks or ISRs. As part of a test campaign, they are executed inside a test harness that generates the necessary input stimuli to trigger the intended behavior under scrutiny (unit test / integration test). While the target system is running, trace data is emitted via the trace port of the target system. Attached to this port is the CEDARtools trace box.

The trace box processes the incoming trace data at runtime. The highly compressed trace stream is decoded inside a FPGA to reconstruct the control flow of the software system under observation. The processing speed matches the execution speed of the target system to enable the live monitoring of the software system. The trace box allows virtually unlimited observation periods since only those parts of the trace stream are stored for later offline processing that match predefined criteria. The criteria are given by the test engineer. One possibility is the collection of bad-case scenarios for the points of interests (i.e., the tasks and ISRs of the system). Here, the internal ring buffer of the trace box is partitioned in up to 32 slices. Each slice can store the trace of one observation of a task/ISR together with the observed execution time of this instance. If the currently running instance of the task/ISR exceeds the previous maximum observed execution time, the currently running instance replaces the one previously stored in the slice of the ring buffer. Another possibility is the collection of traces for specific trigger conditions, for example only after a specific combination of tasks has been executed. The trace data stored in the ring buffer can be exported for further offline processing or as a witness for specific execution scenarios.

Finally, the exported trace files are used in a hybrid timing analysis of the software system. Here, the binary executed on

¹Certification of safety-critical systems may, however, sometimes require evidence that is documented in the form of traces. Hence, CEDARtools allow to export selected traces.

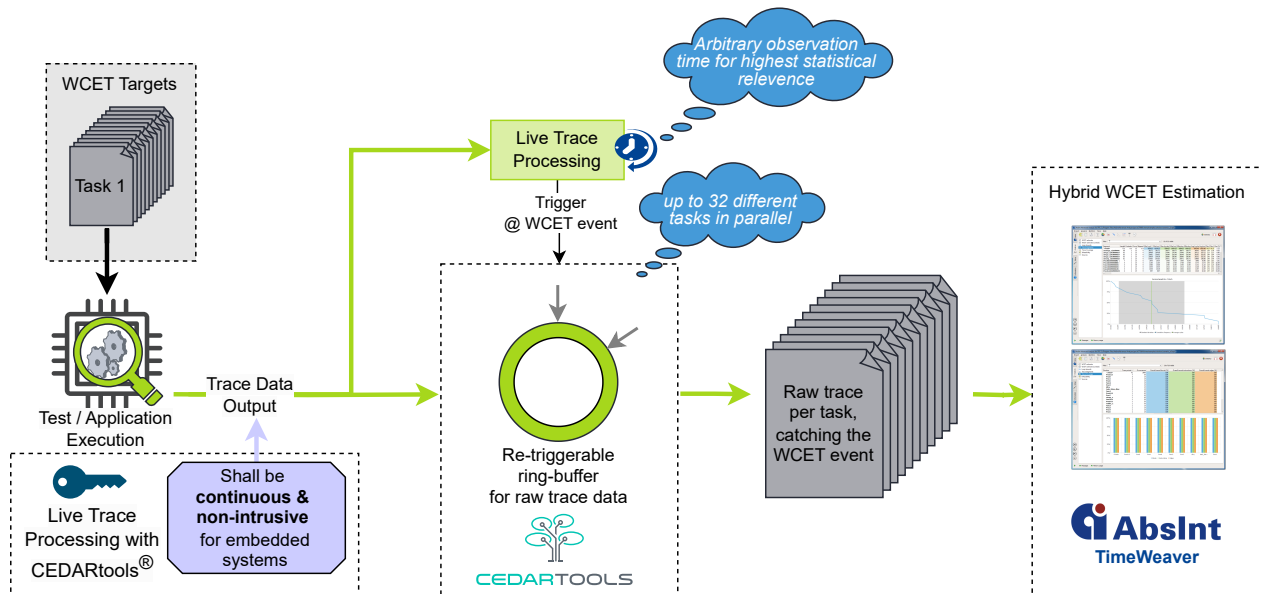


Figure 4: Coupling of CEDARtools and TimeWeaver for hybrid WCET estimation of multicore systems.

the target system, the trace files, and the entry points of the tasks and ISRs that are of particular interest are the input for TimeWeaver. The inputs are processed as explained in Section 3. Path extrapolation during the static path analysis phase of TimeWeaver constructs a critical path from the trace segments obtained via the CEDARtools trace box. Thus, the coupling will find the worst possible path through the CFG even if it has never been observed directly. The result of the hybrid timing analysis is an estimate of the WCET, together with the visualisation of the critical path and statistics for the various trace segments used to compute the estimate. One example for a timing analysis result is depicted in Figure 5. In case the coverage report of TimeWeaver uncovers code regions that have less than the required percentage of instruction, edge, or flow coverage, CEDARtools can be instructed to save exactly those trace sequences that cover the code region in question.

We argue that the targeted selection of trace snippets within an arbitrarily long observation period has a higher statistical relevance than the recording of trace snippets that are randomly located within a limited observation period. The analysis of several avionics applications, which we are not allowed to publish, has confirmed this assumption. To demonstrate how the method works, a sample application was developed with a random Gaussian-distributed execution time. As expected, we obtained a realistic WCET estimate with the presented methodology which is 52% higher than when using a randomly selected full trace sequence within a limited observation period. However, these results cannot be generalized and are highly dependent on the application under investigation.

6 Conclusion

For multi-core systems, the main challenge for WCET analysis is the interference generated by other cores running in parallel. If the platform provides no robust resource partitioning and robust time partitioning, static WCET guarantees are unrealis-

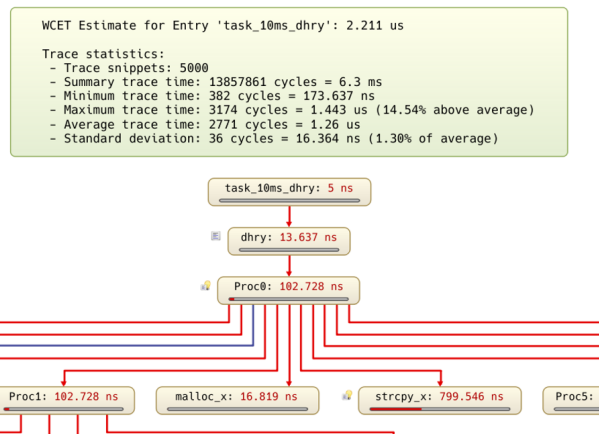


Figure 5: Result visualisation of a WCET analysis using the TimeWeaver/CEDARtools coupling for the NXP LX2160 processor, highlighting the extrapolated critical path that leads to the WCET estimate in red. The statistics include the observed BCET (“minimum trace time”) and the observed WCET (“maximum trace time”).

tically pessimistic. In this article we have presented a hybrid WCET analysis that combines static value, loop, and path analysis with non-intrusive measurements to compute interference-aware WCET bounds and provide feedback on the trace coverage obtained. Key to reliable hybrid WCET estimation is the ability to observe a processor in detail over arbitrary time periods, with no or at most exceedingly minimal instrumentation. This is achieved by embedded trace units usually already implemented in all processors in combination with new powerful live control-flow analysis tools. Our approach has been implemented by a coupling and extension of the tools TimeWeaver and CEDARtools. It is compliant to the software verification requirements of the EASA AMC 20-193 guidance.

References

- [1] AbsInt Angewandte Informatik GmbH. TimeWeaver. <https://www.absint.com/timeweaver>.
- [2] Accemic Technologies GmbH. CEDARtools. <https://accemic.com/cedartools/>.
- [3] ARM Ltd. CoreSight™ Architecture Specification v2.0, 2013. ARM IHI 0029B.
- [4] ARM Ltd. Arm Cortex-A53 MPCore Processor Technical Reference Manual, 2018. ARM DDI 0500J.
- [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*, 13(4):82:1–82:37, 2014.
- [6] L. Convent, S. Hungerecker, T. Scheffel, M. Schmitz, D. Thoma, and A. Weiss. Hardware-based runtime verification with embedded tracing units and stream processing. In C. Colombo and M. Leucker, editors, *Runtime Verification - 18th International Conference, RV 2018, Limassol, Cyprus, November 10-13, 2018, Proceedings*, volume 11237 of *Lecture Notes in Computer Science*, pages 43–63. Springer, 2018.
- [7] C. Cullmann, C. Ferdinand, G. Gebhard, D. Grund, C. Maiza (Burguière), J. Reineke, B. Triquet, S. Wegener, and R. Wilhelm. Predictability Considerations in the Design of Multi-Core Embedded Systems. *Ingenieurs de l'Automobile*, 807:26–42, 2010.
- [8] N. Decker, B. Dreyer, P. Gottschling, C. Hochberger, A. Lange, M. Leucker, T. Scheffel, S. Wegener, and A. Weiss. Online analysis of debug trace data for embedded systems. In J. Madsen and A. K. Coskun, editors, *2018 Design, Automation & Test in Europe Conference & Exhibition, DATE 2018, Dresden, Germany, March 19-23, 2018*, pages 851–856. IEEE, 2018.
- [9] EASA. General Acceptable Means of Compliance for Airworthiness of Products, Parts and Appliances (AMC-20) – Amendment 23. AMC 20-193 Use of multi-core processors, 2022.
- [10] 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.
- [11] Freescale Semiconductor, Inc. QorIQ™ P4080 Communications Processor Product Brief, Rev. 1, 2008. http://cache.freescale.com/files/32bit/doc/prod_brief/P4080PB.pdf.
- [12] S. Hahn, M. Jacobs, and J. Reineke. Enabling compositionality for multicore timing analysis. In *Proceedings of the 24th International Conference on Real Time and Networks Systems*, October 2016.
- [13] IEEE-ISTO. IEEE-ISTO 5001™-2012, The Nexus 5001™ Forum Standard for a Global Embedded Processor Debug Interface, 2012.
- [14] Infineon Technologies AG. Debug Support – AURIX™ TC2xx Microcontroller Training V1.0. https://www.infineon.com/dgdl/Infineon-AURIX_Debug_Support-Training-v01_00-EN.pdf?fileId=5546d46269bda8df0169ca5c0d9a2537[retrieved: January 2021].
- [15] H. Kallwies, M. Leucker, M. Schmitz, A. Schulz, D. Thoma, and A. Weiss. Tessla - an ecosystem for runtime verification. In T. Dang and V. Stolz, editors, *Runtime Verification - 22nd International Conference, RV 2022, Tbilisi, Georgia, September 28-30, 2022, Proceedings*, volume 13498 of *Lecture Notes in Computer Science*, pages 314–324. Springer, 2022.
- [16] D. Kästner, L. Mauborgne, S. Wilhelm, C. Mallon, and C. Ferdinand. Static Data and Control Coupling Analysis. In *11th Embedded Real Time Systems European Congress (ERTS2022)*, Toulouse, France, June 2022.
- [17] D. Kästner, M. Pister, and C. Ferdinand. Obtaining DO-178C Certification Credits by Static Program Analysis. In *ERTS2022*, Toulouse, France, June 2022.
- [18] 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 *Open Access Series in Informatics (OASISs)*, pages 1:1–1:11, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.
- [19] M. Leucker and C. Schallhart. A brief account of runtime verification. *J. Log. Algebraic Methods Program.*, 78(5):293–303, 2009.
- [20] MIPI Alliance, Inc. MIPI PTI v2.0 Specification for Parallel Trace Interface, May 2011.
- [21] J. Nowotsch, M. Paulitsch, D. Buhler, H. Theiling, S. Wegener, and M. Schmidt. Multi-core interference-sensitive WCET analysis leveraging runtime resource capacity enforcement. In *26th Euromicro Conference on Real-Time Systems, ECRTS 2014, Madrid, Spain, July 8-11, 2014*, pages 109–118. IEEE Computer Society, 2014.
- [22] T. B. Preußner, S. Gautham, A. D. Rajagopala, C. R. Elks, and A. Weiss. Everything you always wanted to know about embedded trace. *Computer*, 55(2):34–43, 2022.
- [23] K. Schmidt, D. Marx, J. Harnisch, A. Mayer, U. Dannebaum, and H. Christlbauer. Non-Intrusive Tracing at First Instruction, 2015. SAE Technical Paper 2015-01-0176.
- [24] A. Schranzhofer, J.-J. Chen, and L. Thiele. Timing predictability on multi-processor systems with shared resources. In *Workshop on Reconciling Predictability and Efficiency at EMSOFT 2009*, 2009.
- [25] S. Stattelmann and F. Martin. On the Use of Context Information for Precise Measurement-Based Execution Time Estimation. In B. Lisper, editor, *10th International Workshop on Worst-Case Execution Time Analysis (WCET 2010)*, volume 15 of *Open Access Series in Informatics (OASISs)*, pages 64–76. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2010.
- [26] V. Suhendra and T. Mitra. Exploring locking & partitioning for predictable shared caches on multi-cores. In *Proceedings of the 45th annual Design Automation Conference, DAC '08*, pages 300–303, New York, NY, USA, 2008. ACM.
- [27] TRISTAN. Together for RISC-V Technology and Applications. <https://tristan-project.eu>.
- [28] S. H. VanderLeest and S. R. Thompson. Measuring the impact of interference channels on multicore avionics. In *2020 AIAA/IEEE 39th Digital Avionics Systems Conference (DASC)*, pages 1–8, 2020.

- [29] A. Weiss, S. Gautham, A. V. Jayakumar, C. R. Elks, D. R. Kuhn, R. N. Kacker, and T. B. Preußner. Understanding and fixing complex faults in embedded cyberphysical systems. *Computer*, 54(1):49–60, 2021.
- [30] R. Wilhelm, D. Grund, J. Reineke, M. Schlickling, M. Pister, and C. Ferdinand. Memory hierarchies, pipelines, and buses for future architectures in time-critical embedded systems. *IEEE Transactions on CAD of Integrated Circuits and Systems*, 28(7):966–978, 2009.
- [31] Xilinx, Inc. Aurora Protocol Specification, Sept. 2007.

Session We.4.A
**Autonomous System & Digital
Twins**

Wednesday 12th June

15:00

–

Auditorium

Digital twin for embedded software. State of art in industry and deployment at AMPERE ePowertrain

Jean-Marie Quelin
AMPERE ePowertrain Systems & tuning
AMPERE vehicle (Renault Group)
Lardy, France
jean-marie.uelin@renault.com

Christian Becker
Production Software and SIL simulation
dSPACE GmbH
Paderborn, Germany
cbecker@dSPACE.de

Abstract — Virtualization is a technology that has evolved over the last ten years. The solution reaches a level of feature that fits to the new needs of the automotive industry. The paper presents the state of the art and the return on experience of AMPERE ePowertrain Team by showing the deployment of SIL and HIL on several use cases.

Keywords—*virtualization, embedded software, virtual ECU, electric powertrain control, HIL, SIL*

I. INTRODUCTION

More and more customer features are involving software in so-called mechatronics systems. There are hundreds of electronic control units in a transport vehicle. Safety regulations are more stringent and lead to more and more tests for validation or certification. Distributed architecture with several ECUs per domain is moving towards more centralized architectures consisting of high-performance computers (HPCs), zonal and edge ECUs. Software update over the air, embedded & cloud services, add complexity to the system and increase embedded software content.

Validations are taking more and more time on such complex systems. Being able to make some validation earlier in the development is a key point to respect delivery pace and quality in the field.

This paper will present the state of art of V-ECU for SIL simulation from the point of a tool vendor. While hardware-in-the-loop (HIL) simulation is a well-established process to test and validate ECU functions, the following chapters refer to the usage of virtual testing solutions, where a lot of dynamic can be observed in the market. Software virtualization is a technology that has evolved within more than a decade of existence, the latest technical evolutions and maturity of the solutions can answer automotive industry disruptive challenges. The Powertrain department of Renault, now AMPERE ePowertrains, that was one of the pioneers to use the virtualization concept, will in a second part of this paper focus on how virtualization can be used in software development and validation of ePowertrains. After presenting the return on experience, the conclusions will propose guidelines for further developments.

II. STATE OF THE ART

A. Motivation

Software-in-the-loop (SIL) is a major trend around test and validation of mechatronic systems. To accelerate the software development, software integration steps and quality assurance, a PC-based simulation is used to find errors earlier in the development process. Due to the increasing complexity of E/E architectures, the System-under-test (SUT) shall include distributed and already integrated software functions, so that it's no longer necessary to wait for the integration on the real hardware. There are different levels of integration, where a SIL simulation can help: during function development or ECU integration, or before integrating a system with multiple networked ECUs.

Another major advantage of PC-based SIL simulation is the scalability. Without the need of dedicated hardware resources, SIL simulations can run in parallel and especially in cloud environments to flexibly cover the demands for test execution and allow fast feedback loops to improve the software.

It is important to note, that SIL testing is one of several test environments, which shall complement each other, see [1] for description of test methods and test environments. Especially the Hardware-In-The-Loop (HIL) testing environment is going to be extended by SIL testing in requirements-based testing, because many steps in the testing process can be applied either for SIL or HIL in a similar manner, and artefacts can be partly re-used.

B. V-ECUs and ECU Networking

This chapter explains a classification of V-ECUs in different levels and a Layered standard for V-ECU based on FMI to address the challenges for the availability of V-ECUs. The creation of V-ECUs means, that the software-under-test is prepared for execution in a virtual environment, e.g. by replacing the code, which accesses the microcontroller hardware or its periphery.

1) V-ECU Levels

Since the usage of V-ECUs at different stages of development is of common interest, different levels have been defined by Prostep IVIP, see [2]. Here are the following levels of V-ECUs are defined:

- **Level 0 V-ECU (Controller Model):**
Can consist of a model, e.g. a MATLAB Simulink model, or code generated from it, which is not yet on production code level. Used to test control algorithms in an early development stage.
- **Level 1 V-ECU (Application Level):**
Production application software can be integrated into a Level 1 V-ECU. Only a minimum set of BSW is included to execute the code, e.g. in case of AUTOSAR the OS and RTE is needed.
- **Level 2 V-ECU (Simulation BSW):**
Before the production BSW integration took place, it may be required to add BSW to the V-ECU for simulation purposes. Most common use cases are adding a COM-stack to the V-ECU for communication via an automotive bus system, or adding BSW for diagnostics, to prepare and execute diagnostics tests.
- **Level 3 V-ECU (Production BSW)**
When the integration of the production BSW is done, e.g., by the ECU supplier, a Level 3 V-ECU helps testing this integration without the need of real hardware and an existing HIL setup. The HW-dependent parts of the BSW need to be replaced, e.g., the AUTOSAR MCAL Layer.
- **Level 4 V-ECU (Target Binary)**
A Level 4 V-ECU contains the production code for the real target ECU. An instruction set simulation (ISS) is required to execute this type of V-ECU, i.e. that the machine code is simulated based on a simulation model of the target hardware on register-level, see [3]. Here no cut in the SW-Stack is necessary, and all hardware accesses can remain inside the V-ECU. To some extent performance tests can be done, but a major trade-off is, that setting up an ISS can be expensive, and the simulation speed is much slower compared to Level 0-3 V-ECUs, which can be cross-compiled for the simulator platform.

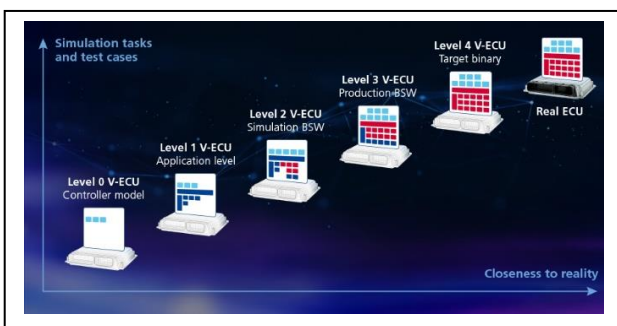


Figure 1 : Prostep ECU's level definition

A standard like AUTOSAR helps a lot to create Level 1-3 V-ECUs, since potential code interfaces are explicitly defined, e.g., the micro controller abstraction layer. This way the integration of the ECU software can be tested at the ECU supplier in an early phase and the OEM can validate, that the ECU software is working correctly in a larger scope with

environment models or further ECUs. Depending on the integration target, the appropriate level of V-ECU can be selected. The V-ECUs as SIL described in this article are fitting to the Level 0, 1 and 2.

2) FMI-Layered Standard for V-ECUs

In the future it will be possible, to generate and exchange Level 1 to Level 3 V-ECUs in a standardized way, so integration costs can be reduced and coordination between OEMs and suppliers will be more efficient. The usage of FMI for integrating C Code into different testing environments like SIL and HIL, has been put into operation, see [4]. The FMI 3 standard allows the definition of a layered standard, which is currently under development, see [5]. With this standard, the following functionality will be available:

- Usage of XCP service inside the V-ECUs
- Integration of communication stack and simulation of the behavior of automotive busses (CAN, LIN, Ethernet and FlexRay)
- Usage of different abstraction levels for network communication. A cut on physical signal level ("High-cut"; cut below COM module), or on a network level ("Low-cut"; cut on Driver-Level) can be applied, while a Low-cut enables a more accurate bus simulation on a Pdu-level.

Once the FMI Layered standard for V-ECU is available and the tools support the creation as well as the integration of standardized V-ECUs, the extensive availability of V-ECUs can greatly facilitate the introduction of SIL in future testing strategies.

C. Development Processes

There is a variety of processes for developing automotive software, which can range from typical, traditional V-Cycles approaches towards more agile approaches, see [6]. Model-based development (MBD) is well-established, but also C/C++ Code development is done manually. An important goal is to find errors as early as possible and be able to investigate the behavior of the software in an early phase.

What is common to all approaches is, that the software is tested within its development environment, where module-tests can be executed by stimulation of the inputs and validation of the behavior of the software function.

The integration of a software module into a larger scope to test the interaction with other software parts or how it behaves in a closed-loop simulation, is a familiar approach in automotive industry, and used more and more frequently. As integration tests are executed on SIL, before having the final software on real hardware, errors are found and fixed earlier before the building of the real software.

D. Environment Models & Test Environments

Since the availability of V-ECUs is one important building block for establishing SIL testing, the environment models for closed-loop SIL testing need to be integrated as well. This highly demands an openness of a SIL simulator to be connected to any 3rd-party simulators, so that existing models and virtual execution platforms can be reused and need not to be reimplemented. But if it is possible to implement the environment model natively on the simulation platform, this has the advantage higher execution speed.

A last step towards a complete test environment is the connection of experimentation and test execution frameworks to the SIL simulation. Here is a high demand for re-using established testing environment as well as the tests. This is supported by established standards e.g., for test automation or measurement and calibration defined by the ASAM organization, see [7]. Accessing devices under test with an XCP Service by using A2L-Files is defined by the ASAM standards MC-1 XCP and MCD2-MC, and executing tests based on standardized API is provided by the XIL standard. These standards are state-of-the-art in MIL, SIL and HIL, and are an important basis for the overall solution to enable SIL and the usage of digital twins (software and environment) in closed loop.

III. THE SHIFT TO SDV

Future ECU networks need to fulfill requirements, which are often summarized under the term “software-defined vehicle” (SDV), e.g., to enable over-the-air updates (OTA) so that the features of a car can be defined and upgraded by software. Such architectures are more centralized and comprise high-performance computers (HPCs), where the applications are executed on POSIX-based operating systems. A scalable, cloud-based simulation for such POSIX-based applications is also important.

But the shift towards SDV does not only imply new hardware architectures and software platforms; there are 5 major game changers for the future automotive product-development, see [8]:

- From waterfall to agile processes.
- From distributed, embedded software to a centralized electrical and electronics architecture.
- From start of production to life cycle economics.
- From product tech to customer experience.
- From engineering-driven to data-driven development.

Distinguishing features of the cars are already realized by software for many years, but with the shift towards SDV this becomes even more important. Development cycles of software from its specification to the over-the-air update in the car need to be heavily increased e.g., to meet future requirements for agile development. This requires continuous integration (CI) processes, where software developers can integrate their code frequently and automatic tests detect integration errors promptly. Evolutions of software are expected to be available on a daily (weekly) basis to be integrated in the vehicle simulation platform and tested using automatic tests.

This leads to the trend to connect development environments to the simulation platforms, e.g., by directly and automatically integrating the code into a V-ECU or by coupling the simulation environment of the software developer with a simulator, which integrates further parts of the system, e.g., environment / restbus models or further V-ECUs.

One of the challenges for SDV is to be able to validate new features added at the launch of the car, and allow updating and bufixing features during its life cycle, where homologation plays an important role.

Offboard services can be added to the vehicle simulation platform and their interactions with inboard service tested. There is no need to have the real offboard services available, only the data flow model is modeled and faults can be easily injected.

As the development of SDV involves partnership making the integration of software using virtual ecu is a key answer to the challenges.

Before creating this simulation platform at the level of the car and reflecting the E/E architecture, each individual V-ECU must be available and tested in a simulated environment.

IV. IMPLEMENTATION AT AMPERE EPOWERTRAIN

A. Context

Since first implementation, described in [9] AMPERE ePowertrain decided to review their objectives. The previous requirements are now extended with focus on synergies between SIL and HIL, the compatibility with AUTOSAR, the usage of production C code, the possibility to extend content of basic software, addressing potential users and support of ASAM MCD standard for calibration to allow usage of existing tool in tuning team.

The following description of our development process is the current one and is not changed with the introduction of SIL testing.

AMPERE buy the ECU hardware and corresponding basic software from a supplier. AMPERE team develop the applicative software using Model Based Design approach (Simulink) and generate the corresponding production C code. Generally, one Simulink file (called specification) corresponds to one C module. The supplier integrates applicative software code to basic software code and deliver a binary that is flashed on the ECU.

Simulink specifications (that are the Level 0 V-ECU described by Prostep) are tested individually, this is model-in-the-loop (MIL) validation. Specifications are assembled to compose a full applicative software and tested in a Full MIL platform. The main drawbacks are the assembly of specifications and not production code and simulation time. When coupled to a plant model, the simulation runs 10 to 100 slower than real time for a hybrid controller. Only automated tests can be executed as interactive session is not possible.

The work of integrating thousands of specifications in Full MIL is an additional effort. Signal naming issues are caught, and specification need to be updated, when issues are found. This integration is done at the level of the specification, not production C code.

Issues related to applicative software, under AMPERE responsibility, are discovered by the supplier during the integration phase or during testing by OEM on HIL. Several loops are needed between OEM and Tier#1 to solve software issues. Typical issues are task order in a same scheduling, datatype mismatch, task not triggered by the good scheduling or naming of signal between codes and bus specification. As the integration is done with specifications and not production C codes, those issues are not found before shipping the codes to the supplier.

The target software, shipped by the supplier, is tested on hardware in the loop bench. The HIL bench needs to have the real software and real hardware to finalize the HIL plant

model. Test activities cannot start until real software is received.

B. Deployment

We implement a digital twin of the applicative software (so called Virtual ECU, V-ECU), which can be used all along the development and validation cycle of software to solve some of the previously described drawbacks.

1) Process

During the framing phase, the content of SIL simulation platform is defined to fit the need of validations and the ports of plant models to match corresponding ports of V-ECU.

Two flows can start in parallel: building the plant model and the V-ECU.

As the functional part of the plant model is common for SIL and HIL, it must be compatible with real time execution. This ensure that SIL is also real-time compatible.

The build of V-ECU starts with the list of codes, the order of execution of the functions and the list of ports.

When plant model and V-ECU are available, they are connected to release the simulation platform.

After verification tests ensuring the expected behavior, the simulation platform is released.

2) Level of V-ECU

Using Level 1 V-ECU (100% of applicative software) is the first step and already deployed on several AUTOSAR projects. Level 2 V-ECU (adding COM stack and Diag stack) is planned in the second semester.

Adding a model of CAN communication (code generated by the SIL toolchain, not production code) between the plant model and the Level 1 V-ECU is also an alternative to Level 2 V-ECU with CAN. In this case, this model can be reused for HIL plant model. The CAN message can be manipulated on SIL in the same manner as on HIL, so the CAN HIL test can be prepared on SIL and executed on HIL. If CAN failure and diagnostic tests are not yet feasible on SIL, those tests must be done with production basic software and hardware.

3) Tool continuity for SIL and HIL

Using the same dSPACE building and experiment toolchain for SIL and HIL allow to share proprietary artefacts. Plant model exported as SIC (Simulink Implementation Container) is used for SIL and HIL without modification. CAN model exported as BSC (Bus Simulation Container) is used for SIL and HIL. Experiment tool (ControlDesk, INCA) for manipulating or visualizing V-ECU and plant model data are the same, configuration done for SIL are reused on HIL. The additional effort for deploying SIL is therefore limited. The gains are reduced license cost, manpower by not redeveloping similar artefacts and training as using same tools.

4) Plant model

The preferred format for plant model is FMU. Simulink models are exported as SIC (Simulink Interface Container) or FMU. AMESim multiphysical models are exported as FMU.

Simulink or AMESim plant model are validated by comparing the results with real data recorded on vehicle or benches when available.

The plant model must be capable of real time execution on HIL, with fixed step time.

The complexity of the model depends on the tests done on the simulation platform and on the expected acceleration factor. For the moment the target for SIL software validation is to be at least real time compatible for the assembly plant model plus SIL software. Our experience for HEVC controller is that the bottleneck is the plant model not the V-ECU.

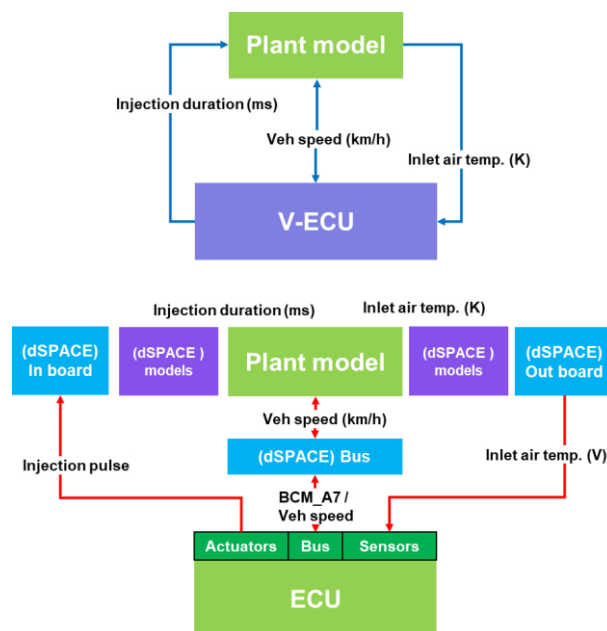


Figure 2 : Setup of SIL-Simulation and HIL-Simulation System

5) Two V-ECU formats

We provide V-ECU in two formats: dSPACE proprietary and FMU. These two formats correspond to two different usages, that are explained hereafter. Having a V-ECU standard compatible with the two usages would avoid this double delivery.

a) Proprietary dSPACE V-ECU

VEOS is the simulation platform executing the V-ECU and the plant model. ControlDesk is the experimentation tool: user changes V-ECU calibrations, plant model parameters, visualizes V-ECU internal and plant model variables, records signals, defines and runs stimuli.

The users are software designer and validation engineers of AMPERE ePowertrain division.

The objective is to maximize the synergies with HIL validation as one of the objectives is to move some HIL tests to SIL. Validation engineers have same user interface and can switch easily from HIL to SIL. Measuring tool used on HIL are used on SIL.

HIL automatic tests using ControlDesk and INCA test can be used on SIL, so same test can be executed on both SIL and HIL. Details on automatic tests toolchain is given in chapter 6) Test automation.

Access to internal variables or tunable parameters of V-ECU is done using XCP server. ControlDesk is the tool to interact with software and plant model, using XCP server. Record files are using ASAM MCD standard [4].

b) FMU V-ECU

The targeted audience is simulation platforms using ePowertrain software models as Simulink, the V-ECU FMU replaces the Simulink models. The users are other divisions of AMPERE or Renault Group and the perimeter of simulation platforms are either system or full vehicle E/E architecture. Tests consist of integration of ECUs and checking driver scenario involving several ECUs.

No license is required for executing the V-ECU FMU.

V-ECU is exported as FMU 2.0 Co simulation. The FMU is provided as compiled FMU without the source code, ensuring integrity of content.

The exposed internal variables, parameters and ports can be set by editing one single XML file (modelDescription.xml) bringing a first level of 'black box'.

An XCP server is embedded, allowing access to internal variables and calibration of software.

FMU 3.0 will be evaluated when the layered standard for CAN will be available.

6) Tests automation

Current HIL automatic test toolchain is based on Python, and Excel. The test is written using 'keyword' with parameters in an Excel sheet. Each keyword is mapped to 'Python keyword code' to interact with the experiment or measurement tool API. The Excel file is parse line by line and the keyword executed, and a result is written back in the Excel file. A specific keyword allows analysis of record and send back a status.

Using same experiment tool (ControlDesk), same measurement tool (INCA) allows usage of same keyword and same 'Python keyword code' for SIL. Using same plant model, same A2L file makes existing tests for HIL usable on SIL in real time without rewrite, when the test is executed on SIL.

Next step will be to develop automatic tests on SIL.

V. USE-CASES

1) Using V-ECU during software development phase

When a set of new specifications is developed and validated on MIL, the corresponding production C code is integrated in a V-ECU representing the first software loop embedding these codes. Objective is to send to the Tiers#1 a set of validated and assembled production C codes of applicative software. A first set of calibrations is tested and published with C codes and ready to be used on HIL. This is deployed since 2023 on an EV controller software planned for a mass production in 2025.

Two production software for an EV platform planned to be on production in 2025 are using V-ECU for their development: EV Controller (non AUTOSAR) and BMS Controller (AUTOSAR). A V-ECU is built each time a software is requested to the Tier#1. The V-ECU is available as proprietary (dSPACE) and FMU format.

2) Reduce the number of mockups

Software mockups are sometime created to test new features on cars without requesting the integration by the Tier#1 on production ECU.

V-ECU reduces the number of mockups as the new feature is tested and verified on V-ECU. The development is very short and managed by the software designer if a V-ECU is used as basis for integrating the evolution. There is no need of a vehicle or engine bench at this first stage.

When the evolution is mature and calibration available, a real mockup is built and tested and validated on bench or vehicle. The mocked is delivered when it's usable by other team, they don't see the iterations.

3) Evaluate software achitecture

V-ECUs is used for validating new software architecture. Refactoring of specifications, simplification of specification, different schedules can be tested and pre-integrated on V-ECU without the need to request a full software to Tier#1. The benefits or the degradation are evaluated by comparing results of two versions.

The merge of two ECUs is tested by building a joined V-ECU based on existing production C codes. The software components of the two previous ECU are assembled and integrated in one new software. The behavior of resulting software is evaluated without the real hardware. The integration and tests are done in parallel of the development of the new hardware platform and corresponding basic software. The objective is to get a usable software from the first assembly of applicative and basic software. A V-ECU is started for a Powertrain Domain Controller composed of EV, BMS, OBC controllers with objective to get a first functional applicative software when the first basic software and hardware will be available.

4) Replacing Matlab Simulink model by V-ECU

Simulation platforms at the level of the vehicle need model of ECU. When software specifications are available as Simulink, they are used as model of the software. There is a tradeoff between the fidelity of the software model (number of specifications) and simulation time to reach at least real time to allow interactive test session as on vehicle or HIL. This simplification of the content is an extra activity and needs specific skills.

Replacing Simulink specification by a Level-1 V-ECU brings a faster simulation time (around ten to one hundred time for an HEVC controller with ~1500 specifications running on a PC with Intel(R) Xeon(R) W-2223 CPU @ 3.60GHz). Internal calibrations and variables are accessible the same way as on real ECU using xCP. The simulation time is not impacted by the number of internal variables recorder as observed in [9].

When delivering a V-ECU as FMU, the corresponding plant model needed for the integration of the V-ECU in the final platform is provided. Records of V-ECU outputs and corresponding stimuli on inputs are delivered showing the expected behavior of V-ECU.

A simulation composed of HEVC controller plus plant model, for a driving cycle of 100 s taking one hour and half can now be run in real time.

V-ECU as FMU for two software loops (HEVC, Inverter controller) have been built and are added to Simulink platform. The target is to propose a V-ECU FMU, for each

official software loop, for simulation platform needing models of ePowertrain software.

5) *Preparing HIL tests*

As now the experiment toolchain is the same for SIL and HIL, the validation engineer writes the test sheet on SIL. Measurement tool (INCA, ControlDesk) are configured on SIL, and same configuration is used on HIL.

Calibration of new feature software are prepared and tested on SIL. A new calibration set is exported as CDFX [10] and can be reused in a calibration data lifecycle management tool like CRETA or INCA to be reused on HIL or vehicle.

This allows an earlier start of validation on HIL when the real software is received. This phase is a way of showing the potential of SIL validation to engineer used to validate on HIL.

6) *Validation synergies*

All functional tests where the ECU hardware or basic software is not involved can be done on V-ECU. Propagation of signal, state flow, switch are good candidates. Functional integration tests are done on V-ECU.

All validations related to basic software or ECU hardware will still be done on HIL, as only the production software will include the basic software accessing the production hardware. Actual SIL simulation does not support core allocation and is not representative of task duration.

VI. RETURN OF EXPERIENCE / IMPROVEMENT

1) *Skills*

Control engineers using Simulink may not have skill to build V-ECU based on C code.

Basic knowledge of C programming, compilation, compiler optimizations are necessary to be efficient during the debugging of building the V-ECU.

Mastering of Microsoft Visual Code, GCC, CMake can change drastically the performance when optimizing compilation process, execution time or accessing the severity of warnings.

AUTOSAR basic knowledge is an added value as documentation or articles often use specific AUTOSAR vocabulary or concept.

2) *Continuous integration, continuous delivery*

As the process of building a V-ECU is repetitive, developing scripts to automate the tasks is a good investment in term of quality and delay. When the scripts are stable, they can be included in a pipeline of continuous integration. This helps to give autonomy to designer: using an existing and working build pipeline, a rebuild can be triggered by just sending a new code.

But scripts will not replace analysis by engineer when the build or test fail.

3) *Incremental build*

Using a tool chain compatible with incremental build drastically reduces compilation time during the iterative phase of the building of V-ECU when integration issue or compilation issue are resolved one by one. A rebuild of a V-ECU of ~1500 specifications with few new specifications is taking 15 min compared to 1 hour and half.

4) *FMI standard*

Including simulation of CAN (frames, messages, CRC) in the FMU is not available for FMU 2.0. If required, this is done using CANoe to modelize the Frame, CRC, ... in the simulation platform. FMI 3.0 Layered standard for network communication could bring easier integration of V-ECU in simulation platform with several V-ECU communicating through CAN as the network abstraction is included in FMU.

5) *Synergies between SIL and HIL*

Artefacts like plant model, CAN model, automatic tests developed for SIL are used on HIL. To take benefit of the synergies, the development and experimentation toolchain must be the same. One step further is to have a unique team for designing the plant model for SIL and HIL and same validation engineer using SIL then HIL.

6) *Challenges*

It's quite easy to build V-ECU if you already have a stabilized process to create production C code and to define the content of an applicative software.

Deploying V-ECU is seen in a first time as extra activity without added value. After some experimentations, interest increased and new usage are identified and deployed. The challenge is now to give them tools to rebuild a V-ECU in autonomy.

V-ECU offers the possibility to have access to a software several weeks before the real software and this should be exploited by tuning teams to prepare their calibration. As this is a new process, we must convince them of the gains for their activity.

It takes time to convince validation engineers used to work on HIL to migrate a part of their activity on SIL. The tests that can be done on SIL must be identified and arguments given. All tests not linked to hardware or basic software and focused on functionality and no performance tests can be performed on SIL. Using same environment plant model for the tests mentioned above reduces difference of behavior between SIL and HIL. Difference of behavior for the same test executed on SIL and HIL must be analyzed, and root cause understood. If the behavior on HIL can't be reproduced on SIL, the test must be executed only on HIL. A good starting point is to try to reproduce a bug found on HIL on SIL.

FMU V-ECU is seen as a 'black box' for people used to integrate model of software as Matlab Simulink as they don't see the content and can't modify the content. The high number of ports compared to simplified Matlab Simulink model is also an obstacle.

A part of integration activity on applicative software is done when building the V-ECU, but the part related to the hardware or basic software remains for the integrator of the full software.

7) *Next steps*

All next ePowertrain software will be 'AUTOSAR' software. Current V-ECU build toolchain, developed for non AUTOSAR software will be adapted.

The deployment of a CI / CD tool chain is planned to increase quality but also to allow more people to build variant of existing V-ECU.

The different scenario of V-ECU rebuilt must be analyzed to find the best answer to reduce build time or complexity of process with the objective to give more autonomy to the final

users. Three scenarios will be examined: change of code without changing the code interfaces, change of SWC ports and change of ports at the boundary of V-ECU.

Connecting V-ECU and plant model is time consuming and error prone. Port names, plant model architecture must be standardized to allow usage of tool features like 'autoconnect' or development of scripts using tool API.

VII. CONCLUSIONS

The trends we described in the automotive field at the begin of this paper makes the digital twin for embedded software an unavoidable technology to reach the challenges, all the actors of this industry are facing.

In the case of AMPERE ePowertrain described in this paper, they saw and confirmed several gains and advantages.

Deployment of SIL for software development can reduce development effort and improve quality by finding bugs before the delivery of codes to the supplier, preparing HIL test environment. Debugging the software can be done on a PC without specific tool and by stepping through the code within the developers IDE.

Having a model of the future software is also helpful to build system or vehicle simulation platform to test system requirement or debug ECU integration connected through network.

In short term AMPERE think that more and more domains will use that technology. In mid-term they will integrate all their virtual devices into an overall simulation, relying on standards (e.g. FMI/FMU, AUTOSAR, XIL API) and co-simulation capabilities.

The usage of standards is a key point to ease the introduction and usage of SIL and to create digital twins for testing new E/E architectures.

From a tool vendor point view like dSPACE, the virtualization technology is used almost by all the automotive

players. Some of them deployed it to create a whole Virtual E/E architecture, while other are using it for testing ECU application software. Team responsible of the applicative software are working up to level 2 V-ECU, when the team producing the full software are focusing on level 3.

While software virtualization technology is used increasingly in the automotive for industrial project, the other industries are seeing it with interest and are starting to explore its benefits.

REFERENCES

- [1] ASAM Report describing test methods and test environments: [ASAM-Test-Specification-Study-Group-Report-2022_V1-0-0.pdf](#)
- [2] Prostep Whitepaper "Requirements for the Standardization of Virtual Electronic Control Units (V-ECUs)": https://www.ps-ent-2023.de/fileadmin/prod-download/WhitePaper_V-ECU_2020_05_04-EN.pdf
- [3] Almasri, I., Abandah, G., Shhadah, A., Shahrouh, A. (2011, December). Universal ISA simulator with soft processor FPGA implementation. In Applied Electrical Engineering and Computing Technologies (AEECT), 2011 IEEE Jordan Conference on (pp. 1–6: [Universal ISA simulator with soft processor FPGA implementation | IEEE Conference Publication | IEEE Xplore](#)
- [4] Jan Röper, Arpita Bhattacharjee, Franz Kramer, Purushottam Kuntumalla, Andreas Junghanns "Combining Real and Virtual Electronic Control Units in Hardware in the Loop Applications for Passenger Cars" <https://hal.science/hal-03699658/document>
- [5] FMI Layered Standard on GitHub: [GitHub - modelica/fmi-ls-bus](#)
- [6] Katumba, Brian, Knauss, Eric - Agile Development in Automotive Software Development: Challenges and Opportunities - 15th International Conference, PROFES 2014, Helsinki, Finland, December 10-12, 2014. Proceedings, At: Helsinki, Finland
- [7] Association for Standardisation of Automation and Measuring Systems (ASAM)- Standards: [Standards \(asam.net\)](#)
- [8] Mc Kinsey: Smartphones on wheels: New rules for automotive-product development: [New rules for automotive product design and development | McKinsey](#)
- [9] Yohan Jordan, Dirk von Wissel, Adrian Dolha, Jakob Mauss "Full Virtualization of Renault's Engine Management Software and Application to System Development" ERTS 2018
- [10] CDF: <https://www.asam.net/standards/detail/cdf/>

Towards safe obstacle detection for autonomous train operation: Combining track and switch detection neural networks for robust railway ego track detection

Philipp Jass* , Carsten Thomas , Tina Hiebert and Gustav Plettig

*Department of Energy and Information — Computer Engineering
HTW Berlin, University of Applied Sciences
Berlin, Germany*

*philipp.jass@htw-berlin.de

Abstract—Similar to autonomous driving on the road, automated and autonomous train operation also offers many advantages. These include relieving the burden on train drivers, as well as a possible increase in line capacity or the redevelopment of previously unprofitable route sections. One of the most important tasks of an autonomous train control system is to monitor the surroundings and, above all, the route to be traveled. This must be continuously monitored for possible obstacles in the train’s path, just as a human train driver does. In order to perform this task, sensors are required that record data about the train’s surroundings. Such sensors in autonomous systems are usually cameras, radar or lidar sensors. To detect obstacles on the track, the critical zone must first be identified. For trains, this area is called the clearance gauge and describes the space that the train occupies when traveling on a track. In complex scenes with switches, the section of track that the train travels through – the ego track – must be determined depending on the status of the switches. This paper presents an image-based approach for embedded on-board ego track determination, combining track and switch information in order to achieve a more robust ego-track prediction.

Index Terms—On-Board Railway Track Detection, Machine Learning, Computer Vision, Robustness

I. INTRODUCTION

Rail transportation is considered as a safe and energy-efficient mode of transport. But the technology needs to evolve to cope with increasing demands for traffic density and flexibility. Requests to channel more trains in closer sequence on the same network infrastructure and the interweaving of long distance and regional passenger traffic with cargo transportation creates new challenges for control and protection systems, and for train drivers and other involved personnel. On-board systems for driver assistance and systems enabling automated train operation are elements helping to respond to these challenges.

Funded by the German Federal Ministry of Education and Research (BMBF) (Grant Number 01IS22029C), within the scope of Project ‘Certifiable machine learning based controls for safety-critical applications (CertML)’ as part of the program ‘KI4KMU - Research, Development and Use of Artificial Intelligence Methods in SMEs’.

Such systems shall either support the driver in their key operative duties, or even replace the driver and fully assume these tasks. One of the key capabilities required for such systems is the safe and reliable detection of obstacles on the train path. The obstacle detection performance required must be at a level similar or better compared to that of human train drivers. Consequently, the systems must be able to cover a range of several hundred meters distance (direct line of sight), operating conditions that include different light conditions such as shadows, rain and fog, backlight, and dawn. Safety integrity levels (SIL) for the covered functions can be derived through the analysis of use cases and associated hazards.

Railway track topology includes switches that connect individual railway tracks (either merging or splitting), and crossover points where individual tracks cross each other. In general, obstacles may be detected on any of the tracks in front of a vehicle. Taking all these into account would potentially lead to many “false positive” detection events, leading to unnecessary alarm or braking reactions and rendering the system unusable. Instead, the system should be able to understand which path the vehicle will actually take considering switch positions and crossings (the “ego track”), and to take only those obstacles into account that are situated on the ego track.

In principle, ego track detection could be based on a combination of train localization information, track topology data, time table and route planning information. Yet, each of these information elements may be either not accurate enough or not up-to-date. E.g., track-exact localization by GNSS (Global Navigation Satellite System) is extremely difficult to realize and infrastructure-based localization means are not available everywhere. Therefore, ego track detection must be either completely based on or at least substantiated by on-board means.

In order to perform both, ego path detection and obstacle detection, sensors are required that observe the railway infrastructure in front of the train. On-board perception systems usually use combinations of daylight and/or infrared cameras,

and sometimes additional detection means such as LiDAR or radar sensors. Based on the acquired sensor information, the ego path is identified taking into account the sensed track topology (potentially including all visible railway tracks), switches and crossings, switch positions, and railway signals. Such ego track then may be considered as region-of-interest that may be subsequently scanned for obstacles or other anomalies.

In this paper we present an approach for ego track detection based on combining deep neural networks (DNNs) for track detection and switch detection, thereby implementing a two-step approach involving plausibility checks that enhance its robustness considering complex track topologies. In the following sections, we first present related work and explain the need and the challenge of robustly identifying the ego track as a precondition for safe obstacle detection. In the next section, we explain our approach that is based on combining two specialized neural networks to detect tracks and switches, and to combine the detection results to properly identify the ego track. Further we present experimental results for the different approach variants implemented and provide our conclusions.

II. RELATED WORK

Compared to the automotive sector, less research has been carried out in the field of advanced driver support systems and autonomous driving in the railway sector. Yet, during the last years, the subject has started to attract a lot of attention. Railway operators invest into these topics, state funding is being made available, and several large research consortia have been established like "Railenium" in France and "Digitale Schiene" in Germany. Momentum has been established that led to a considerable number of approaches and initial solutions in this area.

Several studies that have examined the importance and necessity of autonomous trains for our society, as well as the related risks and challenges, e.g. Trentesaux et al. [1]. Hyde et al. [2] and others worked out that obstacle detection is necessary for trains with a higher grade of automation (GoA). Since in those systems the safety-critical monitoring function of the driver is replaced by a software control system, those systems need to fulfill safety regulations and requirements. Safety regulations for automated driving of metros do exist (EN 62267), whilst the similar regulative framework for main-line railway – which is a much more diverse and challenging environment compared to metros – is still under development. Initial analyses for safety requirements are already available. One such analysis performed for the German railway authority focusing on ATO at GoA level 3 determined required safety integrity levels of SIL0 to SIL2 depending on the task taken over by the automated system, with SIL3 required in exceptional cases [3].

For railway obstacle detection, both classical computer vision (CV) methods and approaches based on machine learning (ML) have been presented in the literature. Ristić-Durrant et al [4] have carried out a comparative literature analysis on the different approaches. This work and other more recent

work [5] indicate that ML-based methods are better suited for obstacle detection tasks due to their robustness regarding complex scenes and diverse operating conditions. On the other hand, traditional approaches have advantages with respect to the certifiability for safety-critical applications such as automated train operation.

In [6]–[10], neural networks are proposed for track detection as a first task in an obstacle detection pipeline. Semantic segmentation is used to identify rails or complete tracks, by assigning each pixel in the image to a semantic class.

Many published obstacle detection approaches do not explicitly aim to identify the ego track, but rather include all visible tracks in front of the train as region of interest for obstacle recognition [11], or compute all possible paths starting from the track immediately in front of the vehicle (i.e., excluding all those tracks without a direct connection to the current train position) [12]. As explained earlier, this may lead to many "false positive" identifications of obstacles that are truly obstacles, but situated on tracks that will actually not be used by the train.

To identify the ego track in the set of all visible tracks, switches and switch positions have to be identified. Jahan et al. [13], e.g., present an object recognition network that can recognize switches in railroad scenes, including their status.

Identifying the ego track with a single neural network, extracting the characteristic features for rails and tracks as well as the features for switches and switch position information is extremely challenging. Our own experiments in this direction indicated that the approach works for simple scenarios, but has difficulties to properly identify the ego track in more complex rail topologies. Therefore, we propose a different approach in this paper, combining two specialized neural networks for the identification of tracks and the detection of switches and switch positions (see section III-A). However, a paper recently published by Laurent [14] shows very promising results also for a single-network ego track detection.

A major challenge for machine learning approaches in the railroad sector is the availability of annotated training and test data, as underlined by [4] and many other publications in this field. Very few datasets in this field are publicly available. One of the most widely used public datasets is the Railsem19 dataset with 8500 annotated scenes [15]. Other, rather novel ones are RailSet from the "Railenium" context with 6600 images [16], and the Open Sensor Data for Rail dataset, published by the "Digitale Schiene" initiative and containing approx. 1500 multi-sensor frames [17].

III. COMBINED EGO TRACK DETECTION APPROACH

A. Approach

As explained in section II, safe and reliable obstacle detection requires a robust approach for the detection of the ego track. Detecting the ego track with a single network approach is very challenging and potentially will not achieve the required detection performance for complex scenes. Therefore, in this paper we propose to split this task into several subtasks. The idea behind this is that each of the subtasks

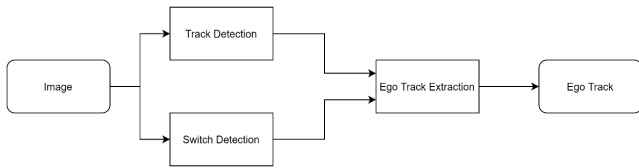


Fig. 1: Combined Architecture for robust on-board ego track detection. Blocks with rounded corners represent input and output data. Rectangular blocks represent architectural elements.

can be optimally performed with a corresponding specialized neural network. For detecting the tracks, the widely used and practice-proven approach of treating rail detection as a semantic segmentation task is retained. This approach has the advantage that the image areas in the image that belong to the track are provided as direct result of such a recognition. These image areas are also the areas that need to be scanned for potential obstacles. In typical images containing several railway tracks, the ego track cannot easily be identified. Only the operational context of the moving train determines which of the contained track segments represents the ego track. Therefore, instead of detecting the ego track, it is convenient to let the segmentation network detect all tracks contained in the image. Thus, rail detection is a sub-task of ego track detection.

In order to be able to decide which of the identified tracks is the ego track and which tracks are neighboring tracks, it is necessary to recognize and understand the switches themselves and their position. By the position of a track we mean the position of the switch blade, which determines the direction in which the train will travel at a switch. The second sub-task of ego track recognition is therefore switch recognition. A switch can be identified as an object in the image by its characteristic structure. In this paper, we propose to use a neural network for object recognition for this sub-task.

Finally, the third sub-task is the extraction of the ego track by combining the previously generated track and switch recognition results. In order to better meet existing safety requirements and objectives, it is advisable to program this task based on rules and not to solve it with a neural network. This allows for better testability and thus verifiability of the system's safety properties. The resulting architecture of the proposed combined system for robust on-board ego track detection is shown in figure 1.

B. Custom Ego Track Detection Dataset

For our work, we used a specifically created dataset that comprises track data, differentiating ego track and other tracks, and switch data including switch position data. The images of the dataset are frames from the "minute by minute" documentary from the Norwegian Broadcast Corporation, a video from the driver's perspective of the railway between Trondheim and Bodo [18]. An example of an annotated image is displayed in figure 2, containing track labels and switch labels.

Track labels are segmentation masks which use different labels

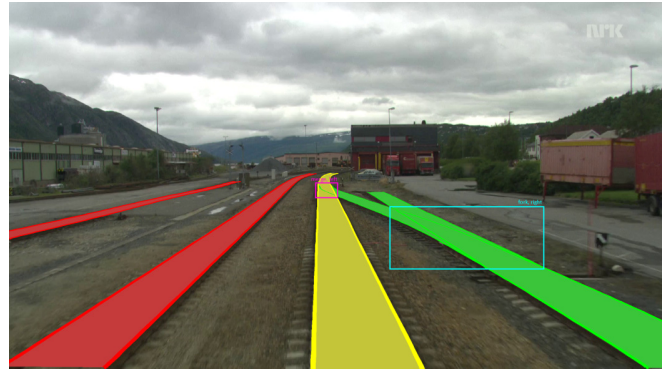


Fig. 2: Labelled image. The yellow track is the ego track, red and green tracks are left and right neighbor tracks. The blue box is a fork label, the pink box is a merge label.

for ego track, left neighbor track(s) and right neighbor track(s). Furthermore, rails and trackbed have different labels. These labels allow for a variety of training scenarios: a neural network can be trained on the ego track only or include neighbor tracks as well while taking into account only the rails or the whole track.

Switch labels are bounding boxes and also provide information on the type of switch, merge or fork, and the relative direction: left, right or unknown. The unknown direction is used for obscured or far off switches, where the direction cannot be determined.

There are 6802 images with labelled tracks and 2334 images with labelled switches. The datasets with labelled tracks and switches have an intersection of 424 images where both, tracks and switches, are labelled.

To increase the number of images for our experiments, we additionally used the RailSem19 dataset [15] by adapting the available labels to our labelling scheme. The resulting combined dataset contains 15302 images.

C. Neural Networks

a) Rail Detection: We used a MobileNet-SegNet architecture – which had shown good performance in our earlier work – as DNN for rail detection. It consists of a backbone network for feature extraction (the encoder) and a decoder network. The encoder part uses a MobileNet architecture for extracting features from the input image. The architecture was described in a paper by Howard et al. [19] and was developed for computationally efficient image processing in mobile applications. A SegNet architecture, as described by Badrinarayanan et al. [20], was used as decoder. Based on the extracted features, this decoder generates a segmentation mask, which marks the detected track area. For the implementation of the network we used the code from Gupta [21].

The described network was trained using stochastic gradient descent with batch size 8 and learning rate 0.01 for 100 epochs. For the training process we used the combined dataset of 15302 images described in section III-B which we split into training (70%), validation (15%), and testing (15%) sets (the RailSem19 images were used only in the training set).

We trained the network twice. The first training was done for direct (single-shot) ego track detection. The network was trained with the images of the training dataset and masks marking only the ego track in the image. The resulting DNN is referred to as MS-ego in the following. For the second training, the same images were used, but with masks that marked all tracks in the image. This trained DNN is referred to as MS-all in the remainder of the paper. In this way, we obtained one network that tries to detect the ego track directly on the image and one that tries to detect all the tracks in the image.

Those trained neural networks predict the rails, i.e. the track area, as a segmentation mask. Such a segmentation mask is a gray-scale image representing the confidence of the DNN for every pixel, that it belongs to the track area, which means black pixels in this image represent background pixels (0% track) and white pixels (pixel value 255) represent confident track pixels (100% track). All pixel values in between represent a respective track confidence. To obtain a binary mask, which is required for the ego track extraction algorithms, the segmentation masks need to be preprocessed. In our approach this included thresholding with threshold 0.5, morphological closing with kernel size (10,10) and filtering out small contours with size less than 200 pixels, as those areas are too small to represent correct track areas.

b) Switch Detection: To detect the switches, we used a DNN based on the YOLO architecture, which is a well known architecture for object detection tasks. We used YOLOv8 [22] for object detection. YOLOv8 gives a choice of several different model sizes (n,s,m,l,x). For our task we used the largest (i.e., x) of the available models to get more accurate results.

For training the YOLOv8 switch detection DNN, we used our previously described switch dataset (see section III-B, split into training (70%), validation (15%) and testing (15%) set. We trained the network to recognize the type of the switch (fork, merge) and also the direction it is set for (left, right, unknown), resulting in a total of 6 different classes.

To improve the training results, we tuned the training parameters using the built-in tuning algorithm from the ultralytics package. This algorithm automatically mutates the parameters and tests them to analyse the fitness of the model. This tuning ran for 300 iterations of 10 epochs each.

We used the *patience* parameter for the final training. This parameter allows to set the number of epochs after which the training will be stopped if there is no significant change in the training process. For training of the YOLO network, we used batch size 20 and trained for 600 epochs.

The trained YOLO switch detection DNN predicts switches in images as a bounding box marking the switch location and a respective class of the switch describing its attributes.

D. Ego Track Extraction

The task of ego track extraction is to extract the ego track, i.e., the track that the train will follow, from the prediction results of the rail and switch detection. For our work, we use the following characteristics of the ego track:



Fig. 3: Example image with track segments (colored lines with marked start and endpoint) and ego track (white overlay).

The ego track is always a single track. Consequently, its mask contains only one contour. For an on-board ego track detection system with forward-looking sensors, we can assume that in case multiple track contours are present in the image, the ego track contour is always the closest one to the image center at the bottom of the image. This contour covers the entire area of the track to be traveled. In our approach, the ego track includes both the rails and the track bed. At switches, the switch position determines the further course of the ego track. If a switch position is unknown or if the switch is set in such a way that the train cannot cross it, the safely detected ego track ends at this switch. Otherwise, the ego track continues beyond the switch for the rail section indicated by the switch position.

The ego track consists of an arbitrary number of track segments, but at least one. In the following, a track segment refers to a continuous track section without branches. Such a segment ends at a switch, i.e. a branching point, and one or two new segments begin depending on the type of switch. Figure 3 illustrates these terms in an example picture.

Two different approaches for ego track detection were implemented during our work. Both algorithms are explained in detail in the following and compared to each other there after.

a) Early Fusion Approach: The early fusion approach aims to extract the ego track directly from the binary segmentation mask generated by preprocessing of the neural network output. This approach is based on convexity defects of the segmentation masks, so only the contours of the segmentation mask are analyzed. The following steps are executed:

Step 1 - Determine Contour Containing the Ego Track: Neighbor tracks without a connection to the ego track may lead to multiple contours in the image. As described above, the contour containing the ego track will be the one closest to center at the bottom of the image. Therefore each contour close to the bottom of the image is analyzed to determine the left and right bottom points. The mean of the left and right bottom points is assumed to be the center bottom point

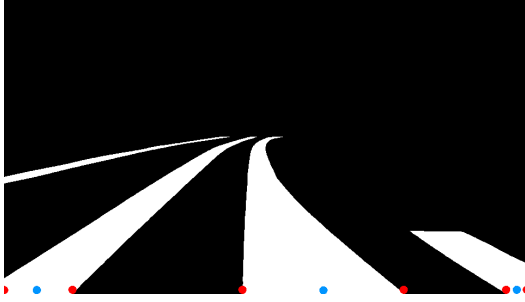


Fig. 4: Early fusion: Multiple track segmentation mask with detected bottom points (center bottom point – blue, left / right bottom points – red).

4. The contour with the center bottom point closest to the image contains the ego track.

Step 2 - Find Switches on Ego Track Contour: To find the switches that are on the ego track contour, the intersection of the ego track contour and the bounding box of each switch is calculated. Only if the area of the intersection exceeds a certain value, the switch is assumed to be on the ego track contour. If there are no switches on the ego track contour, the algorithm stops as area of the ego track contour contains only the ego track. Otherwise the ego track contour is processed further to separate forking and merging neighbor tracks.

Step 3 - Find Frog Point of the Switch: Usually the contours of tracks with switches have a significantly deep convexity defect with the frog point at the farthest point. Curved tracks also show a convexity defect on the inner side of the curve, but such defect is relatively shallow compared to its extent. Therefore only convexity defects with a large depth and a small extent are considered as frog points. In an ideal case, were the tracks fork uniformly to the left and right, the frog point would be at the farthest point of the defect. As this is not always the case, points with a high curvature within the convexity defect are added to the possible candidates for the frog point. From these candidates, the point is selected as the frog point of the switch, which is closest to

- the top line of the bounding box of the switch, if the switch is a fork.
- the bottom line of the bounding box of the switch, if the switch is a merge.

Step 4 - Split Contour at the Switch: To split the contour at the switch, the other end of the switch – more precisely a point opposite to the frog point – must be determined. The other end of a switch is assumed to be at the bottom line of the fork bounding box or the top line of the merge bounding box, respectively. Along this line the opposite point is on the right or left side of the track, depending on the setting of the switch. The opposite point of the switch is determined by following the contour until the level of the bottom line of the fork bounding box or top line of the merge bounding box is reached while ensuring that the line between the frog point and the opposite point is completely within the track contour.

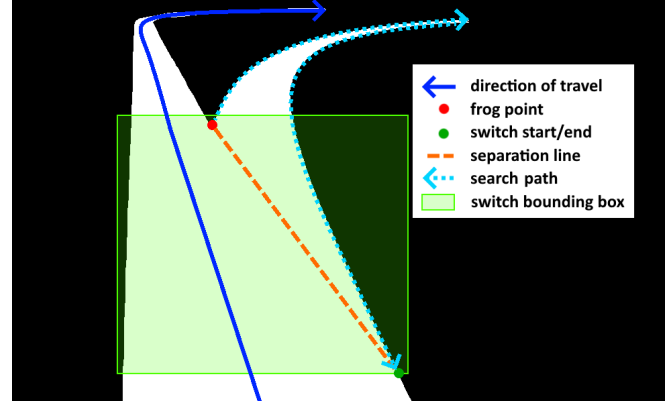


Fig. 5: Early fusion: Contour splitting at a left directed fork.

The frog point and opposite point are marked with red and green dots respectively in Figure 5. The contour is split along a separation line between the frog point and the opposite point.

These steps are repeated until all neighbor tracks are separated from the ego track and the algorithm terminates in step 2 as no more switches can be found on the ego track contour.

b) Late Fusion Approach: In contrast to the early fusion approach, the ego track is not determined directly from the segmentation mask output by the neural network in late fusion ego track extraction. Instead, the results of the rail detection are first preprocessed to describe the rail areas as compactly as possible.

A track is a very simple structure. Its course is determined exclusively by the two rails. These always have a constant distance in the real world. This makes it possible to describe the track along a single centerline. This centerline runs parallel to the two rails in real world coordinates and has the same distance to both. In an image, the centerline is not parallel to the rails anymore due to the perspective of the camera, but still always is in the middle between both rails. The idea of the late fusion approach is therefore to determine the ego track based on the centerlines of the detected track areas. The procedure for this approach is described below.

Step 1 - Find Centerlines: The first step in this process is to calculate the centerlines of the segmentation mask. For each row of the binary images, resulting from the segmentation mask preprocessing, contiguous sections of white pixels are determined, and the mean value between the first and last pixel coordinate of the area is calculated for each of the sections.

This calculation is performed separately for each contour in the binary image in order to obtain the centerline for each track segment individually. If the contour contains a switch, a Y-shape can be recognized. In this case, the individual track segments can be separated from each other by splitting the contour at the point where the number of center points

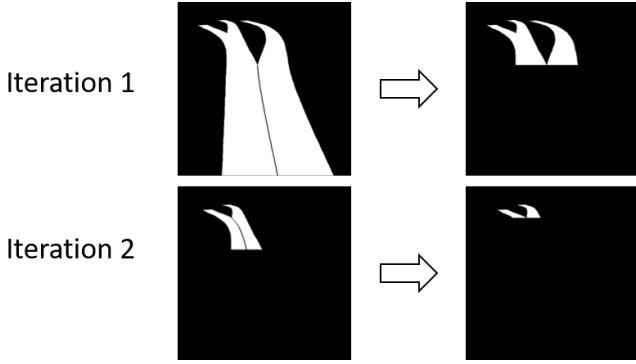


Fig. 6: Late fusion: Splitting of segmentation mask into individual track segments.

changes in between adjacent image rows, i.e. the switch frog (see Figure 6). If a contour contains multiple switches, this procedure is applied recursively until all track segments are separated from each other.

Step 2 - Filter Centerlines: Once the centerlines have been calculated, they need to be filtered. Due to fuzzy edges at to top and the bottom in the segmentation masks, very short centerlines can appear at the beginning and ending of tracks, as well as outside of the track area. Those erroneous centerline fragments are filtered out. Furthermore, centerlines are merged if the upper end of a segment is located very close to the lower end of another segment. This is done in order to describe each track segment with only one centerline, if it was split into multiple individual centerlines due to irregularities in the contour edges.

Step 3 - Identify Switches in the Rail Detection Results: Using the centerlines found in this way, it is possible to identify switches by its characteristic centerline structure independently of the DNN switch detection results. A switch is a point, where three centerlines are starting or ending close to each other. Two centerlines always end at exactly the same y-level, and a third begins in the adjacent pixel row (see Figure 7).

The exact arrangement of the centerline ends even allows a distinction between merging and forking switches. However, the exact setting of the switch cannot be determined using the centerlines. This is one of the reasons why the object detection network is used for switch detection.

Step 4 - Check Plausibility of switches: Since there are now two switch detection results after step 3, it is obvious to check their plausibility against each other. Both the switch detection network and the rail detection network results provide the position of the switch in the image as well as the switch type. Therefore, both can be compared with each other. For this plausibility check, a certain tolerance zone has to be defined for the switch position based on the detected bounding box from the object detection, because depending on the position of the switch in the image, the centerline ends are not always

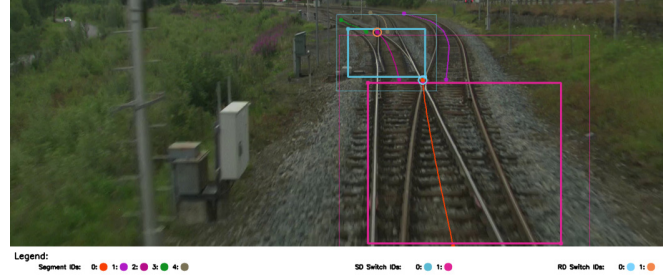


Fig. 7: Late fusion: Identification of switches in rail detection results. Colored lines mark detected centerlines. Solid circles mark segments start and end points. Circles around start/end points mark a switch detected from rail detection result. Bold rectangles mark switch detection bounding box results. Narrow rectangles mark tolerance area around switch bounding box. The legend below the image shows Element IDs for the respective colors.

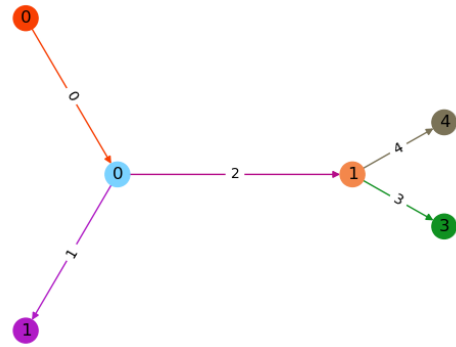


Fig. 8: Late fusion: Track network graph generated from rail detection and switch detection results for the example image in fig. 7 .

within the bounding box, but in the immediate vicinity. If both the position and the type of the switch match, it can be confidently assumed that the switch is correct. The setting of the switch can then be taken from the switch detection result.

Step 5 - Create Track Network Graph: Once the rail segments and switches have been located in the image, the tracks contained in the image can now be represented in a network. This is done using a directed graph. The rail segments are represented as edges and the switches as nodes of the graph. In addition, start and end nodes are created for segments that do not start or end at a switch. This graph (see example in Figure 8) represents the information about the connection of the track segments.

Step 6 - Find Ego Track Segments: Once a graph of the track network has been created from the detection results, it can be used to determine the segments belonging to the ego track. Using the assumptions on the ego track position introduced initially in section III-D, the start node that matches these assumptions best in the graph can be

determined. Starting from this node, the next node of the directed graph is searched recursively until an end of the track is reached. Such an end can either be an end node or a switch where the track ends. With this procedure, a list of segments and nodes belonging to the ego track can be created

Step 7 - Determine Spline Points: As can be seen in the figures for calculating the centerlines (see figures 6 and 7), the centerline found in the switch area does not describe the correct course of the two track segments entering or leaving the switch. For this reason, the centerlines of the corresponding segments cannot simply be output to create the ego track centerline. The incorrect centerline is always located in the section of the switch area that has only one centerline. This area extends from the switch blade to the switch frog. Interpolation is required to reconstruct the correct track centerline there. The entire switch area is defined by the bounding box that is provided by the switch detection results. However, in order to keep the interpolated area as small as possible and thus the interpolation as precise as possible, not the entire area in the bounding box is interpolated, but only the area of the single segment from the edge of the bounding box to the center of the switch. For all other areas, the centerlines of the segments are assumed to be correct. For the interpolation, interpolation points are selected at regular intervals on the y-level from these correct track segments. For the example image, these interpolation points are shown in figure 9. This figure also shows the gap between the interpolation points in the area of the switch.

Step 8 - Interpolate Ego Track Centerline: Finally, spline interpolation is used to generate the ego track centerline. With this method, a quadratic function is adapted to the interpolation points and can then also be evaluated for the switch areas in order to calculate the correct centerline of a track. Figure 9 shows the calculated centerline of the ego track for the example image.

Step 9 - Generate track mask: Based on the centerline, the mask describing the ego track can be reconstructed using the track width. In the real world, this track width is constant. However, since the camera has a perspective distortion, the track width decreases with increasing distance from the camera to the back of the image. However, an examination of the track width for different images has shown that it decreases linearly over the course of the image. The corresponding linear function is used to calculate a corresponding track width for each row in the image. Figure 10 show the reconstructed track area for the example image.

IV. RESULTS

This section presents the experimental results of our investigation. We compared a direct (single-shot) ego track detection approach with our proposed combined approach consisting of rail and switch detection. For both approaches,



Fig. 9: Late fusion: Interpolated centerline using spline interpolation shown as red line. For reference spline points used for the interpolation are shown as well as white dots.

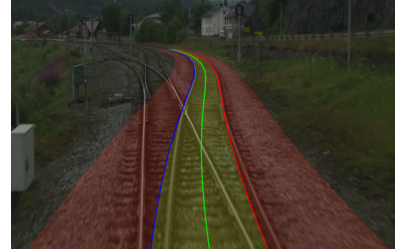


Fig. 10: Late fusion: Generated ego track area. Yellow area marks track area, red area marks safety area around track (according to [23] annex 1 to §9). Blue line marks left ego track edge, red line marks right ego track edge and green line marks ego track centerline.

a neural network was trained with the presented MobileNet-SegNet architecture (see section III-C) for rail detection. For direct ego track detection, this network was trained with masks containing only the ego track for rail detection. This approach is referred to as *singleshot* in the following, utilizing the MS-ego DNN from section III-C. For the combined approach, the same network architecture was used, but trained on masks containing all tracks of the scene (MS-all from section III-C). For the combined approach, both proposed ego track extraction algorithms – early and late fusion described in section III-D – are tested separately. Table I provides an overview of the individual components of the approaches described, as well as the test labels which we use in the following.

For the evaluation of the these approaches, the same test set of images is used for all variants. This test set consists of 2296 images of the dataset described in section III-B. Since

TABLE I: Approaches to be compared and their characteristics and labels

Approach	Rail Detection DNN	Switch Detection DNN	Ego Track Extraction
singleshot	MS-ego	-	-
combined-EF	MS-all	YOLOv8	Early Fusion (EF)
combined-LF	MS-all	YOLOv8	Late Fusion (LF)

TABLE II: Fusion algorithm IoU results on ground truth data.

Image set	Early Fusion	Late Fusion
all scenes	0.972	0.943
switch scenes	0.896	0.921

the proposed combined approach for ego track detection aims to provide an improvement over the state of the art especially for complex scenes including switches, such scenes are of particular interest for the evaluation. In the used dataset, there are significantly more images without switches than with switches. As a consequence, also the test set contains rather few images with switches (only 143). This fact later was identified as problematic (see the result discussion below). In order to evaluate and compare the different approaches investigated in this paper, we performed primarily a quantitative analysis using the Intersection over Union (IoU) metric, complemented by a qualitative analysis for selected images. The IoU metric calculates the ratio of correctly detected areas of a segmentation mask to incorrectly detected areas. The prediction is always compared to the ground truth mask. With GT as the segmentation ground truth mask of the track area and $Pred$ as the predicted mask of the track area, the IoU is calculated using the following equation:

$$IoU = \frac{Area\ of\ Intersection}{Area\ of\ Union} = \frac{|GT \cap Pred|}{|GT \cup Pred|} \quad (1)$$

First, the two fusion algorithms were tested on the ground truth data. This data can be considered as ideal recognition results and should therefore serve to demonstrate how the algorithms work in the ideal case. Both algorithms perform well on the ground truth data. The results can be seen in table II.

Looking at all images of the test dataset, i.e. the first row of the table II, we can see that the early fusion approach has a significantly higher average IoU, indicating a better ego track detection performance. However, the test dataset contains considerably more images without switches than with switches. On such scenes, the basic functional principle of this approach performs better than the late fusion approach. In scenes without switches, the early fusion approach only needs to select the correct contour belonging to the ego track. This usually works very reliably, as the test results show. However, the late fusion approach also shows good overall performance in ego-track detection across all test data. Reconstruction of the ego track area from the centerline seems to be generally feasible. The lower IoU value of the late fusion approach is mainly due to the use of an approximated track width to generate the mask from the centerline. As a result, the predictions are not as accurate as those of the early fusion approach.

However, the second row of the table shows that the late fusion approach performs better when only scenes with switches are considered. Here, the IoU value decreases slightly, but not as much as with the early fusion approach. The detection of the convexity defect for the mask splitting seems to be difficult in some scenes, but the algorithm generally allows a correct

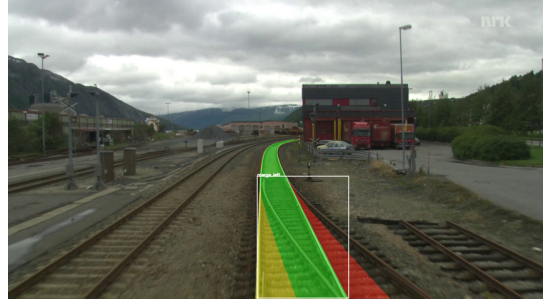


Fig. 11: Late fusion: Incorrect ego track calculation due to (a) missing spline points (b) partially visible merging switch. Green areas=True Positive, Yellow areas=False Negative, Red areas=False Positive. White rectangles represent results of the switch detection network.



Fig. 12: Positive Examples of ego track detection for (a) Early Fusion algorithm (b) Late Fusion algorithm. color meanings as in Figure 11.

detection of the ego track, as indicated by an IoU of almost 0.9. Also for the late fusion approach, the switch scene IoU is lower than that for all scenes. This can be explained by scenes such as the one shown in figure 11. If a merging switch is only partially shown at the bottom of the image because the train has already entered the switch area, the rail detection results will produce a contour that is wider than a single track. In the late fusion approach, the centerline is found and the ego-track mask is generated based on the approximated track width. In this case, however, the centerline no longer runs along the correct course of one of the two track segments leading into the switch, but between them. Since these problem areas are always located at the lower edge of the image, the resulting incorrect areas are quite large and therefore have a major impact on the IoU. As indicated in section V, we intend to improve the late fusion algorithm for handling these specific scenes.

Still, the average IoUs of both algorithms are quite high for the ground truth data, i.e., assuming optimal input from the network predictions. With these positive initial validation results, we consider both algorithms as being able to detect the ego track from a fusion of the detection results of the rail and switch detection and to generate a corresponding mask. Figure 12 shows positive examples for both algorithms for reference.

In the following, we tested the algorithms using the actual recognition results of the two neural networks. Unexpectedly, the results of the generated ego track



Fig. 13: Negative Examples of MobileNet-SegNet all-tracks detection results. color meanings as in Figure 11.

predictions indicate that the singleshot approach, which employs a single network to detect the ego track, yields the most favorable outcomes (see table III). The singleshot approach exhibits a remarkably high IoU of approximately 0.94 across the entire test set and across all subsets examined. Additionally, the network’s predictions are remarkably consistent, as a comparable average IoU value is achieved for scenes with and without switches. Investigating the reasons for this unexpected result, we found three problematic areas regarding our experiments.

Firstly, the MS-all network results shown in table III demonstrate that – in comparison to direct ego track detection – the task of detecting all tracks in the image presents a considerable challenge for the MobileNet-SegNet DNN. The image areas to be detected are larger in this case, and as the neighboring tracks are more likely to be at the edges of the image, there is also a greater distortion caused by the camera. Especially high-complexity scenes are challenging for the MS-all network. For these scenes, it occasionally produces incomplete or inaccurate predictions, including holes in the predicted masks and incorrectly recognized contours outside the actual track area (see figure 13). As both ego track extraction algorithms are predominantly geometry-based, their IoU value is considerably impacted by a low IoU of the MS-all network due to those irregularities.

Secondly, the switch detection DNN performs less well than expected. In the entire test set, there are 143 images with switches that were labeled in the ground truth data. Of these, only 126 images (i.e., 88%) were properly identified by the switch detection network. The performance gap may be attributed to the rather small size of the training data set. As the combined approaches strongly depend on the switch detection correctly identifying switches and their direction, the performance gap likely has a strong impact on the results.

Thirdly, it is important to note that the excellent recognition results achieved by the singleshot approach may be to some extent attributable to the fact that the test set predominantly comprises relatively simple scenes. The number of switches in the scenes is small, with the majority of the ego track switches leading to a straight continuation of the ego track. It is therefore possible that the MS-ego network may have learned this fact as a result of overfitting. This is exemplified by a qualitative analysis of switch scenes in which the ego

TABLE III: Mean IoU results of the fusion algorithms compared to the ego track detection network. GT switch scenes are all scenes with labelled switches. Pred. switch scenes are all scenes with predicted switches from switch detection network. The results of the MS-all DNN are given for reference in the last column, since the combined approaches are based on these results.

Image set	singleshot	combined-EF	combined-LF	MS-all
all scenes	0.945	0.939	0.916	0.937
GT switch scenes	0.941	0.859	0.833	0.907
Pred. switch scenes	0.943	0.867	0.846	0.912

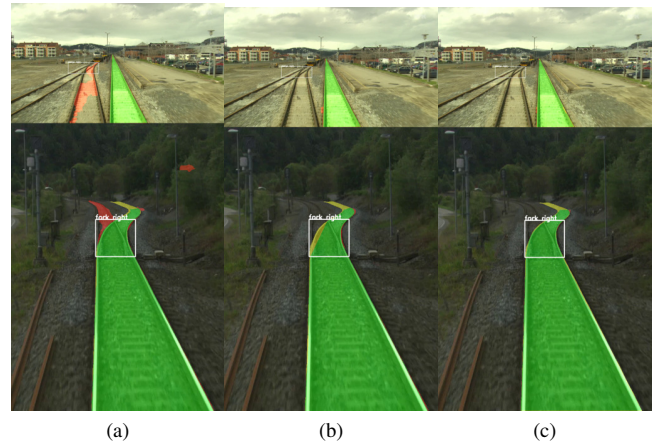


Fig. 14: Examples of ego track detection errors of the singleshot approach. a) shows singleshot output, b) shows combined-EF output and c) shows combined-LF output respectively for the same input images. color meanings as in Figure 11.

track takes a turn at a switch (see figure 14). In the second row of this figure, it can be observed that the singleshot approach utilizing the MS-ego network selects the incorrect track segment that emerges from the switch. In the first row, the singleshot approach fails to detect the ego track shortly after a merging switch. In contrast, the combined approaches manage to detect the ego track correctly in both scenes. It can therefore be assumed that the detection performance of the singleshot approach would decrease on more complex scenes than those primarily included in the test data set.

Despite these identified problem areas, both combined approaches demonstrate a performance on the actual DNN recognition results that is not significantly below that of the segmentation ground truth data used for initial algorithm validation. We found the combined ego track extraction algorithms being capable of detecting the ego track also on these non-optimal segmentation masks. On the actual DNN recognition results, the combined-EF approach is slightly superior to the combined-LF approach as indicated by the

slightly higher IoU. From table III it is also evident, that the detection results of the switch detection play a significant role in both approaches.

V. CONCLUSION

In this paper, we have proposed two algorithms that are able to identify the ego track by combining the neural network detection results for track detection and switch detection.

Our current evaluation of detection performance of these two approaches in comparison to a direct approach utilizing a single neural network indicates that the singleshots approach achieves good results in the analysis of low-complexity scenes. We assume that for higher complexity scenes, the combined approaches presented in this paper are more robust. Whilst qualitative analysis of typical complex scenes supports this assumption, we were not yet able to quantitatively substantiate this assumption due to limitations of the dataset (being still too small and containing primarily low-complexity scenes).

As our analysis has shown, the performance of the two fusion algorithms depends very much on the quality of the recognition results of both DNN, for track segmentation and for switch detection. Therefore, the training of the neural networks must be improved in the future. Our main task in this area will be to develop a larger image database for both training and evaluation. In particular, more images with labeled tracks (ego track and all tracks) and more complex scenes need to be obtained. Also the fusion algorithms must be further amended to counteract their susceptibility to error. With these future improvements, we expect the combined algorithms to perform at a similar or better level than the singleshots approach.

Since – in light of the safety requirements for autonomous train operation – our ultimate aim is a provably safe approach towards ego track detection, we consider the use of an explicitly defined algorithm for detection result combination as an advantage, as such explicit algorithm can be developed according to traditional software safety regulations such as EN 50657. In addition, the combined approaches – specifically the combined-LF approach – offer more possibilities to check the plausibility of the predictions during operation. Some of these possibilities have been presented in this paper, others we will investigate in more detail in the future. Thus, in our view, the presented combined approaches to ego track detection are more suitable for use in safety-critical applications supporting autonomous train operation than singleshots approaches, provided that comparable ego track detection performance can be achieved.

REFERENCES

- [1] D. Trentesaux, R. Dahyot, A. Ouedraogo, D. Arenas, S. Lefebvre, W. Schon, B. Lussier, and H. Cheritel, “The autonomous train,” in *2018 13th Annual Conference on System of Systems Engineering (SoSE)*. IEEE, 62018, pp. 514–520.
- [2] P. Hyde, C. Ulianov, J. Liu, M. Banic, M. Simonovic, and D. Ristic-Durrant, “Use cases for obstacle detection and track intrusion detection systems in the context of new generation of railway traffic management systems,” *Proceedings of the Institution of Mechanical Engineers, Part F: Journal of Rail and Rapid Transit*, vol. 236, no. 2, pp. 149–158, 2022.
- [3] J. Braband, B. Evers, M. Kinas, L. Lindner, D. Mihailescu-Stoica, F. Rexin, F. Adebahr, B. Milius, and H. Schäbe, “Risikoakzeptanzkriterien für das automatisierte fahren auf der schiene.”
- [4] D. Ristić-Durrant, M. Franke, and K. Michels, “A review of vision-based on-board obstacle detection and distance estimation in railways,” *Sensors (Basel, Switzerland)*, vol. 21, no. 10, 2021.
- [5] M. Ziegler, V. Mhasawade, M. Köppel, P. Neumaier, and V. Eiselein, “A comprehensive framework for evaluating vision-based on-board rail track detection,” in *2023 IEEE Intelligent Vehicles Symposium (IV)*, 2023, pp. 1–8.
- [6] H. Li, Q. Zhang, D. Zhao, and Y. Chen, “Railnet: An information aggregation network for rail track segmentation,” in *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 72020, pp. 1–7.
- [7] Z. Tao, S. Ren, Y. Shi, X. Wang, and W. Wang, “Accurate and lightweight railnet for real-time rail line detection,” *Electronics*, vol. 10, no. 16, p. 2038, 2021.
- [8] Y. Wang, L. Wang, Y. H. Hu, and J. Qiu, “Railnet: A segmentation network for railroad detection,” *IEEE Access*, vol. 7, pp. 143 772–143 779, 2019.
- [9] S. Belyaev, I. Popov, V. Shubnikov, P. Popov, E. Boltenkova, and D. Savchuk, “Railroad semantic segmentation on high-resolution images,” in *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 92020, pp. 1–6.
- [10] M. A. Hadded, A. Mahtani, S. Ambellouis, J. Boonaert, and H. Wannous, “Application of rail segmentation in the monitoring of autonomous train’s frontal environment,” in *International Conference on Pattern Recognition and Artificial Intelligence*. Springer, 2022, pp. 185–197.
- [11] S. Khruakhray and J. Srinonchat, “Railway track detection based on segnet deep learning,” in *TENCON 2023 - 2023 IEEE Region 10 Conference (TENCON)*, 2023, pp. 409–413.
- [12] M. Ghorbanalivakili, J. Kang, G. Sohn, D. Beach, and V. Marin, “Tpe-net: Track point extraction and association network for rail path proposal generation,” in *2023 IEEE 19th International Conference on Automation Science and Engineering (CASE)*, 2023, pp. 1–7.
- [13] K. Jahan, J. Niemeijer, N. Kornfeld, and M. Roth, “Deep neural networks for railway switch detection and classification using onboard camera images,” in *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 1252021, pp. 01–07.
- [14] T. Laurent, “Train ego-path detection on railway tracks using end-to-end deep learning,” 2024.
- [15] O. Zendel, M. Murschitz, M. Zeilinger, D. Steininger, S. Abbasi, and C. Beleznai, “Railsem19: A dataset for semantic rail scene understanding,” in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE, 62019, pp. 1221–1229.
- [16] A. Zouaoui, A. Mahtani, M. A. Hadded, S. Ambellouis, J. Boonaert, and H. Wannous, “Railset: A unique dataset for railway anomaly detection,” in *2022 IEEE 5th International Conference on Image Processing Applications and Systems (IPAS)*, vol. Five, 2022, pp. 1–6.
- [17] R. Tilly, P. Neumaier, K. Schwalbe, P. Klasek, R. Tagiew, P. Denzler, T. Klockau, M. Boekhoff, and M. Köppel, “Open sensor data for rail 2023,” 2023. [Online]. Available: <https://data.fid-move.de/dataset/3d7e7406-639f-49f6-bbca-caac511b4032>
- [18] NRK. Nordlandsbanen: minute by minute, season by season. [Online]. Available: <https://nrkbeta.no/2013/01/15/nordlandsbanen-minute-by-minute-season-by-season/>
- [19] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, “Mobilenets: Efficient convolutional neural networks for mobile vision applications.”
- [20] V. Badrinarayanan, A. Kendall, and R. Cipolla, “Segnet: A deep convolutional encoder-decoder architecture for image segmentation.”
- [21] D. Gupta, “Image segmentation keras: Implementation of segnet, fcn, unet, pspnet and other models in keras,” *arXiv preprint arXiv:2307.13215*, 2023.
- [22] ultralytics, “Yolo v8.” [Online]. Available: <https://github.com/ultralytics/ultralytics>
- [23] Bundesamt für Justiz, “Eisenbahn-bau- und betriebsordnung: Ebo,” 1967.

Partially trustworthy action planning thanks to an easily certified plan validator

Jean-Louis Dufour
Safran Electronics & Defense
Massy, France
Email: jean-louis.dufour@safrangroup.com

Abstract—Action planning is the second obstacle (after environment perception) on the path to trustworthy autonomous systems. An action planner is so complex that certifying it would be astronomically expensive. So it will be necessary to associate it with a plan validator responsible for checking plan correctness, to whom the full weight of certification will be transferred. The contribution of this paper is the simple observation of the unexpected proximity between the PDDL planning language and the SCADE synchronous language. From the technical point of view, this proximity allows a simple translation from PDDL to a SCADE model of this plan validator. From the process point of view, if PDDL is accepted as a software specification language, it greatly facilitates validator certification. The two models accept the same plans when all the variables have finite domains, but this is no longer true with an integer-valued variable, and we will sketch a way to deal with this problem.

Index Terms—action planning, certification, PDDL, SCADE, trustworthy AI.

I. INTRODUCTION

A. Context

One day soon, action planning (also called “goal-oriented” and “task” planning) will be embedded in a commercial product, and in our case it may be a safety-critical system requiring certification. Making plans is considered a sign of intelligence, and for this reason action planning is part of what is called Artificial Intelligence (and is therefore also called “AI” planning). Today, when we talk about embedded AI, we think more about perception functions, like object detection (vehicles, people, ...) and segmentation (roads, sidewalks, ...). Intelligence is a nightmare for certification, but the challenges are highly function-dependent. Perception functions have firstly a problem of specification (what is a vehicle? a road?) and secondly, at least in the case of machine learning, a problem of refinement (usually expressed in terms of “functional trust”: does this vehicle presence/absence report actually match a true vehicle presence/absence?). Action planning has no specification problem (modeling reality as a state machine can be subtle, but has never been an issue), has no “functional trust” problem (does this plan really reach the goal? just simulate it¹), but it has another equally big refinement problem: algorithmic complexity.

This complexity has a technical name: “PSPACE-completeness”. *PSPACE* means that the memory required to

handle a given problem has a size bounded by a polynomial of the problem’s size: this is true of all embedded algorithms. The concern is with *complete*, which means that among all the PSPACE problems, it is one of the hardest. This unfortunate property holds true even if we confine ourselves to the simplest planning problems: those who have a finite state space² (no numeric values, no time). The first symptom of this disease is the length of a plan as a function of the size of the problem it addresses: exponential in the worst case! It means that the search space (the tree of action sequences) is very deep (and wide): a planner, to be efficient, has to be very clever to sufficiently prune this search tree. This cleverness has two consequences: on the one hand software complexity, on the other hand execution time and space variability. This second aspect is not the subject of this paper, but we will comment on it briefly, as it has a significant impact on the architecture of the autonomy functions. This time and space variability can reach several orders of magnitude between two problems of the same size, so there is no hope of getting a reasonable estimate of the WCET (Worst Case Execution Time) or the WCMU (MU for Memory Usage: this acronym is still little used, but regarding critical software the underlying concept is well known and treated in a radical way: allocated memory is constant!). So no matter how much time and space the planner is given, there will inevitably be cases where he’ll be interrupted without being able to come up with a plan. A phenomenon of this magnitude does not exist with perception functions (which can be very consuming, but with a limited variability). A first in the world of embedded software: a critical function will sometimes say “I don’t know”. Of course, we would prefer to have a variant of planning that is both polynomial and expressive, but it would be a major breakthrough.

Let’s come back to the first consequence of PSPACE-completeness: software complexity. A planner is in fact a special kind of constraint solver, such as the Z3 theorem prover (Microsoft) or the CPLEX optimizer (IBM). Developing Z3 according to DO-178C recommendations at level A would be astronomically expensive, and quite risky because software involving symbolic computations has never been developed³ in

²Polynomial and NP-complete variants exist [2], [3], but are not expressive enough for embedded systems.

³The nearest example is the CompCert compiler developed at Inria, and clearly it cannot be done by an industrial R&D team, because of the higher-order logic framework needed.

¹But of course if it is not ‘trust’ that is needed but ‘explanation’, a simulation is not an explanation [1].

accordance with these guidelines. Using two different planners is not an option, because they will give different (but equally valid) answers to the same question. But we are in a favorable situation, where the usual workaround is asymmetric redundancy: the planner will be associated with a plan validator, solely responsible for checking plan correctness, to whom the full weight of certification will be transferred. And it turns out that the basic variants of PDDL [4] (Planning Domain Definition Language, the de-facto standard language for writing action planning models) have actually a validator named VAL⁴[5] and its certification would be much simpler than that of a planner, since it is just a plan simulator. The reader who has never heard of PDDL will find a crash course in the appendix.

Before arriving at our proposal, a remark is in order: this asymmetrical redundancy strategy does not provide complete coverage of the potential failures. First of all, when the planner (mistakenly) says there's no plan, it's unverifiable (more precisely, stated in terms of simulation, a validator would have to check an exponential number of candidate plans). Secondly, when the planner (mistakenly) gives a non-optimal plan, this is also unverifiable (for the same reason: an exponential number of plans to check). This explains the first part of the title "Partially trustworthy action planning ...". But as a reviewer pointed out, these coverage holes aren't necessarily that serious, since plan non-optimality and non-existence are *usually* more performance and availability issues than hazardous events.

B. Contribution

In fact, we don't plan to embed (let alone certify) VAL, because it's still a bit too complex⁵ (even if it's much less complex than a planner). This complexity is due to its ability to validate plans for any PDDL model. To tackle this complexity, we will embed a much less generic validator, which is able to validate plans only for a given PDDL domain.

A bit of PDDL terminology is in order: a PDDL model contains two separate parts, the "Domain" and the "Problem". The *domain* is the structural or generic part of the model. For example, in the toy example that will be used in the next sections, the domain says only that a jug can be emptied or filled or poured into another one, but it doesn't give the capacities of the jugs or their initial state (before the start of the plan) or their final state (after the end of the plan). These three informations are the *problem*, which by the way is an unfortunate name to speak of a specific configuration: it would have been better to call him *instance*. All this to say that in this paper, we will not make the difference between *model* and *domain*: when you read *domain* you can understand *model*.

A PDDL planner is called a "domain-independent", because the domain is an input: it is not hard-wired in the planner (this kind of planner exists also for obvious efficiency reasons and is

called "domain-specific"). In the same way, VAL is "domain-independent", which means that in order to compute its OK/KO status it takes two arguments: the plan and the PDDL model. What we propose is a "domain-specific" validator, which takes only one argument: the plan.

The PDDL model describes a state machine, an initial state and a set of acceptable goal states. VAL creates the state machine, initializes it, executes sequentially each action of the plan ('executes' means *dynamically interprets* the PDDL syntax of the action, and the result is a modification of the current state), and checks that the final state is an acceptable goal.

Instead, we will use a *domain-specific* validator, whose SCADE⁶ model will be automatically generated offline. The actions are no more executed by an *interpretation* of their PDDL syntax, but directly by the SCADE code resulting from the translation: the state-machine is hard-coded into the SCADE model, which has now a single input: the plan. In DO-178 jargon,

- the High-Level Requirements (HLR) are the PDDL model (which has to be accepted as a *specification model*, with the DO-331⁷ meaning),
- the Low-Level Requirements (LLR) are the SCADE model (as usual),
- and the validation of the LLRs against the HLRs is the correctness of the translation.

Moreover, given the simplicity of the translator, it is quite possible to *qualify* it (with the DO-330⁸ meaning) in order to further reduce certification activities.

This explains the slightly provocative second part of the title "... thanks to an easily certified plan validator".

So, to sum up, we propose to split the checking function performed by VAL into an offline-and-qualified part (the PDDL to SCADE translator) and a minimalist/specialized online-and-certified part (the SCADE model given by the former translator). Our claim is that *this split will bring a gain in development costs of at least an order of magnitude compared to a naive redevelopment of VAL compliant with avionics regulations*.

C. Related works

Our proposal of a certified plan validator belongs to a (mostly academic) domain called "Runtime Verification" (RV) or "Runtime monitoring". It also belongs to a (mostly engineering) domain called "safety architectures", which is older, so we will begin with it. Lastly, the origin of this work, and specifically of the last section on integer bounding, is the "planning as model-checking" paradigm, but due to its different nature, the associated related works are relegated to this last section.

⁶<https://www.ansys.com/products/embedded-software/ansys-scade-suite>

⁷"Model-Based Development and Verification Supplement to DO-178C and DO-278A".

⁸"Software Tool Qualification Considerations" relates the development effort of a tool to the trust that can be placed in it. Basically, it is "DO-178 for tools".

⁴<https://github.com/KCL-Planning/VAL>

⁵30Kloc of C++ with complex data structures; maybe in the future, a Rust implementation could be certified.

In the railway sector, the two major safety-critical functions, Interlocking and Train Control, were provided by software from the beginning of the 80s. The very first systems based software correctness on N-version programming [6], [7], but it was soon realized that the proof of absence of common mode would not be easy. Hopefully, compared to avionics, railway systems have two main differences: first emergency stop or red light lead to a safe state, which means that it is enough to *detect* bugs (and stop; if availability is not an issue), and second, the safety properties are simpler (no overspeed and sufficient interval). In just a few years, the safety concept evolved towards asymmetrical redundancy, described in [8] under the name of *Safety bag*. The asymmetry comes from the limitation of diverse programming to the safety-critical functions, and it also leads to a reduction of the needed hardware resources: the monitor is overall more frugal than the control function. That's the reason why, today, it is the most widely used safety concept not only in railway, but also in automotive [9]. On the academic side, we will notice two references. First, [10] covers a wide scope, composed of detection and recovery; we only deal with detection. And inside detection, they cover the creation of the planning problem, the planning itself and the execution of the plan; we only deal with planning. In order to detect also problem generation bugs, their *“plan analyzer ... verifies that the produced plan satisfies a number of constraints and properties ... [that] can be obtained from the system specification and from domain expertise but it must be diverse from the planner model”*. The scope is wide, but the coverage of the planner bugs is not guaranteed. Second, [11] automatically synthesizes the Safety bag from the hazard analysis. The analogy with the present proposal is that the plan validator is a safety bag, and we automatically synthesize this Safety bag from a PDDL specification. The difference is that they ensure safety during the execution of the plan, while we check upstream the correction of the plan.

Concerning Runtime Verification (RV), the first framework targeting “reactive” (continuous control) programs has been developed by the Monitoring and Checking (MaC) project [12], [13]. It was followed by Lola [14], which adopts the synchronous paradigm and proposes a variant of Lustre dedicated to RV: the sub-domain SRV (Stream RV) is launched. Then NASA introduces the Copilot framework [15], which is now in version 3 [16]. The main novelty of this version is the availability of arrays: we will see that they play a significant role in the PDDL-SCADE proximity. The ROS ecosystem includes several RV packages, among them ROSRV [17] and ROSMonitoring [18].

II. THE TOP-LEVEL DIAGRAM

The next three sections will only sketch the translation from PDDL to SCADE, which is actually quite obvious to anyone who practices both languages. The aim is to pinpoint the reason for the proximity between the two languages. The current section gives a global overview, and because PDDL is a state-transition formalism (an “action language”), the

next two sections describe respectively the states and the transitions.

SCADE is known as a visual language, but in fact this is ‘just’ a “graphical sugar” above a variant of the textual synchronous language Lustre [19]. The diagrams shown afterwards are handmade, the PDDL to SCADE translator only produces text.

The upper-level node for validation and planning is the diagram in figure 1. It is always the same, whether we deal with basic, numerical, or multi-agent problems: only the types of the signals (*action* and *state*) are adapted. It contains the only non-functional operator of the whole model: the *FBY* operator (“Followed BY”, drawn upside down), which is the standard *delay* operator of the synchronous paradigm ($1/z$ in Simulink: the output is the input of the previous cycle). The other operators are pure functions:

- init_state* outputs the initial state, as specified in the PDDL problem,
- goal* checks if its input is a state which complies to the goal, as specified in the PDDL problem,
- update* applies the action to the input state and outputs the resulting state, as specified in the PDDL domain,

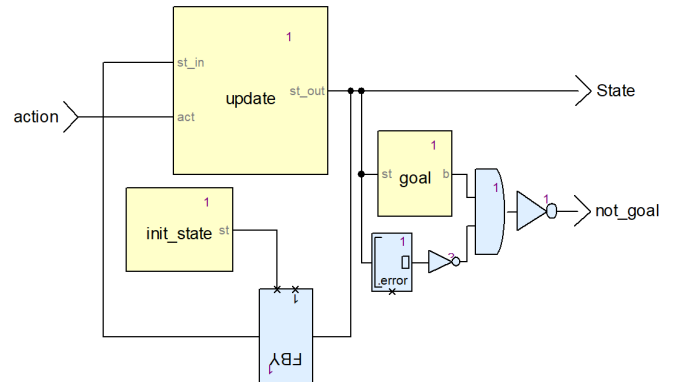


Fig. 1. diagram of the validator

So, this is a completely standard state machine, driven by the flow of actions from the initial state to a final goal-compliant state (at least we hope so). It is not represented in the usual “circles and arrows” state-transition style, but in the more general “state-update” style, because the state-space is usually potentially infinite (as soon as you have an integer variable). In particular, this is the reason why we have not used the SCADE automata diagrams.

The ‘true’ goal is the PDDL goal completed by a check of the *error* field of the state. This boolean field is asserted as soon as something illegal happens, typically when the precondition of an action is *false*. This is a sticky bit (it remains at 1 once set to 1), and the state is also frozen as soon as it is asserted.

The attentive reader may have been surprised to see *not_goal* instead of *goal* on the right of the diagram. It

doesn't matter in our context, it will be explained in the last section.

III. BASIC STATE MODELLING

We illustrate the translation with a classic toy example: the 'buckets' or 'jugs' domain [20]. Before continuing, I advise the reader to take a very quick look at this (non-standard) reference. In this example, two buckets are available, a three gallon one and a five gallon one, near a fountain and initially empty. The goal is to obtain a bucket with exactly four gallons, and three actions are possible: to *fill* a bucket, to *empty* a bucket and to *pour* one bucket into the other one. A subtlety has to be noted regarding this last action: you cannot start pouring and stop arbitrarily in the middle (in order to more directly reach the four gallon goal), because the containers are not graduated. There are only two possibilities:

- either the source bucket can be entirely poured into the destination one (and the source will be empty at the end),
- or it cannot (and the destination will be full at the end).

Here, instead of two jugs, we manage three jugs:

PDDL

```
(:types jug)
(:objects jug3 jug5 jug8 - jug) % problem
```

SCADE

```
const N_jug: uint8 = 3;
type t_jug = uint8;
function is_jug(x: t_jug) returns (b: bool)
  b = x < N_jug;
const
  O_jug3: t_jug = 0;
  O_jug5: t_jug = 1;
  O_jug8: t_jug = 2;
```

The most natural translation is to a SCADE enumeration, but it doesn't work because SCADE arrays cannot be indexed by enumerated values. So any direct subtype of `object` is translated into an integer range $0 \dots N-1$. In this domain we have a static fluent, the capacity of each jug, and a dynamic fluent, the instant content of each jug:

PDDL

```
(:functions
  (amount ?j - jug) - number
  (capacity ?j - jug) - number )
(:init
  % problem
  (= (amount jug3) 0)
  (= (capacity jug3) 3)
  (= (amount jug5) 0)
  (= (capacity jug5) 5)
  (= (amount jug8) 8)
  (= (capacity jug8) 8) )
```

SCADE

```
const capacity: int8 ^N_jug = [3,5,8];
type t_state = {
```

```
  amount: int8 ^N_jug,
  error : bool };
const null_state: t_state = {
  amount: 0 ^N_jug,
  error: false };
function init_state() returns (st: t_state)
  st = ((null_state
    with .amount[O_jug3] = 0)
    with .amount[O_jug5] = 0)
    with .amount[O_jug8] = 8) ;
```

`int8 ^ N_jug` is the type of arrays of `N_jug` signed bytes, and `0 ^ N_jug` is such an array filled with 0's. The type `t_state` contains the dynamic predicates and fluents, and an `error` field that indicates if an error has occurred since the start of the execution of the plan. The constant `null_state` is the object of type `t_state` with each field equal to its default value. The example contains no predicate, but if we also had a dynamic predicate `foo` with say three `jug` parameters, there would have been in `t_state` another field `foo: bool ^N_jug ^N_jug ^N_jug` and in `null_state` the corresponding default value `foo: false ^N_jug ^N_jug ^N_jug` (and yes, it is a 3-dimensional boolean array). The function `init_state` outputs the state object corresponding to the PDDL `:init` clause, and it does this by creating 'mutants' of `null_state`. *There are no side-effects, Lustre is a functional synchronous language.* (`null_state with .amount[1] = 2`) doesn't change `null_state`: it is a new object, identical to `null_state` except for the `.amount[1]` field which is no longer 0 but 2.

Here we find our first modification of PDDL semantics: *numbers are not rationals, but bounded integers as found in the C language* (which is the standard target for critical embedded systems; `uint8 ... int64`). The second modification of PDDL semantics is that *'undefined' doesn't exist: instead, the value 0 is used*. In case of non-initialisation of a function item, the translator prints a warning. If nothing is specified to the translator, `number` is translated by default as `int8`.

The translation of the goal is self-explanatory:

PDDL

```
(:goal (= (amount jug8) 4)) % problem
```

SCADE

```
function goal(st: t_state) returns (b:bool)
  b = st.amount[O_jug8] = 4;
```

The two main points of this section are first that the 'mathematical' numbers of PDDL are not embeddable and must be restricted to more 'computable' numbers, and second that SCADE multi-dimensional arrays permit a very readable translation of the PDDL predicates and functions.

IV. BASIC ACTION MODELLING

The following is a bit technical, and on first reading the reader can go straight to the summary at the end of the section.

The first thing to be clear about is: what is an action? In our jug example, we have one action with 2 parameters: (pour jug1 jug2). So in SCADE, an action will be an array of 3 unsigned bytes: the first is the code of the action and – in our example – has to always be 0 (if not, the validator will set the state error field to true), and the other bytes are the codes of the jugs and have to always be in the range 0 .. 2 (if not, same punishment). More generally, there are often several actions and the array is sized according to the maximum number of parameters among all actions.

SCADE

```
const N_action_param: uint8 = 2;
type t_action = uint8 ^ (1+N_action_param);
```

Let's now have a look at the 'dynamic' action:

PDDL

```
(:action pour
:parameters (?jug1 ?jug2 - jug)
:precondition
  (> (capacity ?jug2) (amount ?jug2))
:effect (and
  (when ; jug1 completely emptied
    (>=
      (- (capacity ?jug2) (amount ?jug2))
      (amount ?jug1))
    (and
      (increase (amount ?jug2)
        (amount ?jug1))
      (assign (amount ?jug1) 0) )
    )
  (when ; jug1 partially emptied
    ... condition omitted ...
    ... effects omitted ...
  ))
))
```

SCADE

```
type
  t_num_set = {
    amount: bool ^N_jug };
  t_num_val = {
    amount: int8 ^N_jug };
const
  null_num_set: t_num_set = {
    amount: false ^N_jug };
  null_num_val: t_num_val = {
    amount: 0 ^N_jug };
function pour(st: t_state;
  jug1: t_jug; jug2: t_jug)
returns(b: bool;
  num_set: t_num_set; num_val: t_num_val)
var
  typing, precondition, when1, when2: bool;
  num_set1: t_num_set;
  num_val1: t_num_val;
let
  typing = is_jug(jug1) and is_jug(jug2);
  precondition = (capacity.[jug2] default 0)
```

```
> (st.amount.[jug2] default 0);
b = typing and precondition;
when1 =
  ((capacity.[jug2] default 0): int8)
  - (st.amount.[jug2] default 0)
  >= (st.amount.[jug1] default 0);
num_set1 = if when1
  then ((null_num_set
    with .amount[jug2] = false)
    with .amount[jug1] = true)
  else null_num_set;
num_val1 = if when1
  then ((null_num_val
    with .amount[jug2] =
      (st.amount.[jug1] default 0)
    with .amount[jug1] = 0)
    else null_num_val;
  when2 = ... ;
  num_set = if when2 ... else num_set1;
  num_val = if when2 ... else num_val1;
tel;
```

The SCADE translation has been simplified: the detection of overflows is missing (an overflow sets to false the boolean output b). But even with overflows, the main point is that **the size of the SCADE translation is only between two and three times the size of the PDDL source**. This is not completely surprising, because it is a *lifted* representation (the alternative is to *ground* the actions before the translation), but it's still remarkable. By comparing line by line the two models, we arrive at the following explanation: the main ingredient of the linearity of the transformation is *the availability of multi-dimensional arrays*.

To understand the translation, we must explain the translation of the effects. Boolean effects, not present in this example, would imply for the function pour two additional outputs, add and del, which would contain for each predicate a boolean array of the same shape indicating the items added (resp. deleted). At this level, the same item could be both added and deleted, and this will make it possible to conform to the standard PDDL semantics.

Numeric effects are a bit more subtle, and the associated semantics is not the standard one. The output num_set contains for each function a boolean array of the same shape, which says whether an item has been *set* or *increased*. The output num_val contains for each function an integer array of the same shape, whose interpretation depends on num_set: set-point or increment. When an item is not updated, num_set contains false and num_val 0: it is incremented by 0! num_set and num_val are built sequentially following the order of declaration of the effects in the PDDL source. It means that if two effects impact the same item (for example two assigns), the effective effect will be the last one in the source, and no error will be raised.

The last domain-specific function is update, which launches the right action to perform with its parameters:

SCADE

```
function update(st_in: t_state; a: t_action)
returns(st_out: t_state)
```

```

var
  precondition: bool;
  num_set: t_num_set;
  num_val: t_num_val;
let
  precondition, num_set, num_val = (case a[0] of
  | 0: pour(st, a[1], a[2])
  | _: (false, null_num_set, null_num_val));
  st_out =
    if precondition and not st_in.error
    then (st_in
      with .amount =
        (num_vect_update<<N_jug>>)(
          st_in.amount,
          num_set.amount,
          num_val.amount ))
    else (st_in
      with .error = true);
tel;

```

`num_vect_update` is a generic function which builds a new function from the old one and the effects to apply. In case of boolean effects, a similar function implements the PDDL semantics ‘add takes priority over del’.

Summary of the section: The main point is that **the size of the SCADE translation is only between two and three times the size of the PDDL source**. By comparing line by line the two models, we arrive at the following explanation: the main ingredient of the linearity of the transformation is *the availability of multi-dimensional arrays*.

V. NUMBERS, INVARIANTS AND VALIDATION

PDDL has unbounded integers (like Python), whereas SCADE has bounded ones (like C). So, we can only say that the set of SCADE plans (those that use sufficiently small numbers) is included in the set of (PDDL) plans. If the inclusion is strict, we have a double problem:

- The SCADE validator may not be able to validate a plan because of big numbers. In formal terms, the SCADE model is not a *refinement* of the PDDL model, and in DO-178 terms, you cannot *validate* the LLRs w.r.t. the HLRs.
- It is a symptom that the PDDL model is a bit weird, in that it isn’t a model of an (enough) bounded part of the real world: in space (the size of the actor movements grid or graph), in time (the number of actions in order to reach the goal), and in “stuff” (the number of robots, jugs, blocks, doors, ...). There is no formal counterpart of this weirdness (because usually, the system level is not formalized), but in engineering terms, either the system requirements are not precise enough, or the HLRs are not a good refinement of them. In any case, we will call this the problem of the *validation of the PDDL model*.

Since the beginning of this paper, the word “validation” has been used with three different meanings, so we will recap them to avoid any confusion:

- VAL (the genuine one) and the SCADE validator do the same thing: *validate* plans,

- the SCADE validator (as a set of LLRs) must be *validated* against the PDDL model (as a set of HLRs),
- the PDDL model (as a set of HLRs) must be *validated* against the system requirements (if you are a software engineer; or against the real world if you are a planning engineer; in fact, it is almost the same problem).

Now, we will deal with the two last bullets in the next two sections. First for LLR validation, we will prove that in any reachable state, the integer variables (the amount of water in each jug) are in fact bounded: this kind of property is called a *(state) invariant*. Then for HLR validation, we will ... in fact look at the same thing, but with two other points of view: system and process. This will open a discussion about the right place of these invariants in the process, and whether PDDL is such a good candidate for the sought-after status of specification language.

A. SCADE model validation ...

The bounding invariants we are interested in are quite obvious, but they depend on the developer’s point of view. The software engineer is focused on its execution platform:

each jug always contain an amount of water between MININT and MAXINT.

An intermediate point of view between software and system may ask for:

each jug always contain an amount of water between 0 and MAXINT.

The system engineer will stick to the real world:

each jug always contain an amount of water between 0 and its capacity.

On the model of the previous section, this last property is true, and it implies the two previous ones if *capacity* \leq *MAXINT*. Moreover, it can be proved by induction. The two proof obligations are the base case (the property holds on the initial state) and the induction step:

- if we start from a state satisfying the invariant,
- and we apply *any* legal action,
- then the resulting state satisfies also the invariant.

Here we must make a small digression to explain what a model checker does. A model is a transfer function between inputs and outputs (like figures 1 and 2), intended to be iterated an arbitrary number of times. One of its outputs is boolean: let’s call it ‘OK’. The purpose of this output is to say that so far, nothing unexpected happened from a design point of view (typically, numeric overflow or functional inconsistency). A model checker will check that, whatever sequence of inputs you impose, the output will never become false. If so, it will simply say: “OK is not falsifiable”. If not so, it will say: “OK is falsifiable, and here is a sequence of inputs which leads to false: ...”⁹.

Let’s return to our problem of inductive proof of an invariant. We model the invariant by a SCADE block of the same

⁹If instead of ‘OK’, you use ‘not_goal’, then a falsifying sequence of inputs satisfies the goal: in other words, it’s a plan. This is the *planning as model-checking* paradigm, and it explains ‘not_goal’ as an output of figure 1, because the block can just as easily be used to check a plan or to generate it.

name, and we ask the model checker of the SCADE toolchain to try to falsify the *inv* output of the diagram in figure 2: this will prove impossible, which demonstrates the invariant.

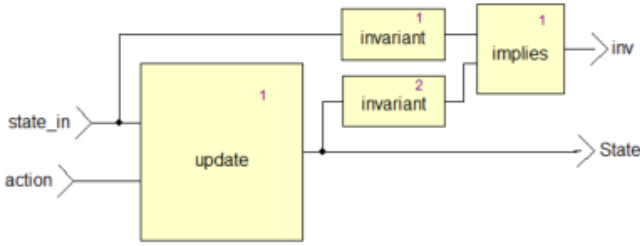


Fig. 2. diagram of the invariant checker

From the process point of view, the tight integration inside the SCADE toolchain of a model checker is very interesting (even if it is not qualified), because we are “sure” we are model-checking the real code and not a more-or-less abstract model of it. Moreover, this is a generic model-checker¹⁰, used in others contexts than SCADE, which adds to the trust that can be placed in him.

Therefore the subject of SCADE validation could be considered to be covered. But in fact we have only seen the tip of the validation iceberg, and to get to the bottom of this, a subtle but fundamental point must be addressed. Here, we are in fact lucky: one the natural invariants (the ‘system’ one) is *inductive* (i.e. it is provable by induction). This is not always the case, and sometimes it is first necessary to find a more complicated inductive invariant, and then prove that this inductive invariant implies the non-inductive bounding box [21]. The simplest example is a rotation (which by the way is just a *linear* transformation): no box is inductive, so we must first use an included disk (which is inductive: any rotation of a disk is itself¹¹). On our PDDL model of the jugs problem, the ‘software’ invariant `amount : int8`¹² is *not* inductive, because when we pour jug1 into jug2, $(-128, -128, \dots) \rightarrow (0, -256, \dots)$ is a legal transition. If you’re bothered by the fact that a bucket can hold a negative number of gallons, imagine that it’s actually a bank account (and a debt has been transferred from account 1 to account 2). The ‘system-software’ invariant `amount : uint8` is inductive, but if we slightly change the PDDL model and suppress the precondition of the `pour` action (which is functionally superfluous), this invariant is *no more* inductive. So, even if the invariants depend essentially on the intrinsic problem posed, the fact that they are inductive also depends on the way we model. All this shows that if a PDDL model is not predisposed to be implemented with bounded numbers, the validation of this implementation can be unintuitive or even difficult. The next section describes what this predisposition could be.

¹⁰<https://www.prover.com/products/prover-psl>. For example, the very same model-checker is used in Simulink: <https://www.mathworks.com/help/sldv/ug/acknowledgments.html>, of course with a specific integration.

¹¹This is the rationale behind the *Ellipsoid* abstract domain in the Astrée static analyzer.

¹²`int8` is the range $-128..127$, `uint8` is $0..255$.

B. ...benefits from PDDL model validation

So far we have seen invariants as means of validating the SCADE model w.r.t. the PDDL model: it is correct to use bounded numbers? But they are also important elements of the PDDL model validation, and we will try to take advantage of it. Let’s recall that this validation consists of being able to convince that the formal model is ‘faithful’ to the informal system specification. For this purpose, it is recognized that, beyond a certain level of complexity, a simple state-transition model is insufficiently meaningful, and that it must be supplemented by other formal properties: they are called ‘safety’ properties when they characterize states (invariants), and ‘liveness’ properties when they characterize paths (multi-step transitions). That’s why formal software specification and development methods, like the B-method [22], set the invariant as the central element of a specification (at the same level as the very definition of what a state is), around which the transitions (named ‘operations’ in B) revolve. A good example is the invariant we used in the previous section:

each jug always contain an amount of water between 0 and its capacity.

It is just as relevant as a SCADE validation element or as a PDDL validation element. We believe that this is not an isolated fact and that it is not due to chance, and we propose to generalize it to the planning models used in certified products:

a PDDL model must be accompanied by a ‘bounding’ invariant, which makes it easy to justify the software implementation of its plan validator.

It is stated and proven at the HLR level by the engineer in charge of the PDDL model, not at the design level. The design level has only a simple justification to produce in order to validate the use of software numeric data types. This is consistent with what is practiced during a formal software development: when the B-method is used, invariants and operations are written and modified by the same person at the same time, and during design and maintenance the machine (the B name for a PDDL model) is continuously kept consistent¹³:

We have used the expression ‘accompanied’, because invariants cannot be included in the PDDL model: this is not provided for by the standard, in other words *invariance* is (unfortunately) not a PDDL concept!

A related concept exists, with which it should not be confused: the *trajectory constraint* (always `<constraint>`). It restrains the acceptable plans (and therefore the search space) to those that satisfy the constraint in each state passed through. It is in fact syntactic sugar: it can be “compiled away” (planning jargon) by increasing the state with an additional boolean variable, which will memorize the respect of the constraint by all actions performed. The set of reachable states is not modified, and in each reachable state that is not in a legal plan, the constraint is not ensured. An invariant of a planning model is

¹³This is a precursor of Test-Driven Development (TDD).

not a restriction, it is a consequence of the model that holds in all reachable states.

C. Related works

[23] are the first to propose to use a model-checker (SMV) as a planner. For the record, it is this interaction between planning and model-checking that is at the origin of a generic and successful technique called *bounded* or *SAT-based* model checking, used by the SCADE’s model checker [24]. This is not necessarily the most efficient approach, but it is often the easiest and quickest way to obtain a planner for a new planning language, and it has been used for all areas of deterministic planning (e.g. linear hybrid planning [25] with UPPAAL, hierarchical planning [26], full hybrid planning [27]) and some areas of nondeterministic planning [28].

Only one year later, this planning-as-model-checking principle was diverted from its initial use by NASA to do model debugging, first with classical model-checkers (including SMV and SPIN) [29] (this is exactly what is done in the former section), then with UPPAAL for temporal planning [30]¹⁴. [31] propose a new validation concept: the *(un)desirable plan*. Usually, a safety property is checked on all reachable states and a liveness property (e.g. if an image is taken, it is eventually uploaded) on all legal paths. Here, these properties are checked only on plans (which must all be desirable). Again, it must not be confused with trajectory constraints.

From the process point of view, [32] are the first to propose to prove invariants “*for defining consistent axiomatizations of planning domains*”. [33] take up the proposal, and one of the steps of their modelling method is described as follows: “*A set of state invariants are constructed. These state invariants are analogous to those used in model based formal specifications of software (e.g. as in VDM ...), The availability of an invariant for the domain promotes the effectiveness of tools to support the validation ... of the domain model*”. [34] go further by modeling a domain directly in a formal software development language where invariants have a prominent place: the B language.

But unfortunately, planning languages struggle to keep up. The first version of PDDL, defined for the AIPS-98 planning competition [35] permits a declaration *(:timeless literal₁ .. literal_n)* that “*consists of a list of literals that are taken to be true at all times ... it should be impossible for any action to change them*”¹⁵. But the next version [36] trims some ‘unused’ features, so *(:timeless ...)* disappears, and in fact no invariant has ever been declared in PDDL.

VI. CONCLUSION AND FOLLOW-UP

This approach to trustworthy action planning is well aligned with the specification (HLR) and design (LLR) steps of the DO-178 guidelines, therefore it is a priori achievable at

¹⁴It can be noted that this first use of UPPAAL as a validation tool precedes (by more than one year) its first use as a planner.

¹⁵This may sound odd, because an invariant usually is more complex than a literal. But this version of PDDL permits to declare ‘derived’ predicates, that can capture this complexity.

reasonable (or at least usual) costs. However, considering a PDDL model alone as HLRs seems questionable, and we recommend associating it with appropriate invariants, in order to facilitate LLR validation.

Moreover, the readability of the translations (we have given a plausible explanation of this) and their modularity (due to the functional paradigm of Lustre) suggests that the approach is in fact applicable to more general planning paradigms than the basic one considered here. The one we are working on is multi-agent planning, with very promising use-cases (for example the MAPF [Multi-Agent Path Finding] problem). This could be demonstrated in a future edition of a robotic challenge like CoHoma [37], if a fully autonomous mode is activated, in order to reduce the cognitive load of the supervisors in charge of safety.

APPENDIX

What follows is a very crash course on PDDL. A PDDL model is a state-transition view of the real world. Another way to say this is that PDDL is an “action” language. Here we modelize a “maintainable light” which counts the number of cycles, in order for example to trigger predictive maintenance. The state is a couple of variables `light_is_on` (boolean) and `counter` (integer). There are two actions: `switch_on`, enabled when the light is off, and the dual `switch_off`.

PDDL domain

```
(define (domain light_with_counter)
  (:predicates
    (light_is_on) )
  (:functions
    (counter) )
  (:action switch_on
    :precondition (not (light_is_on))
    :effect (and
      (light_is_on)
      (increase (counter) 1) ) )
  (:action switch_off
    :precondition (light_is_on)
    :effect (and
      (not (light_is_on))
      (increase (counter) 1) ) )
)
```

The preconditions are propositions that must hold before the action is run. The effects may look like propositions (post-conditions), but they are not: they are lists of *side-effects* on the state. The ‘and’ in `:effect (and ...)` is not a logical conjunction, it simply says that what follows is a list of effects. Be careful, `(not (light_is_on))` doesn’t mean that it *reverses* `light_is_on`, but that `light_is_on` becomes false. Here is now a small problem on this domain:

PDDL problem

```
(define (problem light_2)
  (:domain light_with_counter)
  (:init
    (light_is_on)
    (= (counter) 36) )
  (:goal
    (not (light_is_on))
  )
```

```
(> (counter) 38) )
```

This problem has a solution in three steps:

PDDL plan

```
(switch_off) ; counter == 37  
(switch_on) ; counter == 38  
(switch_off) ; counter == 39
```

If on the other hand we change the goal and we ask for a light on and a counter at 39, then there is no solution, because when the light is on, the counter is even (this is an invariant).

ACKNOWLEDGEMENTS

We thank Jean-Marie Courteille for emphasizing the difference between trust and explanation, Marc-Emmanuel Coupvent des Gravières for the reference [1] on the same topic, and an anonymous referee for pointing out the fact that non-optimality and non-existence of a plan are not necessarily a safety problem.

REFERENCES

- [1] M. Fox, D. Long, and D. Magazzeni, “Explainable planning,” *arXiv preprint arXiv:1709.10256*, 2017.
- [2] T. Bylander, “The computational complexity of propositional strips planning,” *Artificial Intelligence*, vol. 69, no. 1-2, pp. 165–204, 1994.
- [3] C. Bäckström and P. Jonsson, “Time and space bounds for planning,” *Journal of Artificial Intelligence Research*, vol. 60, pp. 595–638, 2017.
- [4] P. Haslum, N. Lipovetzky, D. Magazzeni, and C. Muise, “An introduction to the planning domain definition language,” *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 13, no. 2, pp. 1–187, 2019.
- [5] R. Howey and D. Long, “Val’s progress: The automatic validation tool for pddl2.1 used in the international planning competition,” in *Proc. of ICAPS Workshop on the IPC*. Citeseer, 2003, pp. 28–37.
- [6] H. Andersson and G. Hagelin, “Computer controlled interlocking system,” *Ericsson Review*, vol. 58, no. 2, 1981.
- [7] A. Sjöberg, “Automatic train control,” *Ericsson Review*, vol. 58, no. 1, 1981.
- [8] N. Theuretzbacher, “Using ai-methods to improve software safety,” *IFAC Proceedings Volumes*, vol. 19, no. 11, pp. 99–105, 1986.
- [9] Audi, BMW, Daimler, Porsche, and VW, “Standardisiertes e-gas überwachungskonzept für benzin und diesel motorensteuerungen, version 5.5,” 2013.
- [10] B. Lussier, M. Gallien, J. Guiochet, F. Ingrand, M.-O. Killijian, and D. Powell, “Fault tolerant planning for critical robots,” in *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN’07)*. IEEE, 2007, pp. 144–153.
- [11] M. Machin, J. Guiochet, H. Waeselynck, J.-P. Blanquart, M. Roy, and L. Masson, “Smof-a safety monitoring framework for autonomous systems,” *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 48, no. 5, pp. 702–715, 2018.
- [12] I. Lee, S. Kannan, H. Ben-Abdallah, O. Sokolsky, M. Kim, and M. Viswanathan, “A monitoring and checking framework for run-time correctness assurance,” 1998.
- [13] S. Kannan, M. Kim, I. Lee, O. Sokolsky, and M. Viswanathan, “A retrospective look at the monitoring and checking (mac) framework,” in *Runtime Verification: 19th International Conference, RV 2019, Porto, Portugal, October 8–11, 2019, Proceedings 19*. Springer, 2019, pp. 1–14.
- [14] B. d’Angelo, S. Sankaranarayanan, C. Sánchez, W. Robinson, B. Finkbeiner, H. B. Sipma, S. Mehrotra, and Z. Manna, “Lola: runtime monitoring of synchronous systems,” in *12th International Symposium on Temporal Representation and Reasoning (TIME’05)*. IEEE, 2005, pp. 166–174.
- [15] L. Pike, A. Goodloe, R. Morisset, and S. Niller, “Copilot: a hard real-time runtime monitor,” in *International Conference on Runtime Verification*. Springer, 2010, pp. 345–359.
- [16] I. Perez, F. Dedden, and A. Goodloe, “Copilot 3,” Tech. Rep., 2020.
- [17] J. Huang, C. Erdogan, Y. Zhang, B. Moore, Q. Luo, A. Sundaresan, and G. Rosu, “Rosrv: Runtime verification for robots,” in *Runtime Verification: 5th International Conference, RV 2014, Toronto, ON, Canada, September 22-25, 2014, Proceedings 5*. Springer, 2014, pp. 247–254.
- [18] A. Ferrando, R. C. Cardoso, M. Fisher, D. Ancona, L. Franceschini, and V. Mascardi, “Rosmonitoring: a runtime verification framework for ros,” in *Towards Autonomous Robotic Systems: 21st Annual Conference, TAROS 2020, Nottingham, UK, September 16, 2020, Proceedings 21*. Springer, 2020, pp. 387–399.
- [19] P. Caspi, D. Pilaud, N. Halbwegs, and J. Plaice, “Lustre: a declarative language for real-time programming,” in *Proceedings of the 14th ACM SIGACT-SIGPLAN symposium on Principles of programming languages, 1987*, pp. 178–188.
- [20] B. Willis, J. Irons, and S. L. Jackson, “Die hard with a vengeance,” <https://www.youtube.com/watch?v=2vdf6NASMIIE>, 1995.
- [21] A. Miné, J. Breck, and T. Reps, “An algorithm inspired by constraint solvers to infer inductive invariants in numeric programs,” in *European Symposium on Programming*. Springer, 2016, pp. 560–588.
- [22] J.-R. Abrial, *The B-book: assigning programs to meanings*. Cambridge university press, 1996.
- [23] A. Cimatti, E. Giunchiglia, F. Giunchiglia, and P. Traverso, “Planning via model checking: A decision procedure for ar,” in *European Conference on Planning*. Springer, 1997, pp. 130–142.
- [24] P. A. Abdulla, J. Deneux, G. Stålmarck, H. Ågren, and O. Åkerlund, “Designing safe, reliable systems using scade,” in *International Symposium On Leveraging Applications of Formal Methods, Verification and Validation*. Springer, 2004, pp. 115–129.
- [25] H. Dierks, G. Behrmann, and K. G. Larsen, “Solving planning problems using real-time model checking (translating pddl3 into timed automata),” in *AIPS-Workshop Planning via Model-Checking*. Citeseer, 2002, pp. 30–39.
- [26] U. Kuter, D. S. Nau, M. Pistore, and P. Traverso, “A hierarchical task-network planner based on symbolic model checking,” in *ICAPS*, 2005, pp. 300–309.
- [27] S. Bogomolov, D. Magazzeni, S. Minopoli, and M. Wehrle, “Pddl+ planning with hybrid automata: Foundations of translating must behavior,” in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 25, 2015, pp. 42–46.
- [28] A. Cimatti, M. Pistore, M. Roveri, and P. Traverso, “Weak, strong, and strong cyclic planning via symbolic model checking,” *Artificial Intelligence*, vol. 147, no. 1-2, pp. 35–84, 2003.
- [29] J. Penix, C. Pecheur, and K. Havelund, “Using model checking to validate ai planner domain models,” in *Proceedings of the 23rd Annual Software Engineering Workshop, NASA Goddard*, 1998.
- [30] L. Khatib, N. Muscettola, and K. Havelund, “Verification of plan models using uppaal,” in *Formal Approaches to Agent-Based Systems: First International Workshop, FAABS 2000 Greenbelt, MD, USA, April 5–7, 2000 Revised Papers 1*. Springer, 2001, pp. 114–122.
- [31] M. H. Smith, G. J. Holzmann, G. C. Cucullu, and B. D. Smith, “Model checking autonomous planners: Even the best laid plans must be verified,” in *2005 IEEE Aerospace Conference*. IEEE, 2005, pp. 1–11.
- [32] W. Stephan and S. Biundo, “A new logical framework for deductive planning,” in *Proceedings of the 13th international joint conference on Artificial intelligence-Volume 1*, 1993, pp. 32–38.
- [33] T. L. McCluskey and J. M. Porteous, “Engineering and compiling planning domain models to promote validity and efficiency,” *Artificial Intelligence*, vol. 95, no. 1, pp. 1–65, 1997.
- [34] M. M. West, D. E. Kitchin, and T. McCluskey, “Validating planning domain models using b-amm,” *PlanSIG, 21st/22nd Nov*, 2002.
- [35] M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins, “Pddl-the planning domain definition language,” Yale center for Computational Vision and Control, Tech. Rep. TR-98-003/DCS TR-1165, 1998.
- [36] F. Bacchus, “Subset of pddl for the aips2000 planning competition,” *The AIPS-00 Planning Competition Committee*, 2000.
- [37] Battle-Lab-Terre, “Challenge cohoma 2,” <https://www.defense.gouv.fr/sites/default/files/aid/Support%20CoHoMa%202023.pdf>, 2022.

Session We.4.B

Testing

Wednesday 12th June

15:00

—

Lauragais Room

Design by contract formal verification for automotive embedded software robustness

Vassil Todorov
Stellantis
Poissy, France

Alin Mihalache
Stellantis
Poissy, France

Azzedine Azil
Stellantis
Poissy, France

Armando Hernandez
Stellantis
Poissy, France

Abstract—Preventing software failures is of high importance for safety or security related embedded software. Among the most critical defects are runtime errors such as buffer overflows, accessing data outside the allocated memory, division by zero or data races. The ISO 26262 functional safety standard for road vehicles requires to use static code analysis for unit and integration verification but this method may be unsound and cannot always guarantee exhaustiveness i.e., some defects can still be present in the code. In 2018, ISO 26262 was updated and introduced a recommendation for static code analysis based on abstract interpretation. Abstract interpretation is a formal method which means that it can guarantee mathematically the absence of runtime errors in an exhaustive manner. To be exhaustive it uses approximation algorithms that can bring a substantial number of false alarms. For this reason, this method is not largely deployed in the automotive industry today.

In this paper, we propose to introduce a *design by contract* approach to provide the abstract interpretation static analyzer additional information for the input variables and the parameters to increase its precision and significantly reduce the number of false alarms. For the outputs, we use the analyzer to prove contracts are compliant to the ranges defined by the specification. We automated the procurement of contracts from different sources: a database defining the software architecture, CAN network signals definition or the AUTOSAR ARXML interface definition files. Finally, we provide the results obtained for our production code for analyses with or without contracts and show how effective is their use.

Keywords-Formal verification · Abstract interpretation · Design by contract · Safety · Security · AUTOSAR · Embedded software architecture

I. INTRODUCTION

A. Background on automotive embedded software

Automotive embedded software constitutes a critical component within modern vehicles, facilitating the operation of various systems and enhancing the driving experience. Over the past few decades, the automotive industry has witnessed a remarkable transformation driven by advancements in embedded systems technology. We moved from an era where the car had zero code and the most important was the engine horsepower to a car that is fully programmable containing more than 100 million lines of code. This car is named "Software-Defined Vehicle" or SDV for short [1]. Despite the increase of complexity, the development of automotive embedded software presents unique challenges owing to the stringent requirements for safety, reliability, and real-time performance. The verification and validation of this software

requires sophisticated methodologies, tools, and best practices to ensure it is performing exactly as specified and is compliant with regulatory requirements and industry standards. One of these methodologies is to use formal methods, method also recommended by the automotive safety standard ISO 26262 [2].

B. Importance of formal verification in automotive safety

Ensuring the safety and reliability of automotive embedded software is paramount, given its direct impact on vehicle operation and passenger well-being especially for autonomous driving and battery management systems.

Formal verification techniques enable exhaustive analysis of software designs and implementations against specified requirements and safety standards. By mathematically modeling system behaviors and properties, formal verification methods facilitate the detection of logical inconsistencies, functional errors, and potential hazards in automotive software systems. An overview of the formal methods that can be used in the automotive software development as well as their application on some industrial use cases is presented in [3].

In the context of automotive safety-critical systems, formal verification offers a systematic approach to identifying design flaws and software bugs that could lead to hazardous situations, such as unintended acceleration, loss of control, or malfunctioning safety features. By detecting and addressing potential safety risks early in the development lifecycle, formal verification contributes to the prevention of costly recalls, accidents, and liabilities associated with software-related failures.

C. Objectives of the paper

The primary objective of this paper is to investigate the application of Design by Contract (DbC) formal verification techniques in the context of safety-critical automotive embedded software development. We focus on two formal methods: abstract interpretation and model checking. Abstract interpretation is used on the code level and model checking is used on the model level. In our work, a contract represents essentially assumptions over the inputs of a function and guarantees provided by the outputs assuming the input contracts. These assumptions and guarantees represent a list of values for an input/output interface or an interval of values. The assumptions' role is to reduce the state space, producing more accurate and fast analyses. The guarantees' role is to prove

that a given output will always produce the specified values and nothing else. After positioning this paper with regards to the state of the art, we propose:

- To investigate the possible sources to find contracts in an automotive context: ARXML, architecture, network signals definition
- To formalize the contracts
- An algorithm to calculate automatically the ranges of possible values and checking the real existence of the interfaces in the code
- An intermediate format to store the contracts
- Some safety applications of the extracted contracts

We present the practical application of DbC and conclude for which type of software the contracts helped reduce the number of false alarms for static analyses based on abstract interpretation. Finally, the paper identifies key challenges, opportunities, and future directions.

II. OVERVIEW OF DESIGN BY CONTRACT (DBC)

The Design by Contract (DbC) approach represents a paradigm shift in software engineering, emphasizing the importance of precise specification, contractual agreements, and runtime validation of software components.

A. Contracts for Deductive Proof

DbC was first introduced by Bertrand Meyer [4] [5] following earlier ideas from Floyd-Hoare logic [6] [7]. Floyd-Hoare logic assigns meaning to sequential imperative programs in the form of triples of assertions $\{P\}Q\{R\}$ consisting of a *precondition* on program states and inputs $\{P\}$, and a *postcondition* $\{R\}$ on program states and outputs after transformation by the code Q . The logical meaning of this triple corresponds to: if P is true, then after executing program Q , R will be true if Q terminates. The calculus of Hoare's triples is, in general, undecidable.

The proving by application of Hoare's rules is an intellectual process and is not tool driven. It is up to the author of the proof to define the correct properties between each instruction of the program and to establish its demonstration by applying the different theorems. This activity is not adapted to process thousands of lines of code in an acceptable time.

An initial automation of the process of proving programs was brought by the calculation of the *WP* (*Weakest Precondition*) from Dijkstra [8]. The principle consists in automatically calculating the most general property $WP(S,P)$ holding before a statement S such that property P holds after the execution of S :

$$WP(S, P) \{S\} P$$

The calculus of *WP* is defined for each instruction. The proof process consists in calculating *WP* by going backward from the end of the program for which we want to prove P , up to the beginning. For full correctness, S must terminate.

The returned predicate from the *WP* calculation can rapidly become rather complex. Efficient (quadratic instead of exponential) verification condition generation (including *WP*

generation) were proposed in the following papers [9]–[11]. To automate the process, all modern tools based on *WP* are using automatic theorem provers as back-end. We can cite, for example Alt-Ergo [12], Colibri¹, CVC4 [13], Yices2 [14], Z3 [15].

One application of contracts for automotive embedded software using *WP* and Frama-C [16] was published in [17]. The authors present the limitations of the approach and propose some techniques to overcome some of them to succeed with the automatic proof. Nevertheless, this method has difficulties to scale with the current size of automotive software modules and in this paper, we will not focus on it.

B. Contracts for Object-Oriented Programming

Bertrand Meyer originally proposed to use contracts for Object-Oriented programming. They operate at a higher level of abstraction, focusing on the program behavior, while the axiomatic semantics used for deductive proof operates at a lower level, focusing on formal logical assertions and proofs. He introduced contracts in the Eiffel programming language [18]. They expose the relationships between systems in terms of preconditions and postconditions on operations and invariants on states. A contract on an operation asserts that, given a state and inputs which satisfy the precondition, the operation will terminate in a state and will return a result that satisfy the postcondition and respects any required invariant properties. Meyer's work was also extended to Java via iContract [19]. Today, the C programming language is the state of the art for automotive safety-critical software and our work will not focus on the Meyer's contracts as they cannot apply on it.

C. Contracts for Service-Oriented Architectures

Beugnard et al. [20], inspired by contracts in real life, proposed to use four classes of contracts in the software component world for service-oriented architectures with increasingly negotiable properties: basic or syntactic, behavioral, synchronization, and quantitative. The basic layer specifies operations, their inputs, outputs and possible exceptions. The behavior layer describes the abstract behavior of operations in terms of their preconditions and postconditions. The third layer, synchronization, corresponds to real-time scheduling of component interaction and message passing. The fourth, quality of service (QoS) level, details non-functional aspects of operations. This work is rather theoretical and as far as we know this method is not integrated in the state-of-the-art tools.

D. Contracts for System Design

System design is the process of defining the architecture, components, modules, interfaces, and data for a system to satisfy specified requirements. It is used in most of the industries when designing a new system. In [21], the authors argue that current methods and practices for verification are not scaling because of the increasing complexity of the systems. They present recent approaches for contract-based design like the "orthogonal" one and propose to have a unified treatment

¹Colibri: <http://smtcomp.sourceforge.net/2018/systemDescriptions/COLIBRI.pdf>

of the topic. Contracts are precisely defined and can be used without ambiguity. They also provide a link between interfaces and contracts to show similarities. They provide some examples applied to system requirements engineering and AUTOSAR. The first one uses Boolean conditions and a research tool named MICA. The second example can be applied in the industry by using model checking and the synchronous observers technique [22].

E. Contracts for Runtime Error Detection

This application of contracts for runtime error detection is inspired from what is usually called "runtime assertion check" [23]. For C language the Frama-C plugin E-ACSL [24] automatically translates an annotated C program into another program that reports a failure whenever an annotation is violated at runtime. If no annotation is violated, the behavior of the new program is the same as that of the original one. The benefits of this approach are several: monitoring the execution of the C program and reuse the contracts provided by annotations in the program with other tools of the Frama-C framework e.g., WP. Sometimes the formal proof cannot succeed so transforming the contracts (annotations) into executable specification can complement the proof for difficult goals thus establishing a link between static analysis and monitoring tools.

F. Contracts for Static Code Analysis

Static code analysis is the analysis of computer programs performed without executing them. Today, it is generally realized by an automated tool under a wide range of criteria thus increasing code quality and robustness. Depending on the depth and exhaustiveness of the analysis, the static analysis tools can be classified in three categories:

- **Syntax checkers** or **linters** are limited to analyze the program by using pattern matching. A lot of the MISRA C and C++ rules can be checked at this level. For example, MISRA C Rule 17.7 says that "The value returned by a function having non-void return type shall be used". Checking only the function call syntax is sufficient to see if the return value is used by the code.
- **Unsound static analyzers** report semantic errors in the code, such as runtime errors (division by zero, buffer overflow, etc.). They can also report about data races or stack overflows. These tools use heuristics and reduce generally the depth of the analysis to shorten the analysis time and scale. Thus, they produce false negatives (some true defects are not detected) but they can also bring false positives (alarms that are not true defects). Examples of such tools are Coverity, CodeSonar, Klocwork, Parasoft C/C++test, Polyspace Bug Finder and QAC.
- **Sound static analyzers** are mostly based on a formal method called "abstract interpretation" [25]. This method allows to prove the absence of runtime errors from the class of defects under consideration without missing any defect. These tools can report not only software defects but can also be used to prove functional assertions e.g.,

the output values are always produced in each specified range. False positives are still possible, but these tools can be finely tuned to improve their precision by providing the ranges of the input values thus significantly reducing their number. Examples are Astrée, CodePeer, Polyspace Code Prover and TIS Analyzer.

Contracts for static analysis can only be used with sound static analyzers as they can consider constraints over the inputs and prove the output values are always within a given range. These contracts are sometimes called in the literature "assume/guarantee" or "precondition/postcondition". The benefits are:

- Reduced number of false alarms. Providing ranges for the input values significantly reduces the explored states.
- Guarantee that the outputs are always produced within the specified ranges. This can be obtained by additional assertions provided to the static analyzer and proving the code is safe.

This article focuses on the topic of generating and using contracts for static analysis of industrial software and the benefits for the safety we take from them. As far as we know there is no such complete experiment published in the scientific literature. Our approach can be applied in the industry to adopt sound static analysis and save time for issue analysis.

III. DESIGN BY CONTRACT FOR SAFETY APPLICATIONS

In this chapter we describe a methodology for the embedded software to illustrate how abstract interpretation is used to prove that the software activates the safe state. Subsequently, an example on the safety function will be presented.

We assume in this example that the safety function is based on several ECU car systems (ECU1, ECU2 and ECU3) and the control algorithms are principally made by ECUx. The safety function is described in Fig. 1.

We can imagine also that this functionality must contain the ECUx which established a gateway between CAN networks (ECU1 and ECU2) and LIN network (ECU3).

For example, for this safety function the most important hazardous event is "unexpected OFF of the safety function in the driving conditions". At software level, it means that the output data (Command) of the safety function must not be set to OFF unexpectedly. The safety goal at the vehicle level is ASIL B level. The limp-home or safety state associated is "in case of detected failure that could lead to the violation of the safety goal, the Command shall be set to ON". We suppose also for this example that the entire vehicle ECU for this function (ECUx, ECU1, ECU2 and ECU3) inherit an ASIL B because they can directly contribute to the loss of the Command.

In this case, the software safety requirement allocated to ECUx has an ASIL B associated to the risk "unexpected Command to OFF in the driving conditions". The implementation on this safety goal in the software uses partitions, monitoring principles and ISO 26262 standard. For the safety function in

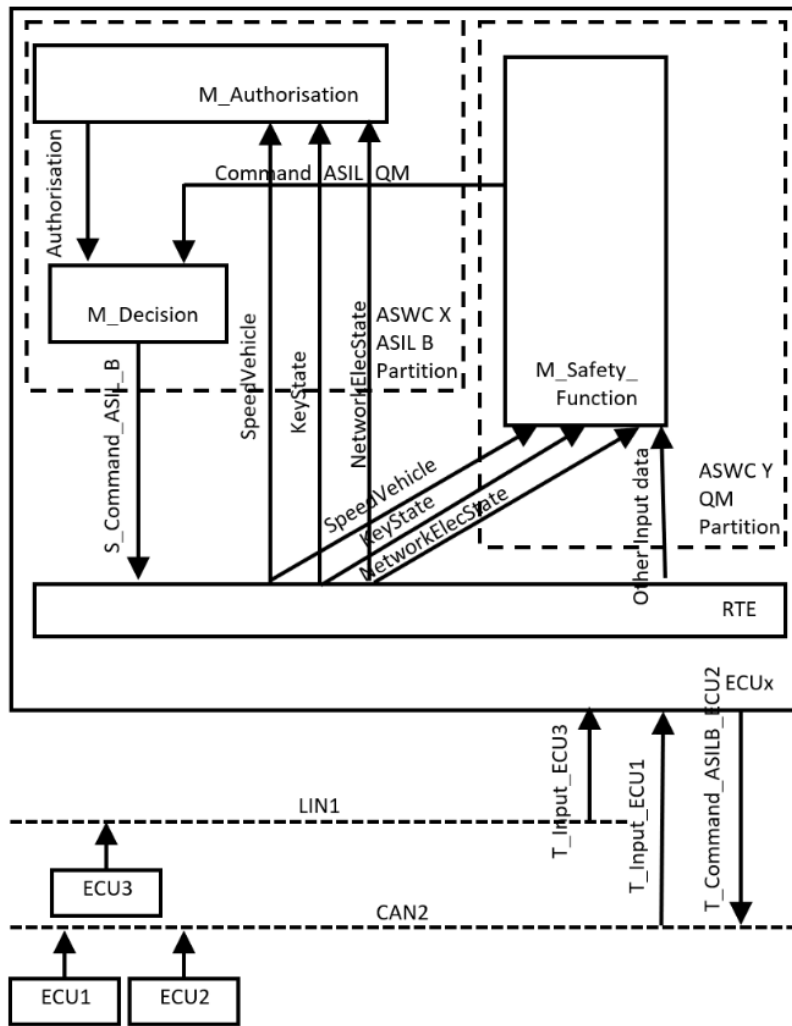


Fig. 1. Safety function: Software architecture of ECUx based on AUTOSAR

the ECUx, a monitoring based on partitions is used: one QM partition and one ASIL B partition. All the functionalities of the safety function, critical and non-critical software modules are included into the ASIL QM partition. The ASIL B partition for the safety function contains only the monitoring on the critical parts. For example, the monitoring may be made by two functions (see figure 1):

- Authorization function: the main goal for the first function is to authorize the switch OFF the Command in the case of the critical part, shown in figure 1.
- Decision function: the aim of the second function is to apply this authorization on the Command which comes from the ASIL QM partition (Command_ASIL_QM in figure 1) to make the Command into the ASIL B partition (Command_ASIL_B in the figure 1).

A. Authorization function – monitoring

To better understand our approach let's take an example for this authorization function described in figure 2. We can

imagine that the Command can switch to OFF in the case of critical part if the key is in the stop position or if the network electrical state at a low level only if the speed of the vehicle is lower than a threshold of 1km/h.

The step 1 of this algorithm is to acquire the software flow made by other software modules or to decode frames. For our example:

- SpeedVehicle for the vehicle speed value,
- KeyState for the key state,
- NetworkElecState for the network electrical state.

The step 2 is dedicated to an out-of-range detection (OOR). The safety mechanisms for error handling is the passage of the data into a safe state. The step 3 is to determine the specific situations where the switch to OFF the Command are authorized. In our example, these specific situations are protected by the vehicle speed:

- Key state at stop position: for the safety stand situation, we authorized the switch to OFF the Command only if the key state is in the stop position and the vehicle speed

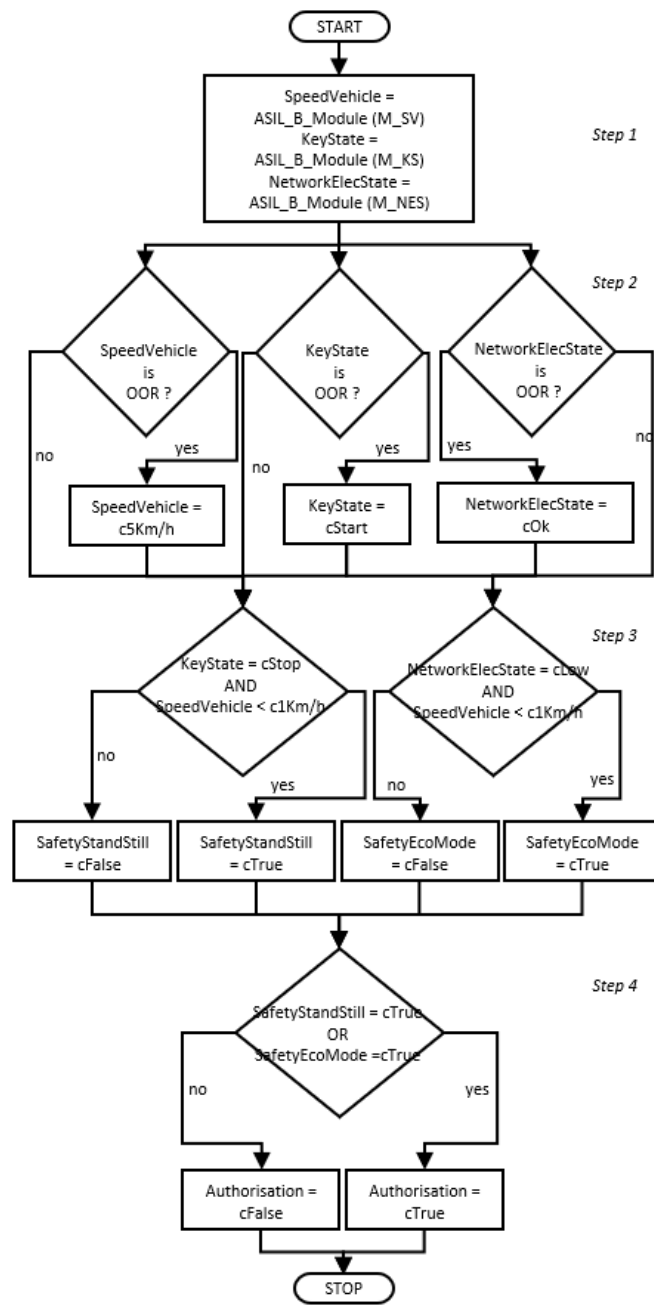


Fig. 2. Flowchart diagram of the authorization function

is lower than 1 km/h,

- Network electrical state at low level: for the safety economic mode situation we authorized the switch to OFF the Command only if the network electrical state is at the low level and the vehicle speed is lower than 1 km/h.

The step 3 is the main condition of the algorithm. The switch to OFF the Command authorization is true only if safety stand situation or safety economic mode situation.

B. Decision function – monitoring

Let's also take an example for decision function described in figure 3.

The step 1 for this algorithm is to acquire the software flow produced by other software modules:

- Command_ASIL_QM for the request coming from the ASIL QM partition,
- Authorization from the previous function.

In step 2 we initialize the output, the Command_ASIL_B as ON – defensive programming. Step 3 is the same as step

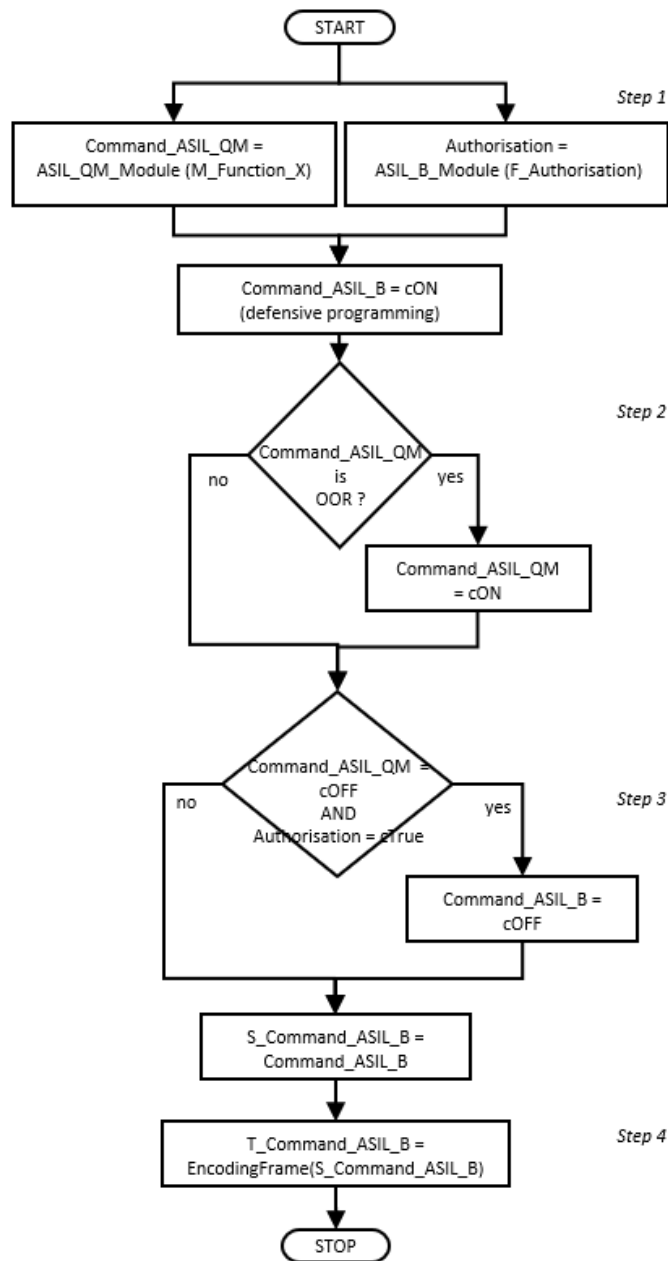


Fig. 3. Flowchart diagram of the decision function

2 for authorization function. Step 4 is the main condition of the algorithm. The output (Command_ASIL_B) is OFF only if no request from the ASIL QM partition and authorized to switch OFF by the ASIL B partition (see figure 1).

C. Using Contracts

To prove that the safe state is indeed activated in case of an error, we have introduced formal methods. The idea is to provide out-of-range values for the inputs as contracts to a static analysis tool based on abstract interpretation and prove that the software will always activate the safe state when it receives this data. When the software activates the safe state,

we use a specific Boolean output that can be proved to be always *true*.

IV. SOURCES FOR CONTRACTS

As presented in the previous chapters, we focus in this paper on the procurement of contracts that will help a static analyzer based on abstract interpretation improve its precision but also increase software robustness and guarantees. These contracts are of two types: input constraints and output guarantees. With the constraints, we save time for results analysis by removing a lot of false alarms. There are two types of inputs:

- Variable inputs: data received that can change over time.

- Constant inputs: used for calibration or configuration. They cannot change during program execution.

With the guarantees, we prove the outputs will always be produced within a given set of values. The target for these contracts is the software component.

In this chapter we discuss, where contracts can be found and how to extract them automatically. We provide an algorithm to select the most precise set of values based on multiple sources.

A. System Design Interface Definitions

First, we start by exploring the system design level. Very often, the system design engineers propose some ranges for the data used for the system design. It can be formatted as an Excel table showing minimum and maximum values for each data flow without further details about concrete types to be used.

B. Software Architecture Design

When the system engineer has validated its design, it goes to the software engineer who is responsible for the software architecture and embedding this design in a car ECU². At that stage, all the interfaces are well determined, the system is cut into software components connected with each other. Different tools from custom to commercial tools can be used on that stage to declare all the interfaces and ports of a software module. The extraction can be done automatically by exporting the contracts in CSV or JSON formats.

C. AUTOSAR Architecture

AUTOSAR (Automotive Open System Architecture) is a standardized automotive software architecture jointly developed by automobile manufacturers, suppliers, and tool developers. It aims to establish an open and standardized software architecture for automotive ECUs to ensure compatibility, scalability, and reusability of software components across different vehicle platforms and manufacturers. ARXML (AUTOSAR XML) is a file format used within the context of AUTOSAR as a standardized format for representing and exchanging AUTOSAR-compliant software descriptions, configurations, and other relevant data. The interaction between different components is presented in Fig. 4

In our context we want to verify the Software Components as the other services are provided by the operating system's provider who is responsible for their verification and validation. For Software Components (SWCs) ARXML³ files describe their interfaces, and the interactions between them within the AUTOSAR architecture. We can extract from them runnables⁴, input and output interfaces, parameters and associated data types and subtypes.

First, we want to identify runnable entities which are functions that can be periodic or based on event as

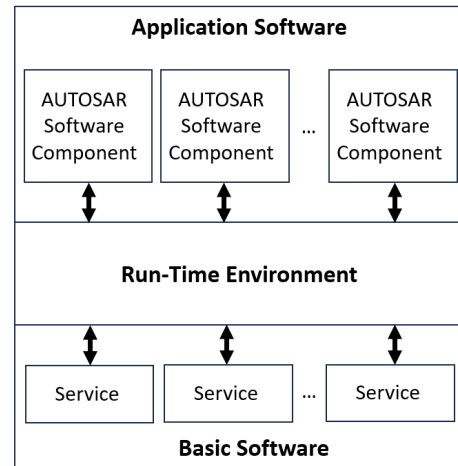


Fig. 4. AUTOSAR architecture

the contracts are defined for each function. Using an XPath notation, all the runnables are located in `"/AR-PACKAGES/AR-PACKAGE/ELEMENTS/APPLICATION-SW-COMPONENT-TYPE/INTERNAL-BEHAVIORS/SWC-INTERNAL-BEHAVIOR/RUNNABLES"`. The runnables are mapped to functions and their real name used by the code can be found in `"PARAMETER-ACCESS/SYMBOL"`.

Then, for each runnable we retrieve its interfaces (inputs and parameters) in `"RUNNABLES/RUNNABLE-ENTITY/DATA-READ-ACCESS"`. In `"RUNNABLES/RUNNABLE-ENTITY/DATA-READ-ACCESS/SHORT-NAME"`, we have the name of the variable present in the code. There are also other types of interfaces used less frequently that can be found inside the `"RUNNABLES/RUNNABLE-ENTITY/DATA-RECEIVE-POINT-BY-ARGUMENTS"` and `"RUNNABLES/RUNNABLE-ENTITY/DATA-RECEIVE-POINT-BY-VALUES"`. For the outputs, the interfaces are specified inside the `"RUNNABLES/RUNNABLE-ENTITY/DATA-SEND-POINTS"` and `"RUNNABLES/RUNNABLE-ENTITY/DATA-WRITE-ACCESS"` tags. To find the data type of an interface, we need to extract the last entity from the `"TARGET-DATA-PROTOTYPE-REF"` tag.

For each interface of each runnable we can find the types associated in the `"RootP_SR_Data"` package.

The types are defined in `"/AR-PACKAGES/AR-PACKAGE/ELEMENTS/IMPLEMENTATION-DATA-TYPE"` which maps each custom type to an AUTOSAR base type and also provide a range constraint. For example, we may have a Boolean type based on integer but constrained only between 0 and 1. The constraints can be found in `"/AR-PACKAGES/AR-PACKAGE/ELEMENTS/DATA-CONSTR"`.

For types containing enumerated values, we can find the definition of their values inside `"/AR-PACKAGES/AR-PACKAGE/ELEMENTS/COMPU-METHOD"`.

²ECU: Electronic Control Unit

³All examples are based on "SoftwareComponent Template – AUTOSAR Rel.4.2.2"

⁴Runnable: In the context of AUTOSAR, a "runnable" refers to a unit of executable code that performs a specific task or function.

D. Network Communication Signals

Vehicle communication between different ECUs is generally supported by a CAN⁵ network. Software components receiving or sending data through this network define a mapping of this data to signals which are grouped into messages. Each message is generally related to a specific function or subsystem within the vehicle and can be transmitted over multiple networks. Each signal has different properties. Here are the primary properties:

- **Data Length:** specifies the number of bits used to represent the data within the signal. Generally, we use the shortest length that can represent the data e.g. 1 bit for Boolean data.
- **Data Type:** CAN signals can represent various types of data, including integers (signed or unsigned), floating-point numbers, Boolean values, or enumerated types. The data type defines how the raw binary data within the signal is interpreted.
- **Data Min/Max:** specifies the range of values for the data.
- **Data Position:** specifies the byte number and bit number where the signal is located inside the message.
- **Data Unit:** Specifies the physical unit of the data.
- **Scaling Factor and Offset:** Signals may have scaling factors and offsets applied to convert raw data values into engineering units. These parameters define the relationship between the raw data values transmitted over the CAN bus and the corresponding physical quantities they represent.

Our idea is to deduce from the network signals the maximum and minimum value it can take e.g., a signal encoded in 3 signed bits will always be inside the range [-4,3]. Reading such a signal, we are sure that we will never get a value outside this interval. On the other hand, writing a data in such a signal, we must guarantee that the value produced by the software component will never be outside of the range which cannot be always true as in the C language it will be at least encoded in 8 bits.

E. Algorithm for precise contracts determination

In this section, we present an algorithm that takes as input the architecture specification of a given software component we want to verify. This specification contains the runnables and their types, their periodicity, their interfaces. For each interface we can have its range of possible values: min/max values may be specified, the type of the interface is always specified, a list of values may be specified, a mapping to a communication signal may be specified. We apply our algorithm to a given component on all runnables that are not used for initialization of the software component and for each of their interfaces we try to get the most precise and relevant interval of values. For example, if a data is encoded in 8 signed bits ranging [-128,127] in the code but is mapped on 3 signed bits ranging [-4,3] on the CAN bus we will consider the interval [-4,3] as the most precise range even if it is

encoded in *uint8* type. If the data type is an enumeration e.g. [1,3,5] and encoded in 3 unsigned bits we will consider the enumerated values as the most precise range. They represent the intersection of all specified values. After finding min/max values, we search each interface in the code and keep only interfaces present in the code. Sometimes the specification may have more interfaces e.g., when the designer decided to remove an interface from the code but not in the specification.

The results produced by our algorithm that find the best min/max value for each interface are stored in a JSON file. It contains all the interfaces (input, output, parameters) for each runnable/function that are used by the code.

The JSON file obtained after contract extraction is considered as an intermediate file which is used as entry data to create contracts in the format of the static analysis tool. It contains not only contracts information but all other necessary information that can permit analyze the software component especially header and source files. *Listing 1* provides an example of such intermediate JSON file.

```
{
  "module": "ASWC_M001",
  "proj_name": "Module ASWC_M001 analysis",
  "proj_description": "Project with contracts",
  "analysis-entry": "__main__",
  "base_dir": "C:/Project1",
  "language": "C",
  "files": ["Rte_Api.c", "Lib_Sim.c", "ASWC_M001.c"],
  "includes": ["Generated/Rte", "headers"],
  "defines": [],
  "init_runnables": ["RUNI_M001"],
  "periodic_runnables": [{
    "name": "RUNA_M001",
    "consume": [{
      "name": "VehicleSpeed",
      "min": 0,
      "max": 255,
      "values": []
    }],
    "produce": [{
      "name": "MIL_LED",
      "min": 0,
      "max": 1,
      "values": []
    }],
    "values": [
      60,
      105,
      150 ]],
  }],
  "name": "KeyState",
  "min": 60,
  "max": 150,
  "values": [
    60,
    105,
    150 ]],
  "name": "MIL_LED",
  "min": 0,
  "max": 1,
  "values": []
}]
```

Listing 1. Intermediate JSON file with contracts

V. INCREMENTAL CONTRACTS APPLICATION

Once the contracts are being extracted from the different specification sources, they can be applied in an incremental way depending on the objectives we want to achieve.

A. Contracts for inputs only

When running static analysis without providing any contract for the inputs, the analyzer will consider the entire range for each input variable thus significantly increasing the state

⁵CAN: Controller Area Network

space. When we provide contract for the inputs, we reduce the state space because only possible values for each input are considered.

B. Contracts for inputs and parameters

To further reduce the state space, we can also provide contracts for the parameters in addition to the contracts of the inputs. Each parameter can also have defined a minimum and maximum value. Parameters or calibrations are used to configure the software for a specific project. Providing a range of values for each parameter is useful for the verification of a software that will be used on multiple projects. We recommend starting by this method.

C. Contracts for inputs and specific parameter values

In addition to the previous methods, we can fix the values of the parameters thus reducing further the state space. Providing a fixed value for each parameter is useful for the verification of a software that will be used only on one project. It requires connection with the integration team to obtain these values.

D. Contracts for guaranteeing outputs

With the previous methods we only provided constraints for the input data to reduce the state space thus reducing the number of false alarms. With this method we provide contracts of type guarantee for the outputs and request the static analyzer to prove they are always valid. For example, if we specify a Boolean output but in the C code it is represented on 8 bits, we want to guarantee that it will only take two values: 0 and 1.

VI. EXPERIMENTS AND METHODOLOGY

Our experiment consisted of analyzing 222 application software modules used in a body ECU and one complex ADAS module containing generated code. We used the Astrée [26] static analysis tool version 23.04i in a batch mode to run the analysis and produce reports. For the body software modules, we developed a tool extracting automatically contracts from the Software Architecture Design Database also including network signals information and runnables. The information is stored in the JSON format mentioned above and is then used to setup automatically analysis projects and run them by batch. We extracted 6728 contracts for inputs, 8058 contracts for parameters and 6258 contracts for outputs. This JSON file can also be used as a source to configure other sound static analyzers such as Polyspace Code Prover.

For the ADAS module we used the System Design Interface Definitions and extracted them directly from an Excel table. We obtained 51 contracts for inputs and 104 contracts for parameters.

We have 3 different scenarios to measure the errors reported by the static analysis tool:

- Without contracts: we first analyze the modules without any notion of contract. In this case, the static analyzer considers the entire ranges of values for each input data.
- Contracts for inputs: we provide contracts for all inputs present in the code.

- Contracts for inputs and parameters: we provide contracts for all inputs and parameters present in the code. This scenario is applied to the 222 application modules.
- Contracts for inputs and specific parameter values: we provide contracts for all inputs and the parameters are tuned to a specific value valid for the project. We applied this scenario to the ADAS module.

VII. RESULTS

In this section, we comment on the results obtained for body and ADAS software modules.

A. Body software results

Historically, the body ECU software (implementing functions like Lighting, Fuel level, Lane keeping assist, Air conditioning, etc.) was developed using integer numbers and a fixed-point arithmetic where needed. The advantage of this method is that the storage types are optimized to the values encoded e.g., if we need to use the interval $[0,10]$ we declare an unsigned integer variable over 8 bits. Today, all modern microcontrollers integrate an FPU⁶ and software designers started to use floating-point arithmetic in their code. It is frequent that a 32 bits floating-point (simple precision) type that can represent more than 4 billion different numbers is used for variables that need only a short set of values e.g. $[0.0,10.0]$. If we do not provide the static analysis tool the specified by the designer interval, it will consider the entire range of 4 billion values for each variable or parameter which will result in a lot of false positive results.

We observe that our method to generate precise contracts for static analysis tools based on abstract interpretation provide better results for modules using floating-point arithmetic or large integers. Larger is the type used by the software designer for representing small ranges, better is the efficiency of the contract. As a consequence, for the body ECU software we reduced the false alarms by 41.59% when providing contracts for modules between 10,000 and 20,000 lines of code using floating-point arithmetic. For the other modules between 1,000 and 10,000 lines of code using integers, only 11.36% of the alarms were removed by adding the contracts. The explanation is that the most used type *uint8* has a rather small range of values. We also noted that the analysis time was not influenced by the contracts as it remained almost constant.

B. ADAS software results

The ADAS module we analyzed is rather complex (10,000 lines of code) and uses massively floating-point arithmetic. For this module after adding the contracts we removed 84.58% of the alarms. We think that this reduction is also the result of using fixed values for parameters instead of a range of values. It permitted us to identify some potential issues and fix them.

⁶A floating-point unit (FPU, or a math coprocessor) is a part of a computer system specially designed to carry out operations on floating-point numbers

VIII. CONCLUSIONS AND FUTURE WORK

The ISO 26262 safety standard recommends using abstract interpretation for static code analysis of ASIL level software. If the context of the software component is not provided to the tool, the results may contain a lot of false alarms especially for codes containing floating-point calculations.

In this paper, we made an overview of different design by contract techniques and provided a method to generate and use contracts of three types: inputs, parameters, and outputs. Contracts over the Inputs and Parameters provide constraints for the static analyzer to reduce the state space of the system. Contracts over the outputs are used to prove that no matter what happens, the outputs will always be produced in the specified range. We proposed an algorithm to obtain contracts automatically by selecting the most appropriate source for each of them. Our experiment showed a significant reduction of false alarms especially for large modules using floating-point arithmetic. Another application of design by contract is in the reuse of components as we may check if the interfaces are compatible (same contracts) or not. By using design by contract, we build reliable reusable components.

As a future work, we would like to investigate how a contract obtained from an external signal could be propagated for all components receiving a copy of the same data and check if the physical dimensions are propagated correctly.

REFERENCES

- [1] Dirk Slama, Achim Nonnenmacher, and Thomas Irawan, *The Software-Defined Vehicle*. O'Reilly Media, Inc., Sep. 2023. [Online]. Available: https://www.bosch-mobility.com/media/global/mobility-topics/mobility-topics/software-defined-vehicle/download/231019_sdv_ebook.pdf
- [2] ISO, *ISO 26262, Road vehicles - Functional safety*. ISO, Geneva, Switzerland, 2018, type: Norm.
- [3] V. Todorov, F. Boulanger, and S. Taha, "Formal Verification of Automotive Embedded Software," in *Proceedings of the 6th Conference on Formal Methods in Software Engineering*, ser. FormaliSE '18. New York, NY, USA: ACM, Nov. 2018, pp. 84–87. [Online]. Available: <https://doi.org/10.1145/3193992.3194003>
- [4] B. Meyer, "Applying "Design by Contract"," *Computer*, vol. 25, no. 10, pp. 40–51, Oct. 1992. [Online]. Available: <http://dx.doi.org/10.1109/2.161279>
- [5] B. Meyer, "Touch of Class: Learning to Program Well with Objects and Contracts", ser. 2009 Springer E-Books. Springer, 2009. [Online]. Available: <https://books.google.fr/books?id=6yupzA8JzjQC>
- [6] R. W. Floyd, "Assigning Meanings to Programs," *Proceedings of Symposium on Applied Mathematics*, vol. 19, pp. 19–32, 1967. [Online]. Available: <http://laser.cs.umass.edu/courses/cs521-621.Spr06/papers/Floyd.pdf>
- [7] C. A. R. Hoare, "An Axiomatic Basis for Computer Programming," *Commun. ACM*, vol. 12, no. 10, pp. 576–580, Oct. 1969. [Online]. Available: <http://doi.acm.org/10.1145/363235.363259>
- [8] E. W. Dijkstra, "Guarded Commands, Nondeterminacy and Formal Derivation of Programs," *Commun. ACM*, vol. 18, no. 8, pp. 453–457, Aug. 1975. [Online]. Available: <http://doi.acm.org/10.1145/360933.360975>
- [9] N. V. Shilov, I. S. Anureev, and E. V. Bodin, "Generation of correctness conditions for imperative programs," *Programming and Computer Software*, vol. 34, no. 6, pp. 307–321, Nov. 2008. [Online]. Available: <https://doi.org/10.1134/S0361768808060029>
- [10] M. Barnett and K. R. M. Leino, "Weakest-precondition of Unstructured Programs," *SIGSOFT Softw. Eng. Notes*, vol. 31, no. 1, pp. 82–87, Sep. 2005. [Online]. Available: <http://doi.acm.org/10.1145/1108768.1108813>
- [11] C. Flanagan, C. Flanagan, and J. B. Saxe, "Avoiding Exponential Explosion: Generating Compact Verification Conditions," *SIGPLAN Not.*, vol. 36, no. 3, pp. 193–205, Jan. 2001. [Online]. Available: <http://doi.acm.org/10.1145/373243.360220>
- [12] S. Conchon, A. Coquereau, M. Iguemlala, and A. Mebsout, "Alt-Ergo 2.2," in *SMT Workshop: International Workshop on Satisfiability Modulo Theories*, Oxford, United Kingdom, Jul. 2018. [Online]. Available: <https://hal.inria.fr/hal-01960203>
- [13] C. Barrett, C. L. Conway, M. Deters, L. Hadarean, D. Jovanović, T. King, A. Reynolds, and C. Tinelli, "CVC4," in *Proceedings of the 23rd International Conference on Computer Aided Verification (CAV '11)*, ser. Lecture Notes in Computer Science, G. Gopalakrishnan and S. Qadeer, Eds., vol. 6806. Springer, Jul. 2011, pp. 171–177. [Online]. Available: <http://www.cs.stanford.edu/barrett/pubs/BCD+11.pdf>
- [14] B. Dutertre, "Yices 2.2," in *Computer Aided Verification*, A. Biere and R. Bloem, Eds. Cham: Springer International Publishing, 2014, pp. 737–744.
- [15] L. De Moura and N. Bjørner, "Z3: An Efficient SMT Solver," in *Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, ser. TACAS'08/ETAPS'08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 337–340. [Online]. Available: https://www.researchgate.net/publication/225142568_Z3_an_efficient_SMT_solver
- [16] F. Kirchner, N. Kosmatov, V. Prevosto, J. Signoles, and B. Yakobowski, "Frama-C: A software analysis perspective," *Formal Aspects of Computing*, vol. 27, no. 3, pp. 573–609, 2015. [Online]. Available: <http://dx.doi.org/10.1007/s00165-014-0326-7>
- [17] V. Todorov, S. Taha, F. Boulanger, and A. Hernandez, "Proving Properties of Discrete-Valued Functions Using Deductive Proof: Application to the Square Root," *System Informatics*, no. 14, pp. 45–54, Jul. 2019. [Online]. Available: <https://system-informatics.ru/en/article/248>
- [18] "Building bug-free O-O software: An Introduction to Design by Contract," Tech. Rep., 2023. [Online]. Available: <https://www.eiffel.com/values/design-by-contract/introduction/>
- [19] R. Kramer, "iContract - The Java(tm) Design by Contract(tm) Tool," in *Proceedings of the Technology of Object-Oriented Languages and Systems*, ser. TOOLS '98. USA: IEEE Computer Society, 1998, p. 295.
- [20] A. Beugnard, J.-M. Jézéquel, N. Plouzeau, and D. Watkins, "Making components contract aware," *Computer*, vol. 32, no. 7, pp. 38–45, Jul. 1999, publisher: IEEE Computer Society. [Online]. Available: <https://imt-atlantique.hal.science/hal-01794333>
- [21] A. Benveniste, B. Caillaud, D. Nickovic, R. Passerone, J.-B. Raclat, P. Reinkemeier, A. Sangiovanni-Vincentelli, W. Damm, T. Henzinger, and K. G. Larsen, "Contracts for System Design," INRIA, Research Report RR-8147, Nov. 2012. [Online]. Available: <https://inria.hal.science/hal-00757488>
- [22] N. Halbwachs, F. Lagnier, and P. Raymond, "Synchronous observers and the verification of reactive systems," in *Third Int. Conf. on Algebraic Methodology and Software Technology, AMAST'93*, M. Nivat, C. Rattray, T. Rus, and G. Scollo, Eds. Twente: Workshops in Computing, Springer Verlag, Jun. 1993.
- [23] L. A. Clarke and D. S. Rosenblum, "A historical perspective on runtime assertion checking in software development," *SIGSOFT Softw. Eng. Notes*, vol. 31, no. 3, pp. 25–37, May 2006, place: New York, NY, USA Publisher: Association for Computing Machinery. [Online]. Available: <https://doi.org/10.1145/1127878.1127900>
- [24] N. Kosmatov and J. Signoles, "A Lesson on Runtime Assertion Checking with Frama-C," vol. 8174, Sep. 2013.
- [25] P. Cousot and R. Cousot, "Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints," in *Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, ser. POPL '77. New York, NY, USA: ACM, 1977, pp. 238–252. [Online]. Available: <http://doi.acm.org/10.1145/512950.512973>
- [26] P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival, "The ASTREE Analyzer," in *Programming Languages and Systems*, M. Sagiv, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 21–30.

Test Suite Augmentation using Language Models - Applying RAG to Improve Robustness Verification

Adam Mackay MEng (QA-Systems)

Abstract—This paper presents a novel approach to test suite augmentation using large language models and retrieval augmented generation (RAG) techniques. By leveraging the power of GPT-4 and a customised RAG framework, we show the effectiveness of automating test case generation to improve robustness verification in safety-critical embedded software systems. Our method involves indexing and retrieving relevant information from the entire software repository, including design documents, requirement specifications, prior test cases, and source code. The generated test cases are then integrated into the Cantata test framework, targeting key areas such as invalid values, overflow scenarios, timeout conditions, and improper state transitions. We applied our approach to two open-source projects, PX4 Autopilot and Apollo Auto, and achieved promising results in terms of increased test coverage, defect detection, and compliance with industry standards like DO-178C and ISO 26262. Compared to human-authored test suites, our method generated a larger volume of tests, exposed a broader array of test scenarios, and unveiled additional high-severity defects. The results validate the potential of advanced language models and RAG techniques in enhancing software reliability and safety through automated testing. Future work aims to refine these techniques further and expand their applications in the software development lifecycle.

I. INTRODUCTION

Robustness testing is crucial for safety-critical embedded software like aviation and medical systems, ensuring they manage errors gracefully and recover from unexpected situations effectively. This methodical approach validates key capabilities such as error detection, fault tolerance, and overload handling, elements required to align with recognised standards like DO-178C.

Manual test case authoring, particularly for requirements-based testing, demands considerable effort and is often fraught with risks of misinterpretations or oversight of corner cases. The evolving nature of software further complicates this, requiring continual test adaptation across versions and functionalities.

Retrieval augmented generation (RAG) offers a promising approach here. It combines language models with an information retrieval system, enhancing automatically generated content with contextually relevant data extracted from documentation and prior tests. This enables an adaptive response from the system, producing text or code that is both coherent and contextually informed. This approach is driven by requirements, ensuring generated tests trace directly to specific functional needs.

In harnessing the generative capabilities of GPT-4 and the assistance of automated agents, we employ RAG across the software repository, integrating code, existing tests, and documentation to autonomously generate test cases. These cases scrutinise the software's robustness across various conditions, such as invalid values, overflow scenarios, and

abnormal conditions, enabling a comprehensive inspection that mitigates the risks associated with manual testing. Additionally, a human-in-the-loop approach is employed, wherein generated tests are manually reviewed to further refine their effectiveness and efficiency, ensuring that the automation does not inadvertently overlook vulnerabilities.

This paper will present our methodologies, share the preliminary results from applying these methods to real-world embedded software projects, and discuss forthcoming work aimed at further refining and expanding these approaches. Our contributions extend beyond the application of RAG and language models to robustness testing; we also provide insights and data about their efficacy, offering a resourceful guide and a benchmark for future explorations and enhancements in automated testing in embedded software realms.

II. RELATED WORK

Our work builds upon the growing body of research on retrieval-augmented generation (RAG) and its applications in various domains. The seminal paper by Lewis et al. [1] introduced the concept of RAG, demonstrating its effectiveness in knowledge-intensive NLP tasks such as open-domain question answering. They showed that by augmenting language models with retrieved information, RAG could significantly improve performance compared to purely parametric approaches. Our work extends this idea to the domain of software testing, leveraging RAG to generate high-quality test cases.

GPT-4 and Large Language Models

We leveraged GPT-4, a distinguished multimodal language model developed by OpenAI, which excels in natural language generation, reasoning, and coding tasks. Renowned for its competence in single-shot and few-shot tasks, GPT-4, when coupled with refined prompting methods, emerges as an ideal model for benchmarking new applications of large language models (LLMs) in innovative manners. Consequently, GPT-4 was instrumental in automating the generation of our test cases, demonstrating its applicability and efficiency in novel use-cases.

Retrieval Augmented Generation (RAG)

The paper "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks" by Lewis et al. [1] provides an overview of the RAG process. As they explain, "RAG works by adding a step to this basic process. Namely, a retrieval step is performed where, based on the user's prompt, the relevant information is extracted from an external knowledge base and injected into the prompt before being passed to the LLM." They note the three key elements are Retrieval, Generation, and Augmentation. This aligns with the RAG framework we employ in our study.

Additionally, the survey paper "Retrieval-Augmented Generation for Large Language Models: A Survey" by Gao et al. [3] categorises RAG research into three types: Naïve

RAG, Advanced RAG, and Modular RAG. Our approach falls under the Advanced RAG paradigm, as we incorporate techniques like query rewriting and prompt summarisation to enhance the base RAG process.

During our research, we also explored the potential of Graph RAG, a recent advancement introduced by Microsoft Research [15]. Graph RAG aims to improve upon baseline RAG by utilising LLM-generated knowledge graphs to enhance retrieval performance, particularly in situations involving complex problems on private datasets. However, after careful consideration, we determined that Graph RAG was not well-suited to our specific use case, which primarily involves working with code and documentation fragments. The knowledge graph approach employed by Graph RAG did not align with the structure and relationships found in our software repository. As a result, we opted to develop our own custom RAG system that could better handle the unique characteristics of our dataset and generate more relevant test cases.

Recent advancements in RAG have focused on improving retrieval strategies and enhancing the interaction between the retriever and generator components. The survey by Gao et al. [3] provides a comprehensive overview of these developments, categorising RAG approaches into Naïve RAG, Advanced RAG, and Modular RAG. Our method aligns with the Advanced RAG paradigm, as we incorporate techniques such as query rewriting and prompt summarisation to optimise the retrieval process. This builds upon the work of Wang et al. [6] and Shao et al. [7], who showed the benefits of iterative retrieval and generation for knowledge-intensive tasks.

In the context of software engineering, there has been growing interest in leveraging large language models (LLMs) for various tasks, including code generation, bug detection, and test case generation. The work by Chen et al. [9] and Nijkamp et al. [10] showcased the potential of LLMs for generating high-quality code snippets and assisting developers in programming tasks. Our work extends this line of research by focusing specifically on test case generation, a critical aspect of software verification and validation.

While existing approaches to test case generation often rely on traditional techniques, such as symbolic execution, model-based testing, and search-based algorithms, our RAG-based approach offers a novel perspective. By leveraging the vast knowledge captured in LLMs and augmenting it with project-specific information, we can generate test cases that are more closely aligned with real-world requirements and specifications. This complements the work of Kang et al. [11] and Li et al. [12], who explored the integration of domain-specific knowledge into language models for improved performance in specialised tasks.

Furthermore, our human-in-the-loop approach sets our work apart from fully automated (algorithmic) test case generation methods. The iterative feedback process, involving experienced testers, allows for the refinement and optimisation of generated test cases based on domain expertise and project-specific considerations. This aligns with the findings of Trivedi et al. [13] and Guo et al. [14], who emphasised the importance of human involvement in guiding and validating the outputs of language models in complex tasks.

There have been noteworthy developments in the automation of unit test generation within integrated

development environments (IDEs) recently. For instance, Visual Studio Code (VSCode) offers a built-in solution for generating unit tests based on the code under development [22]. This feature leverages static analysis techniques and predefined templates to create test stubs and assertions, streamlining the process of writing unit tests. However, these IDE-based solutions often rely on heuristics and lack the deep understanding of project-specific requirements and domain knowledge that our RAG-based approach incorporates. By combining the power of large language models with retrieval augmentation and human expertise, our method goes beyond the capabilities of IDE-based test generation, enabling the creation of more comprehensive and context-aware test cases.

Several other tools and techniques have been developed for automatic test case generation across different programming languages. EvoSuite [23] is a search-based tool that generates unit tests for Java classes using evolutionary computation, aiming to maximize code coverage. Randoop [24] is another Java-based tool that generates tests by randomly selecting method sequences and constructing test cases based on the observed behaviour. In the .NET ecosystem, Pex [25] employs dynamic symbolic execution to generate test inputs that achieve high code coverage. For C and C++ programs, KLEE [26] is a symbolic execution engine that automatically generates test cases by exploring different program paths using constraint solving techniques.

Cantata AutoTest, [16] a feature of the commercial testing framework used in our case studies, utilises advanced code parsing powered by the EDG parser to generate test cases for C/C++ code. The EDG parser provides a deep understanding of the code structure and enables Cantata AutoTest to create more targeted and effective test cases. While these tools have proven effective in their respective domains, they often rely on predefined strategies and lack the adaptability and context-awareness that our RAG-based approach provides. In contrast, our approach leverages the power of large language models and retrieval augmentation to generate test cases that are more closely aligned with real-world requirements and project-specific contexts.

Existing Tools for Code Indexing and Retrieval

To effectively index and chunk the various artifacts, we leveraged open-source code indexing systems commonly used to help IDEs navigate code and documentation. Specifically, we employed ctags [19] and cscope [20] to build a comprehensive index of the codebase. ctags is a programming tool that generates an index file of names found in source and header files of various programming languages, aiding code comprehension. It can index functions, variables, class members, macros, and more, depending on the language. On the other hand, cscope is a text-based source browsing tool that allows programmers to search source code for symbols, definitions, functions, regular expressions, and more.

Emerging Large Language Models

GPT-4 has been a robust tool for our research. However, our next steps will leverage Llama 2 (and its variants), announced by Meta in partnership with Microsoft in July 2023. Llama 2 is an open source LLM with the benefit of running locally on secure systems and is available in sizes of

7, 13, and 70 billion parameters. Llama 2, while maintaining a similar architecture to its predecessor, is trained on 40% more data, offering enhancements in code generation and model training flexibility.

Overall, our research contributes to the growing body of literature on RAG and its applications in software engineering. By demonstrating the effectiveness of RAG in test case generation and highlighting the benefits of human-in-the-loop collaboration, we provide new insights and directions for leveraging LLMs in the verification and validation process. Our work also opens up avenues for future research on the integration of RAG with other software engineering tasks and the development of more advanced retrieval and generation strategies tailored to the unique challenges of the software development lifecycle.

III. TECHNIQUES

Introduction to the RAG implementation

In our project, we applied RAG across the entire software repository, encompassing design documents, requirement specifications, prior test cases, and source code. This comprehensive approach ensured that the generator was provided with a broad context, enhancing the relevance and accuracy of the generated test cases. To facilitate this process, we developed custom tools and integrated them with the Cantata test framework.

It is important to note that we developed our own custom RAG system specifically tailored for working with code and documentation fragments. While we investigated various existing RAG implementations, such as Graph RAG [15], we found that they were not well-suited to our specific use case. Our custom RAG system allowed us to effectively integrate and process the diverse range of artifacts found in the software repository, enabling the generation of highly relevant and context-aware test cases.

Indexing and retrieval system

To effectively index and retrieve relevant information from the software repository, we developed a custom indexing and retrieval system. This system leverages the capabilities of ctags [19], cscope [20], and CodeQuery [21]. ctags and cscope are well-established tools used by IDEs for navigating code and documentation. ctags generates an index file of names found in source and header files, while cscope allows programmers to search source code for symbols, definitions, functions, and more.

We integrated these tools to create a robust foundation for our RAG implementation. CodeQuery further enhances the indexing process by building upon the databases of ctags and cscope to create a SQLite database. This database can be efficiently queried to retrieve relevant code snippets and documentation based on specific criteria.

To complement these existing tools, we developed a custom suite of tools called Magrathea. The Magrathea suite includes functions for extracting relevant fragments of code, documentation, and comments, which are then used to populate the context window of the language model. Some of the key functions in the Magrathea suite are:

- *find_function_end*: Reads a file from a specified start line until the end of the function is found, extracting the complete function body.

- *find_preceding_comments*: Retrieves any comment block found directly before a specified function.
- *find_macros_for_function*: Identifies macros used within specific functions in a source file, along with their definitions if there are non-unique values.
- *find_unique_calls*: Finds unique function calls made from functions in a given source file.

By integrating ctags, cscope, CodeQuery, and the Magrathea suite, we created a comprehensive indexing and retrieval system that allowed us to effectively apply RAG across the entire software repository.

This approach ensured that the language model had access to a rich context window containing highly relevant code snippets, documentation, and comments, enabling the generation of more precise and contextually informed test cases.

The indexing process was performed as a pre-processing step, allowing for efficient retrieval of relevant information during the test generation phase. The SQLite database created by CodeQuery served as a central repository for querying and retrieving the necessary artifacts based on specific criteria, such as function names, symbols, or keywords. This optimised retrieval process contributed to the overall efficiency and effectiveness of our RAG implementation in the context of automated test case generation.

Test generation process

The test generation process follows a structured workflow (figure 1) that leverages the RAG implementation and human expertise. The key steps in this process are:

1. Querying the RAG API: We query the RAG API with a code snippet and an engineered prompt that specifically requests the generation of tests.
2. Retrieval of relevant documents: RAG retrieves relevant documents and prior tests from the customised corpus based on the query.
3. Generation of initial test cases: The language model (LLM) generates an initial set of raw test cases based on the retrieved context.
4. Iterative refinement through human review: The generated test cases undergo an iterative refinement process, where human testers review and provide feedback on the tests' relevance, uniqueness, and potential effectiveness.
5. The framework was then manually prompted for refinements including:
 - Rewriting tests for improved clarity
 - Eliminating redundant or duplicative tests
 - Adding preconditions for clarity
 - Correcting any functional inaccuracies

This review was crucial for boosting precision and overall quality.

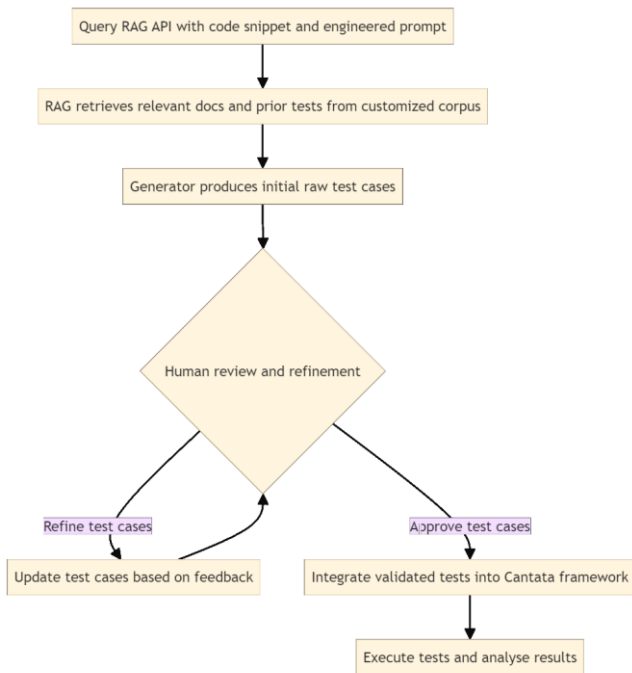


Figure 1 - Test generation workflow using Retrieval Augmented Generation (RAG) and human-in-the-loop review

This interaction shows the collaborative nature of the test generation workflow, involving the user, RAG, LLM, testing platform, and human examiner. Each component plays a crucial role in producing high-quality, context-aware test cases that thoroughly exercise the software under test.

Integration with Cantata

The generated test cases are integrated into the Cantata test framework [16], a commercial platform that provides a stable environment for test execution and result reporting. Cantata supports standard coverage metrics, such as statement coverage, decision coverage, and Modified Condition/Decision Coverage (MC/DC), which are essential for compliance with industry standards like DO-178C.

To enable the execution of robustness test cases, Cantata offers targeted error injection methods:

- Invalid values are introduced by leveraging white-box instrumentation to override variables with out-of-range inputs based on their data types and semantic context.
- Overflow conditions are induced by injecting custom code to provide inputs exceeding maximum buffer sizes defined in requirements specifications.
- Timeout testing is accomplished through code wrapping techniques that artificially increase processing time in the test harness beyond thresholds specified in design documents.

- Invalid state transitions are triggered by programmatically sequencing calls in an incorrect order relative to the expected state machine flow defined in the architecture.

Cantata's flexible error injection features, including white-box instrumentation, custom code injection, and code wrapping, enabled the wide array of automatically generated robustness test cases targeting key areas like invalid values, overflow, timeouts, and improper state transitions.

Experiment Setup

To evaluate the effectiveness of our RAG-based approach for test case generation, we conducted a series of experiments using a diverse set of embedded software projects. Our preliminary dataset consisted of two open-source projects: PX4 Autopilot [17] and Apollo Auto [18]. These projects were selected based on their complexity, availability of source code and documentation, and relevance to the domain of safety-critical embedded systems.

For each project, we applied our RAG-based approach to generate test cases targeting various aspects of robustness, such as invalid inputs, boundary conditions, and resource constraints. We evaluated the generated test cases using several metrics, including code coverage, defect detection rate, and compliance with industry standards like DO-178C and ISO 26262.

To assess the efficiency of our approach, we compared the time and effort required to generate test cases using our RAG-based method with the time and effort required for manual test case creation. We also conducted a qualitative analysis of the generated test cases, involving domain experts who reviewed the tests for relevance, effectiveness, and alignment with project requirements.

Our experiments were designed to answer the following research questions:

1. How does the RAG-based approach compare to manual test case creation in terms of coverage, defect detection, and compliance with industry standards?
2. What is the impact of incorporating human feedback in the test case generation process on the quality and effectiveness of the resulting tests?
3. How does the choice of LLM and RAG configuration affect the performance of the test case generation system?

To ensure the reliability and validity of our results, we conducted multiple runs of each experiment, using different prompting and cross-validation techniques where

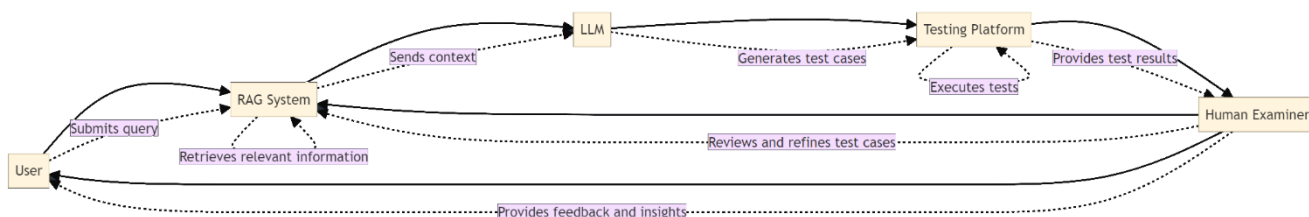


Figure 2 - Architectural overview of the interaction between the user, RAG system, LLM, testing platform, and human examiner in the test generation process.

applicable. We also documented the specific versions of the software tools, libraries, and models used in our experiments to facilitate reproducibility.

Optimisation and configuration

To optimise the RAG process for test case generation, we conducted several iterations, experimenting with various parameters and settings. One critical aspect was determining the optimal size of the context window, which determines the amount of relevant information provided to the model during the generation process. We tested context window sizes ranging from 256 to 1024 tokens and found that a size of 512 tokens struck a balance between providing sufficient context and maintaining computational efficiency.

We also explored different versions of GPT-4, including GPT-4-turbo, which offers faster generation speeds. The sampling temperature, which controls the randomness of the generated output, was varied between 0.5 and 1.0. The top_p parameter, also known as nucleus sampling, was set between 0.7 and 0.9, ensuring that the model considered only the most likely tokens during generation. Additionally, we adjusted the frequency_penalty and presence_penalty parameters to discourage the model from repeating the same output tokens too frequently.

Through these iterations, we identified the optimal configuration for our RAG implementation, which included a 512-token context window, GPT-4 with a sampling temperature of 0.7, top_p of 0.8, and frequency and presence penalties of 0.2.

Furthermore, we discovered that the precise wording of the system prompt played a crucial role in guiding the model to generate relevant and effective test cases. By carefully crafting the prompt to include specific instructions and requirements, we could steer the model towards generating tests that aligned with the project's goals.

Lastly, we employed a multi-shot RAG knowledge injection method, where the model was provided with multiple rounds of context and generated outputs. This approach allowed the model to progressively refine its understanding of the project and generate more targeted test cases based on the accumulated knowledge.

In this early proof-of-concept study, the primary focus was on assessing the code coverage achieved by the generated test cases. While selected cases, particularly those involving failure scenarios, were reviewed for accuracy, a more comprehensive examination of the test cases will be conducted in future stages of the research. The iterative refinement phase involving human examiners played a supportive role, with the team providing feedback on the generated tests based on their domain expertise. This feedback loop allowed for the identification of potential improvements and ensured that the generated tests aligned with the project's requirements. As this study aimed to establish the feasibility of the underlying method, a more rigorous and time-intensive review process will be incorporated in subsequent research phases to further validate the quality and effectiveness of the generated test cases.

We applied our technique to two different open source embedded C/C++ projects with the aim to validate to the stringent DO-178C Level A standards.

PX4 Autopilot (PX4)

PX4 [17] is an open-source flight control software for drones and other unmanned vehicles. It provides a flexible platform for users who need to control any kind of vehicle from a computing board. The platform contains everything needed for a full UAV system: flight stack, middleware to communicate between processors, and developer APIs to enhance existing functionalities. The combination of PX4's open-source nature, broad applicability, and mature, reliable ecosystem made it a compelling choice for applying this research.

- *Size and Test Generation:* A subset of the codebase, comprising around 35,000 Lines of Code (LOC), was subjected to our method, yielding 378 generated test cases. These tests emphasised robustness verification, exploring conditions like invalid values and overflow situations.
- *Comparison to Human-Authored Tests:* Compared to a suite of 265 tests written by developers, our method provided a significantly denser set of tests.
- *Code Coverage:* The tests achieved excellent code coverage, addressing 98% statement, 95% decision, and 90% MC/DC coverage, and were evaluated using Cantata. With further refinement of our methods we are confident of satisfying the DO-178C requirements for structural coverage analysis.
- *Defect Discovery:* Interestingly, these tests brought 14 new defects to light. Predominantly related to overflow issues, two defects impacted the function of calculating the optimal flight path and were categorised as medium severity according to code experts.

Apollo Auto (AA)

Apollo [18] is an open autonomous driving platform that aims to provide a comprehensive, safe, and reliable solution for autonomous driving. The project encompasses various technologies related to autonomous driving, such as perception, planning, control, and end-to-end deep learning, to enable vehicles to navigate in various environments and scenarios. The project was selected due to its open codebase, comprehensive documentation and robust coding practices.

- *Size and Test Generation:* On a larger section of this codebase of 50,000 LOC, our method generated 512 test cases, which also zeroed in on robustness aspects.
- *Comparison to Human-Authored Tests:* A traditional, developer-written test suite for Project B consisted of 342 tests, indicating a robust enhancement in test comprehensiveness through our technique.
- *Code Coverage:* A good level of code coverage was achieved, encapsulating 88% statement, 85% decision, and 80% MC/DC coverage, verified via Cantata. With further refinement of our methods we are confident of satisfying all the structural coverage requirements of ISO 26262
- *Defect Discovery:* Additionally, 9 new defects were discovered, mainly pertaining to validating invalid inputs, of which 3 were linked to the functionality of alerting the driver of potential hazards and were of high severity according to code experts.

Method/ Tool	Project	Statement Cov. (%)	Decision Cove (%)	MC/DC Cov. (%)	Test Failures	Defects Found
RAG-based Approach	PX4	98%	95%	90%	378	14
RAG-based Approach	AA	88%	85%	80%	512	9
Algorithmic Test generation	PX4	85%	80%	75%	0	0
Algorithmic Test generation	AA	80%	75%	70%	0	0
<u>Manual Testing</u>	<u>PX4</u>	<u>90%</u>	<u>85%</u>	<u>80%</u>	<u>265</u>	<u>0</u>
<u>Manual Testing</u>	<u>AA</u>	<u>85%</u>	<u>80%</u>	<u>75%</u>	<u>342</u>	<u>0</u>

Table 1 – Summary of initial results

To provide a clear overview of the performance of our RAG-based approach compared to other tools and manual testing, we present a summary of the results in Table 1. The table includes the statement coverage, decision coverage, MC/DC coverage, number of failure tests, and number of defects found for each method/tool and project. As clear from the table, our RAG-based approach consistently achieves higher coverage metrics and detects more defects compared to the other tools and manual testing across both projects.

Our RAG-based approach generated a diverse range of test cases targeting various aspects of robustness, with code coverage being a key driver in achieving a comprehensive set of tests. The generated tests encompassed scenarios such as:

Boundary value testing: Tests were generated to validate the system's behaviour when inputs approached or exceeded the defined limits, such as maximum and minimum values for numeric inputs or edge cases for string inputs. This included testing with values just inside and outside the valid ranges to ensure proper handling of boundary conditions.

Invalid input testing: Tests were created to assess the system's response to invalid, malformed, or unexpected input data, including null pointers, out-of-range values, and data types inconsistent with the expected format. These tests aimed to verify the system's ability to gracefully handle and recover from invalid inputs without crashing or exhibiting undefined behaviour.

Resource management testing: Tests were designed to evaluate the system's performance and stability under resource-constrained conditions, such as limited memory, high CPU utilisation, or low power scenarios. These tests helped identify potential memory leaks, resource exhaustion issues, and other performance bottlenecks.

Interrupt handling testing: Tests were generated to verify the correct handling of interrupts and the system's ability to prioritise and manage multiple interrupt sources. These tests ensured that the embedded software could respond to real-time events and maintain deterministic behaviour.

Error handling and recovery testing: Tests were generated to assess the system's ability to detect, isolate, and recover from errors, such as communication failures, sensor malfunctions, or unexpected resets. These tests verified the effectiveness of error handling mechanisms and the system's

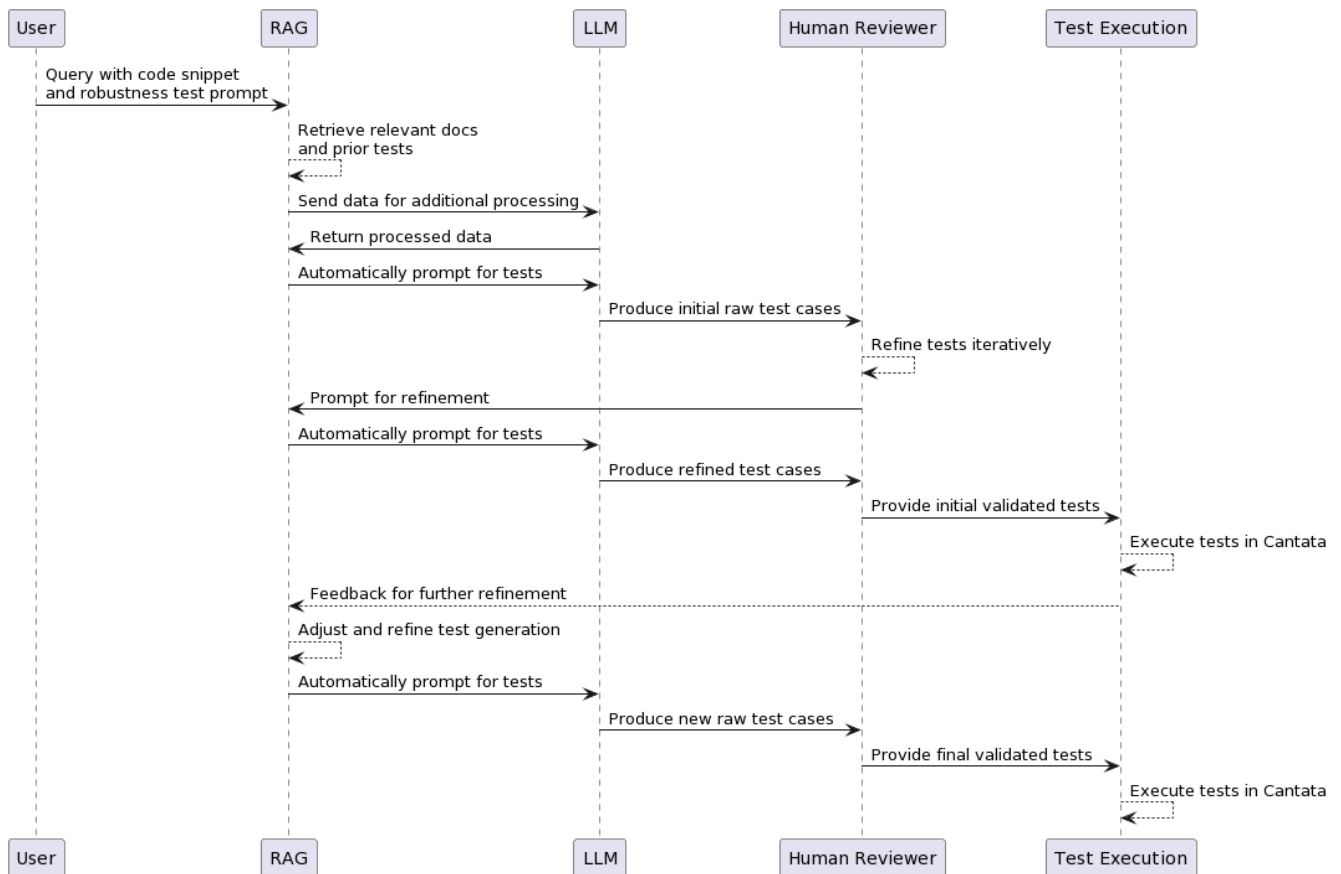


Figure 3 - The iterative process of RAG, LLM and review

capacity to maintain a safe state or initiate appropriate recovery procedures.

Included in the Annex are sample prompts used to generate various types of test.

Code comments and documentation fragments played a crucial role in informing the scope of the generated tests. By supplementing the code with context-relevant requirements and the developers' intentions, parsed from comments, our approach unlocked a performant set of tests. This contextual information guided the generation of test cases that aligned with the specific functionality and constraints of each code module, resulting in more targeted and effective testing.

The generated tests were executed dynamically, allowing for the validation of the system's behaviour under various conditions and the identification of any defects or vulnerabilities that could compromise the reliability and stability of the embedded software. By leveraging code coverage metrics, we ensured that the generated tests thoroughly exercised the codebase, reaching critical paths and uncovering potential issues that might have been missed by manual test creation efforts.

The human effort involved in generating additional robustness tests was significantly reduced compared to manual test creation. The RAG-based approach autonomously generated a comprehensive set of initial test cases, which were then reviewed and refined by human testers.

This review process (figure 3) typically involved:

1. Assessing the relevance and coverage of the generated tests
2. Identifying any missing critical scenarios or edge cases
3. Providing domain-specific insights and context to enhance the tests
4. Optimising the test data and expected outcomes
5. Ensuring alignment with project requirements and quality standards

On average, the human effort required for reviewing and refining the generated tests was approximately 30% of the time that would have been needed to create the tests from scratch manually. This highlights the efficiency gains achieved through the RAG-based approach.

By leveraging RAG to generate a wide range of robust test cases and incorporating human expertise to refine them further, our approach effectively identified defects and coding oversights that were previously undetected. This highlights the value of combining generative AI techniques with human domain knowledge to enhance software quality and reliability.

Comparison with Other Validation Tools

Compared to traditional validation tools, our RAG-based approach offers several distinct advantages. One key strength lies in its adaptability and scalability across diverse software projects. While many existing tools are tailored to specific programming languages, frameworks, or domains, our method leverages the language-agnostic nature of

LLMs. By training on a vast corpus of natural language and code, LLMs can effectively generate test cases for a wide range of software systems, regardless of their underlying technologies. This flexibility enables seamless integration into various development workflows and reduces the need for specialised validation tools for each project.

Another significant advantage of our approach is its ability to generate test cases that are grounded in real-world requirements and specifications. Traditional validation tools often rely on predefined templates, heuristics, or static analysis techniques, which may not fully capture the nuances and complexities of a given software project. In contrast, our RAG framework actively retrieves and incorporates relevant information from project documentation, user stories, and domain-specific guidelines. By combining this contextual knowledge with the generative capabilities of LLMs, we can produce test cases that are more closely aligned with the intended functionality and user expectations.

Furthermore, our RAG approach excels in generating test cases for complex, multi-component systems. Many existing validation tools struggle to handle the intricacies of interconnected modules, asynchronous operations, and distributed architectures. However, the retrieval component of our framework enables the LLM to gather and synthesise information from multiple sources, allowing it to generate test cases that span across system boundaries. This holistic perspective is particularly valuable in identifying integration points, data flow issues, and potential performance bottlenecks.

Lastly, the human-in-the-loop aspect of our approach sets it apart from fully automated validation tools. While automation is crucial for efficiency, the involvement of human experts adds a layer of qualitative assessment and domain-specific insights. The iterative feedback process allows for the refinement of test cases based on the unique requirements and priorities of each software project. This collaborative approach ensures that the generated test cases are not only technically sound but also aligned with the broader goals and constraints of the development team.

It is important to note that many of the other validation tools discussed in the "Related Work" section, such as EvoSuite, Randoop, Pex, and KLEE, are algorithmic in nature and focus on specific programming languages. While their approaches to test case generation can be compared and contrasted with our RAG-based method, these tools could not be directly applied to the case study projects, which primarily involve embedded C/C++ code.

The only tool that could be directly applied to the case study projects was the existing algorithmic AutoTest generation feature of Cantata, the commercial test framework used in our experiments. Cantata's AutoTest feature employs a combination of static analysis and path analysis to generate test cases based on the structure of the code under test.

In summary, our RAG-based approach offers several advantages over traditional validation tools:

- Adaptability and scalability across diverse software projects, regardless of the programming language or domain

- Ability to generate test cases grounded in real-world requirements and specifications by leveraging the knowledge captured in LLMs and project-specific documentation
- Effectiveness in generating test cases for complex, multi-component systems by synthesising information from multiple sources and considering system-wide interactions
- The value of human-in-the-loop collaboration in refining test cases, ensuring their alignment with project goals and constraints

While algorithmic tools like Cantata's AutoTest feature can be useful in certain scenarios, our RAG-based approach provides a comprehensive and flexible solution for test case generation in the context of safety-critical embedded systems.

V. CONCLUSIONS

Observations and Achievements

- *Increased Test Generation:* The approach not only produced a larger volume of tests compared to human-authored ones but also exposed a broader array of test scenarios.

- *Compliance with Stringent Standards:* The automated generation of tests did not compromise the ability to achieve high code coverage, ensuring adherence to vital industry standards like DO-178C and ISO 26262

- *Unveiling of Hidden Defects:* Notably, our approach unveiled additional, including high-severity, defects, which eluded detection by human-authored test suites.

Future Pathways

- *Exploring Llama 2 and other emerging LLMs:* While GPT-4 has been instrumental in our research, we plan to explore the potential of other emerging LLMs, such as Llama 2, for test case generation. As discussed in the "Related Work" section, Llama 2 offers several advantages, including the ability to run locally on secure systems and its enhanced training data. By leveraging these new LLMs, we aim to further improve the efficiency and effectiveness of our RAG-based approach.

- *Optimising Llama 2 for Embedded Projects:* This involves calibrating Llama 2 with various software documentation, adapting the model to the domain-specific language and patterns of embedded systems, and enhancing its performance with embedded system code.

- *Broadening RAG Applications:* Our future efforts will look towards employing RAG earlier in the software development process. This involves deriving test cases directly from requirements, co-developing test cases and code, and establishing a feedback mechanism beneficial for system designers.

Our preliminary results validate this approach and instil confidence in utilising advanced language models for robustness testing in safety-critical domains. The next phases of this research will further refine techniques and expand test generation capabilities to bolster automated testing for safety-critical systems. This ongoing pursuit aims to enhance software reliability and safety where flawless operation is indispensable. By integrating generative models with retrieval augmentation and rigorous human review, a pathway emerges to achieve robust and nuanced automated

testing. Through iterative refinement of these synergistic techniques, the journey towards more reliable, resilient, and efficient testing methodologies continue.

ACKNOWLEDGMENTS

I would like to express my gratitude to QA Systems for supporting this research as part of an internal project aimed at exploring future pathways for their commercial testing product, Cantata. This work would not have been possible without their commitment to pushing the boundaries of software testing and their dedication to enhancing the reliability and quality of safety-critical systems.

I acknowledge that certain commercially sensitive details regarding the implementation of this research have been omitted from this paper to protect QA Systems' intellectual property. I respect their need for confidentiality and appreciate their trust in allowing me to share the broader insights and methodologies developed during this project.

I am grateful for the collaboration and support provided by the QA Systems team throughout this research. Their expertise in software testing, particularly in the domain of safety-critical embedded systems, has been invaluable in shaping my approach and refining my techniques. I look forward to continued collaboration with QA Systems as I further develop and apply these advanced testing methodologies to real-world projects.

REFERENCES

- [1] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, Douwe Kiela. *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*, 12 Apr 2021
- [2] Jiawei Chen, Hongyu Lin, Xianpei Han, Le Sun. *Benchmarking Large Language Models in Retrieval-Augmented Generation*, 20 Dec 2023
- [3] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Qianyu Guo, Meng Wang, Haofen Wang. *Retrieval-Augmented Generation for Large Language Models: A Survey*, 5 Jan 2024
- [4] Yu He Ke, Liyuan Jin, Kabilan Elangovan, Hairil Rizal Abdullah, Nan Liu, Alex Tiong Heng Sia, Chai Rick Soh, Joshua Yi Min Tung, Jasmine Chiat Ling Ong, Daniel Shu Wei Ting. *Development and Testing of Retrieval Augmented Generation in Large Language Models - A Case Study Report*. 29 Jan 2024
- [5] Shervin Minaee, Tomas Mikolov, Narjes Nikzad, Meysam Chenaghlu, Richard Socher, Xavier Amatriain, Jianfeng Gao. *Large Language Models: A Survey*. 20 Feb 2024
- [6] Xintao Wang, Qianwen Yang, Yongting Qiu, Jiaqing Liang, Qianyu He, Zhouhong Gu, Yanghua Xiao, and Wei Wang. *Knowledgpt: Enhancing large language models with retrieval and storage access on knowledge bases*. 17 Aug 2023
- [7] Zhihong Shao, Yeyun Gong, Yelong Shen, Minlie Huang, Nan Duan, and Weizhu Chen. *Enhancing retrieval-augmented large language models with iterative retrieval-generation synergy*. 23 Oct 2023

- [8] Ran Elgedawy, John Sadik, Senjuti Dutta, Anuj Gautam, Konstantinos Georgiou, Farzin Gholamrezae, Fujiao Ji, Kyungchan Lim, Qian Liu, Scott Ruoti. *Ocasionally Secure: A Comparative Analysis of Code Generation Assistants*. 1 Feb 2024
- [9] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, *Evaluating large language models trained on code*. 14 Jul 2021
- [10] Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, Caiming Xiong. *CodeGen: An Open Large Language Model for Code with Multi-Turn Program Synthesis*. 27 Feb 2023
- [11] Minki Kang, Jin Myung Kwak, Jinheon Baek, and Sung Ju Hwang. *Knowledge graph-augmented language models for knowledge-grounded dialogue generation*. 30 May 2023
- [12] Xingxuan Li, Ruochen Zhao, Yew Ken Chia, Bosheng Ding, Lidong Bing, Shafiq Joty, and Soujanya Poria. *Chain of knowledge: A framework for grounding large language models with structured knowledge bases*. 21 Feb 2024
- [13] Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. *Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions*. 23 Jun 2023
- [14] Zhicheng Guo, Sijie Cheng, Yile Wang, Peng Li, Yang Liu. *Prompt-guided retrieval augmentation for non-knowledge-intensive tasks*. 28 May 2023
- [15] Jonathan Larson, Steven Truitt, *GraphRAG: Unlocking LLM discovery on narrative private data* 13 Feb 2024
- [16] Cantata, QA-Systems
<https://www.qa-systems.com/tools/cantata/>
- [17] PX4 Autopilot
<https://px4.io/>
- [18] Apollo Auto
<https://www.apollo.auto/>
- [19] Universal Ctags
<https://github.com/universal-ctags/ctags>
- [20] Cscope
<https://cscope.sourceforge.net/>
- [21] CodeQuery
<https://github.com/ruben2020/codequery>
- [22] Visual Studio Code. Generate Unit Tests for your code.
https://code.visualstudio.com/docs/python/testing#_generating-tests
- [23] Gordon Fraser and Andrea Arcuri. *EvoSuite: Automatic Test Suite Generation for Object-Oriented Software*. Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering, 2011.
- [24] Carlos Pacheco and Michael D. Ernst. *Randoop: Feedback-Directed Random Testing for Java*. Companion to the 22nd ACM SIGPLAN Conference on Object-Oriented Programming Systems and Applications Companion, 2007.
- [25] Nikolai Tillmann and Jonathan de Halleux. *Pex-White Box Test Generation for .NET*. Tests and Proofs, 2008.
- [26] Cristian Cadar, Daniel Dunbar, and Dawson Engler. *KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs*. Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation, 2008.

Prompting – example flow

This JSON-formatted example of a RAG interaction demonstrates how the user query combined with retrieved documents drives the LLM's generation of test cases. This illustrates the interplay between the user, RAG system, LLM, and human examiner in an iterative process to refine the test cases, as outlined in our test generation workflow.

```
{
  "prompt": {
    "model": "gpt-4",
    "messages": [
      {
        "role": "system",
        "content": "You are an expert C and C++ unit test writer. Write tests to exercise boundary values, equivalence classes, every statement, every decision true and false. Write tests that call every boolean condition both true and false. Write suitable robustness tests. Return valid json."},
      {
        "role": "user",
        "content": "We are only using the Cantata test harness to perform checks on the return values when calling the software under test."
      }
    ]
  }
}
```

Along with the name and description of the test, we require a block of code achieving the following:

- setting up any input data required for the test
- calling the entry point to the test
- if the function returns, set a variable to hold the return value
- performing a Cantata CHECK on any return

We also require a string of expected calls to onward functions. This string needs to be a comma separated list exactly in the order the calls are made and of the format:

```
<function_name_1>,<function_name_2>,
```

etc. Where <function_name> is the name of the function that we expect to be called during the test. We could be expecting one or more calls to:

```
low_pressure_check
get_systolic
get_diastolic
high_pressure_check
sound_alarm
```

```
"
  },
  {
    "role": "assistant",
    "content": "I understand. Please provide the code to generate tests for."},
  {
    "role": "user",
    "content": "Here is the code to test:
<truncated>},
  {
    "role": "assistant",
    "content": "Thanks for this piece of code, I will generate some small unit tests for this code. Please specify the entry point function to drive our first batch of tests."},
  {
    "role": "user",
    "content": "Our entry point is:
pressure_check"},
  {
    "role": "assistant",
    "content": "OK."},
  {
    "role": "user",
    "content": "Provide a list of function calls with expected returns and expected calls."}
  ],
  "functions": [
    {
      "name": "test_it",
      "description": "input to drive the creation of unit tests",
      "parameters": "<truncated>"
    }
  ],
  "function_call": {
```

```

        "name": "test_it"
    },
    "temperature": 1,
    "max_tokens": 2000,
    "top_p": 1,
    "frequency_penalty": 0,
    "presence_penalty": 0,
    "stream": false
  },
  "response": {
    "id": "xxx",
    "choices": [
      {
        "finish_reason": "stop",
        "index": 0,
        "logprobs": null,
        "message": {
          "content": null,
          "role": "assistant",
          "function_call": {
            "arguments": "{
"tests": [
  {
    "test_name":
"Test_pressure_check_with_monitor_OK",
    "test_description": "Test the pressure_check
function when monitorStatus is MONITOR_OK",
    "test_code": "
#define MONITOR_OK 1
extern int monitorStatus;
extern int systolic;
extern int diastolic;

void Test_pressure_check_with_monitor_OK()
{
  int retVal;

  // setup
  monitorStatus = MONITOR_OK;

  // exercise
  retVal = pressure_check();

  // verify
  CHECK_INT(retVal, 1);
  CHECK_INT(systolic, 120); // The expected return
from get_systolic
  CHECK_INT(diastolic, 80); // The expected return
from get_diastolic
}"",
      "call_list": "get_systolic,get_diastolic"
    },
    {
      "test_name":
"Test_pressure_check_with_monitor_NOT_OK",
      "test_description": "Test the pressure_check
function when monitorStatus is NOT MONITOR_OK",
      "test_code": "
#define MONITOR_OK 1
extern int monitorStatus;
extern int systolic;
extern int diastolic;

void Test_pressure_check_with_monitor_NOT_OK()
{
  int retVal;

  // setup
  monitorStatus = !MONITOR_OK;

  // exercise
  retVal = pressure_check();

  // verify
  CHECK_INT(retVal, 0);
  CHECK_INT(systolic, -1);
  CHECK_INT(diastolic, -1);
}"",
      "call_list": ""
    }
  ]
}

```

```

  ],
  "created": xxx,
  "model": "gpt-4-0613",
  "object": "chat.completion",
  "system_fingerprint": null,
  "usage": {
    "completion_tokens": 330,
    "prompt_tokens": 684,
    "total_tokens": 1014
  }
}

```

In this example the part **underlined and bold** is the key element to direct the system to generate tests of a specific type. Here we are looking for **boundary and robustness tests**.

Expanded Prompt Examples

"content": "You are an expert C and C++ unit test writer. Write tests to exercise boundary values, equivalence classes, every statement, every decision true and false. Write tests that call every boolean condition both true and false. Write suitable robustness tests. Return valid json. This includes creating scenarios where edge cases are tested, such as the smallest or largest possible values, values on either side of a boundary, and common robustness challenges in system behavior."

In this example we instruct the LLM to focus on generating specific types of unit tests, namely boundary value tests, equivalence class tests, and robustness tests for C/C++ code. These tests are crucial for ensuring that the software behaves as expected under various edge conditions and stress scenarios.

This approach emphasizes comprehensive testing by ensuring all paths and conditions are evaluated, enhancing the software's reliability and performance.

Variations for Different Testing Scenarios

1. Invalid Input Testing

"content": "As a specialist in C and C++ testing, generate unit tests that deliberately use invalid, unexpected, or outlier inputs. Verify that the software gracefully handles these inputs by either rejecting them with appropriate error messages or by ensuring no system instability. Return valid json."

This prompt directs the LLM to create unit tests that focus on how the application handles inputs that are incorrect or out of the expected range, which is essential for validating input validation and error management in the software.

2. Resource Management Testing

"content": "Create unit tests for C and C++ applications that assess the software's management of

system resources. These tests should include scenarios where resources are limited, such as low memory availability, and ensure the application releases all resources after use. Return valid json."

This prompt focuses on testing the software's ability to handle and efficiently manage system resources, crucial for applications that operate in resource-constrained environments or require high reliability.

3. Interrupt Handling Testing

"content": "Develop unit tests that simulate various interrupt conditions in C and C++ programs. Ensure that the software correctly handles and recovers from these interrupts, maintaining data integrity and proper operational state. Return valid json."

The tests generated from this prompt should ensure that the software can handle unexpected interruptions, maintaining stability and data integrity, which is vital for real-time and critical systems.

4. Error Handling and Recovery Testing

"content": "Generate unit tests for error handling and recovery mechanisms in C and C++ code. The tests should force the software into error states and evaluate the recovery procedures to return to a normal operating state. Return valid json."

This prompt instructs the LLM to create tests that assess the effectiveness of the software's error handling and recovery processes, critical for applications where uptime and reliability are essential.

Mixing tests and mathematical analysis: A launcher use case

Authors: David LESENS, Mathilde DUCAMP, Philippe RANOARIVONY (ArianeGroup)
<first name>.<last name>@ariane.group

1 Abstract

This paper shows on a case study (launcher sequence) how simple semi-formal methods and tools (SysML modelling, Domaine Specific Language, Simplex algorithm) can be used to improve the development and the validation of industrial systems without using complex formal methods which are sometimes difficult to manage by engineers.

Keywords: Model-based system and safety engineering, Requirement engineering, Verification methods, Formal method, Automatic test's generation

2 Introduction

This paper addresses the development of space launchers, for which a failure may have major financial consequences.

Using formal methods (e.g. [1]) to support the verification and validation of such critical real-time systems is often recommended because classic tests cannot be exhaustive (the system is demonstrated to be correct only on the used scenarios). Formal methods provide power-full tools to demonstrate that a system is bug-free on all the possible scenarios. On the other hand, formally proving timed properties on a system may also be quite difficult for non-experts.

This paper shows on a case study how informal requirements captured by simple tools directly usable by system engineers (such as Excel) or readable by system engineers (such as SysML [5] state-charts or activity diagrams) can be safely translated in more formalised descriptions.

Then it presents a method mixing testing and mathematical analysis (a classic simplex algorithm) allowing verifying system requirements and their implementation in the embedded software in the particular case of the design of launcher sequences.

After this introduction, the section 3 introduces the notion of launcher sequences, subject of the case study, and the process to develop them. The section 4 presents how launcher sequence's requirements can be translated in mathematical equations solvable by a classical simplex algorithm. The corresponding process is detailed section 5. This approach is demonstrated on a toy example in section 6. Finally, the section 7 compares this approach with other experiments performed by ArianeGroup with formal methods.

3 Introduction to launcher sequences

3.1 Objectives

The design of a launcher is highly complex and we will not present this in the paper. Instead, we focus on a sub-part of the design that concerns the launcher sequences. A launcher sequence describes all the acyclic events occurring during a launcher mission (ignition and stop of engines, stage release, payload services, etc.). Such launcher sequences shall satisfy requirements coming from several sources:

- Analysis of launcher flight phases: The definition of the launcher high-level behaviour specifies the flight phases, such as ground phase, atmospheric / extra-atmospheric phases, propulsive / orbital phases...
- Launcher versatility: The launcher shall be able to fulfil several kinds of mission (with different booster configurations, targeting different orbits...). Each mission corresponds to a specific launcher sequence
- Subsystems: Each subsystem (engine control, flight control, tank management, etc.) has its own needs toward the launcher sequences (e.g. duration of an equipment warmup, maximal frequency for commanding a piece of equipment, etc.).
- Transverse requirements affecting the relationships between the subsystems: For instance, correct pressurisation of the tanks needed for an engine ignition, navigation data needed by the guidance, etc.

The design of launcher sequences requires a synthesis of all these needs.

3.2 Process overview

The following figure summarizes the low part of a quite classical development process (this process is detailed section 5) based on a V development cycle (see [9]). The red circle highlights the topics addressed by this paper.

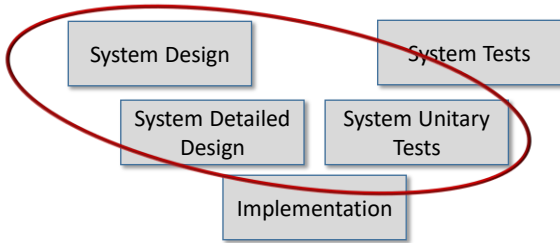


Figure 1: Simplified development process

- The system design defines the subsystems and the high levels interactions between them. It defines in particular the launcher sequences.
- The system detailed design provides all the details allowing the actual development of the subsystems and of the flight software. It details in particular the content of the launcher sequences.
- The system unitary tests allow testing each subsystem, in particular the launcher sequences.
- The system qualification tests the system as a whole (including all the subsystems and the launcher sequences).

3.3 Design of the launcher sequences

A space launcher is designed to be versatile, i.e. to be able to execute several kinds of mission (different trajectories, different number of boosts, different number of payloads, etc.). As a result, each mission corresponding to a specific launcher sequence. The flight software implements the launcher sequences and selects the sequence to be executed depending on the values of mission data.

Let us take the following very simple example. The launcher is supposed to be able to execute two kinds of sequences which are selected through the mission data "Option" (which can take the values "O1" or "O2"; "O1" may for instance require a sub-sequence with a boost phase, and "O2" a sub-sequence with a ballistic phase). These two possible sequences are designed through a SysML activity diagram.

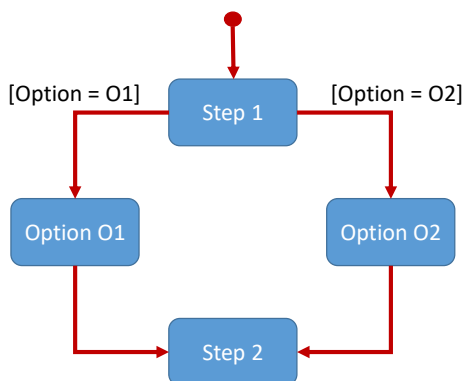


Figure 2: Design of the launcher sequences

"Step 1", "Option O1", "Option O2" and "Step 2" correspond to sub-sequences of the launcher sequences. The Figure 2 represents the two following launcher sequences:

- [Step 1; Option1; Step2] when Option = O1
- [Step 1; Option2; Step2] when Option = O2

3.4 Detailed design of the launcher sequences

The detailed design of a sub-sequence specifies the events which may occurred during this sub-sequence and their times of occurrence

The system engineers design the sub-sequences in an Excel file. An example of such design is given below (with only one event in each sub-sequence).

Sub-Seq	Time	Events	Properties
Step 1	T1 + 1mn	Ev1	
O1	T2 - 2mn	Ev2	
O2	T2	Ev1	
Step 2	T2 + M1	Ev2	Ev2 <= Ev1 + M2

Figure 3: Example of sub-sequence's definition

The "Step 1" sub-sequence specifies for instance that at the time "T1 + 1mn", the "Ev1" event shall occur.

An event may be an internal event decided by the launcher sequence (e.g. a command, such as the control of an engine) or an external event observed by the launcher sequence (e.g. the end of the warmup of a piece of equipment).

The column "properties" provides additional properties the launcher sequence shall satisfy. The "Step 2" sub-sequence specifies for instance that the "Ev2" event shall occur less than M2 minutes after the "Ev1" event (e.g. an engine shall be used less than 3 minutes after its warmup).

"T1" and "T2" are the times of occurrence of key-events decided by the launcher guidance (such as the time to ignite an engine or to release the fairing). Their values are computed during the flight and may vary depending on the launcher behaviour (e.g. in case of an over or under propulsive engine, in case of wind, etc.). The times of the other events (e.g. "Ev1" or "Ev2") are defined relatively to the times of these key-events.

"M1" and "M2" are mission data allowing customising the flight software for a particular mission.

3.5 Implementation of the launcher sequences

ArianeGroup has put in place a MBSE (Model Based System Engineering) method based on SysML [5] (customised and completed by a dedicated formalised profile) and on Domain Specific Languages (DSL) to describe the detailed functional design of the launcher system (see [6]). For instance:

- Use of a set of state-charts to describe the system and subsystem's high level behaviours,

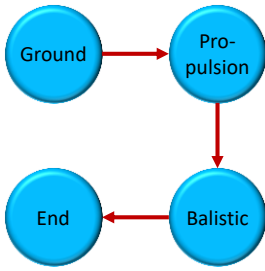


Figure 4: System state-chart

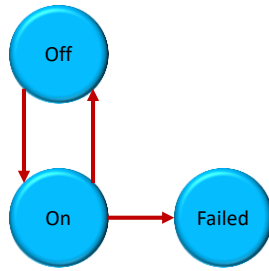


Figure 5: State-chart of a piece of equipment

- Use of block diagrams to describe the exchanges of information between the subsystems.

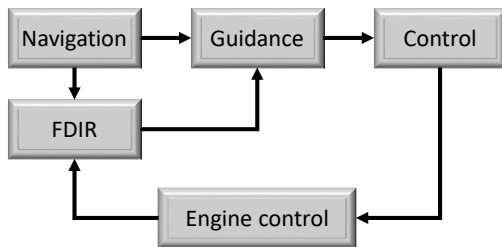


Figure 6: Main block diagram

- Use of a textual Domain Specific Language (DSL) to describe the launcher sequences and the FDIR (Fault Detection Isolation and recovery),

```

if Failure_Detected then
  goto Abort_Flight;
at T1: Cmd_Lift_Off;

```

Figure 7: DSL describing the mission

- Etc.

An event (see section 3.4) may typically be a guard of a state-chart (internal event of a launcher sequence, i.e. the launcher sequence requests the triggering of a transition of a state-chart) or an output of a sub-system (external event of a launcher sequence, such as the detection of an empty tank).

Automatic code generation is then used to generate a part of the flight software from these descriptions.

4 Formal specification and analysis

4.1 Formalisation of the design

This section shows how the informal description presented section 3 can be formalised to allow automatic analysis (see section 4.2).

The description of Figure 3 corresponds to the way the system engineers wish to describe the launcher sequences. As is, this description may be ambiguous and prevent an automated analysis. For instance:

- What is the meaning of “ $Ev2 \leq Ev1 + M2$ ”, considering that “ $Ev1$ ” and “ $Ev2$ ” are events (and that we cannot compare events), and that they may occur several times?
- How can we ensure that an input event (by definition not mastered by the launcher system) occurs at a given time?
- Is there an order of occurrence specified by the table of the previous section?
- Etc.

The system engineers and the method engineers have thus discussed to define a formal syntax and a formal semantics for both the SysML activity diagrams and the content of the Excel sheets in order to allow automatic analysis. The syntax has been defined to be as close as possible from the informal description used by the system engineers in order to be accepted by them, and the semantics as intuitive as possible.

The sub-sequence (first column of the previous table) specifies in which cases the line is applicable. In Figure 3, depending of the value of the enumerated mission data “Option”, the sequence may be:

- If Option = O1

Time	Events	Properties
T1 + 1mn	Ev1	
T2 - 2mn	Ev2	
T2 + M1	Ev2	$Ev2 \leq Ev1 + M2$

Figure 8: Instantiation of the sequence for option O1

- If Option = O2

Time	Events	Properties
T1 + 1mn	Ev1	
T2	Ev1	
T2 + M1	Ev2	$Ev2 \leq Ev1 + M2$

Figure 9: Instantiation of the sequence for option O2

The name of an event used in a property (such as “ $Ev2 \leq Ev1 + M2$ ”) shall be interpreted as the time of occurrence of the event (i.e. the property shall be interpreted as “ $time(Ev2) \leq time(Ev1) + M2$ ”).

When an event occurs several times, such event in a property shall be interpreted as the last occurrence of this event in the past, starting from the line on which the property is specified.

- In option “O1”, “ $Ev1$ ” in the property refers to “T1 + 1mn”.
- In option “O2”, “ $Ev1$ ” in the property refers to “T2”.

An accuracy (or acceptable variability) shall be specified for each occurrence of an event. Indeed, without such accuracy specification, the description

may over-constrain the system design. A line of the table may look like:

Time	Events	Accuracy
T	Ev	[-100ms .. 500ms]

Figure 10: Specification of an accuracy

Meaning that the “Ev” event shall occur after “T - 100ms” and before “T + 500ms”.

At first sight, a line of the table shall be executed strictly before the next one.

- In option “O1”, it means that “ $T1 + 1mn < T2 - 2mn$ ” and “ $T2 - 2mn < T2 + M1$ ”.
- In option “O2”, it means that “ $T1 + 1mn < T2$ ” and “ $T2 < T2 + M1$ ”.

After further discussion with the system engineers, this last rule has appeared to be too simplistic as shown by the following example. Let us consider the case of an equipment warmup. The system shall wait until the piece of equipment is ready before using it. In practice, the duration of an equipment warmup is not deterministic. Let say that it take between 2mn and 6mn.

Here is an example of sub-sequence with an equipment warmup (where the “other event” is independent from the equipment warmup).

Time	Events	Accuracy
T	Start warmup	[0ms .. 0ms]
T+5mn	Other event	[0ms..0ms]
T+6mn	End warmup	[-4mn, 0ms]
T+7mn	Use equipment	[0ms .. 0ms]

Figure 11: Sub-sequence with an equipment warmup

Through this description, a system engineer wishes to express the fact that “end warmup” occurs (obviously) after “start warmup” and that “use equipment” shall occur after “end warmup”. However, he/she has not in mind that “end warmup” shall occur after “other event”. Indeed, “end warmup” can occur between “T+2mn” and “T+6mn”, i.e. potentially before the Time of occurrence of “other event”. These two events are in fact not related.

We have thus decided to provide explicitly a “kind” to the events to distinguish “input events” and “output events”. We obtain the sequence:

Time	Events	Kind	Accuracy
T	Start warmup	Out	[0ms .. 0ms]
T+5mn	Other event	Out	[0ms..0ms]
T+6mn	End warmup	In	[-4mn, 0ms]
T+7mn	Use equipment	Out	[0ms .. 0ms]

Figure 12: Specification of the input / output events

The previous rule can then be refined by

- An “output” event shall always occur after the event specified in the previous line. It means intuitively that the system shall wait until the end of the execution of the previous line.
- The accuracy of an “input” event shall always have a null upper bound. It means that the column “Time” specifies the latest possible time of occurrence of the input event (it can occur before this time but not after).

On this example, it implies that “use equipment” shall occur after “end warmup” but that the “other event” and “end warmup” are not scheduled. Intuitively, it means that the “input event” “end warmup” has not to “wait” for the “output” “other event” (which would be in contradiction with the definition of “input” and “output” events).

Finally, considering all the possible dispersions (actual engine thrust, occurrence of failures), the mission analysis computes minimal and maximal values the key-event’s times can take. For instance:

- T1 in [Min1 .. Max1]
- T2 in [T1+Min2 .. T1+Max2]

Meaning that it is physically not possible for T1 to be less than “Min1” or greater than “Max1”, and that T2 is defined relatively to T1.

Thanks to these rules, the initially informal description of the launcher sequences can now be automatically analysed.

4.2 Formal analysis

In the definition of T1 and T2, “Min1”, “Max1”, “Min2”, “Max2”, “M1” and “M2” are mission data corresponding to times or durations (i.e. they are constants which values depend on the mission). The launcher sequence is completely defined by the enumerated mission data “Option” and these timed mission data. In addition, the launcher sequence will be correctly defined if the values taken by this mission data are compatible with the rules expressed section 4.1.

As introduced in section 3.5, ArianeGroup uses a DSL to model the launcher sequences and an automatic code generator for their implementation (this model being an implementation of the Figure 3 seen as a specification).

In order to ensure that the final product is correct, one has to demonstrate that:

- The design is consistent with the rules specified section 4.1.
- The code (automatically generated from the DSL) is consistent with the design.

From Figure 8 and Figure 9, one can derive some equations on the mission data:

- For “Option = O1”:

- “Min2 - 2mn > max1 + 1mn” (scheduling between the two first lines of Figure 8)
- “T2 + M1 > T2 - 2mn” (scheduling between the two last lines of the Figure 8), i.e. “M1 > -2mn”
- “Max2 + M1 <= max1 + 1mn + M2” (property)
- For “Option = O2”:
- “Min2 > max1 + 1mn” (scheduling between the two first lines of Figure 9)
- “T2 + M1 > T2” (scheduling between the two last lines of the Figure 9), i.e. “M1 > 0”
- “T2 + M1 <= T2 + M2” (property), i.e. “M1 <= M2”

In addition some obvious rules shall be applied, such as “Max1 >= Min1” and “Max2 >= Min2”.

These equations may be solved with a simplex algorithm to compute minimal values for the mission data Min1, Max1, Min2, Max2, M1 and M2.

A system simulator (including the flight software and simplified behaviours of the avionics equipment) is then applied to analyse each sequence, using these minimal values of mission data to ensure that the launcher sequences are correctly implemented in the flight software [7]. The demonstration of the satisfiability of this set of equations ensures that the system is correct for any values of mission data.

5 Process of development of the launcher sequences

This approach of formalised requirements and analysis has been included in a more general process involving also non-formalised requirements. The following diagram summarises this development process.

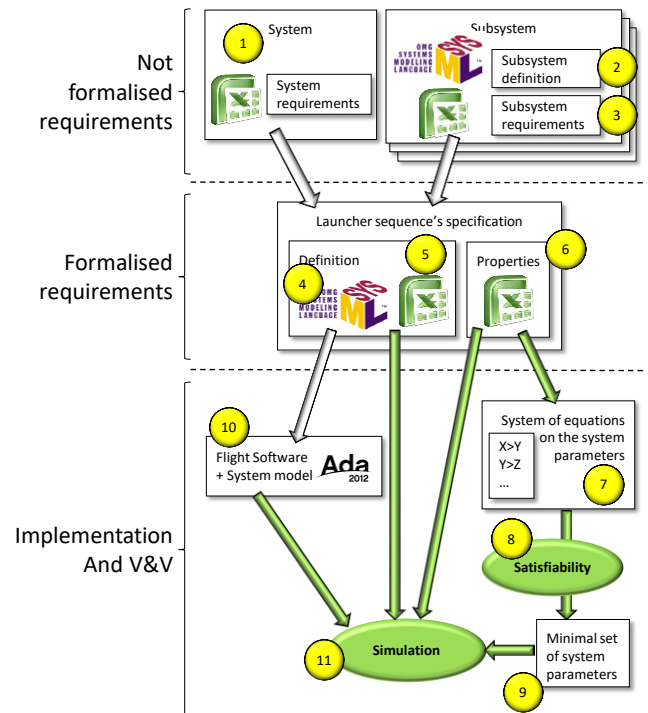


Figure 13: Launcher sequence development process

(in the following description, the numbers inside the parentheses refer to the numbers in the Figure 13)

- Each phase of the launcher mission (ground phase, atmospheric / extra-atmospheric phases, propulsive / orbital phases...) specifies transverse system requirements (1) toward the launcher sequences. These requirements are not formalized (written in textual English in an Excel file).
- Each subsystem has its own definition partially formalised in SysML (2) (customized and completed by a specific profile, e.g. with state-charts, internal block diagrams, etc.), and completed by not formalised textual requirements (3) written in textual English in an Excel file (e.g. a minimal delay is required before the opening and a closing of an electro-valve).
- All the requirements toward the launcher sequences are then analysed and formalised:
 - In a second SysML model (4) to capture the sequence's architecture (see the example of Figure 2). This SysML model is developed in co-engineering by the method team for the system team.
 - In Excel sheets to formalize the sub-sequences detailed definitions (5) and the properties (6) on the sequences. The consistency between the formalized descriptions and the non-formalized ones is performed by manual review.

- This formalised specification is automatically analysed by an in-house tool to generate a set of equations on mission data (7) for each sequence, i.e. for path of the sequence's architecture.
- A classical off the shelf "simplex" algorithm is applied on each set of equations to automatically demonstrate that these equations are satisfiable (8) (i.e. that there is a solution to the problem) and to automatically generate minimal sets of mission data satisfying this set of equations (9).
- The flight software is implemented (10) through SysML and DSL modelling (manually developed) and automatic code generation by an in-house tool.
- For each sequence (i.e. each path of the sequence's architecture), the set of minimal mission data (7) is used to test (with automatic test generation by an in-house tool) that the flight software behaves exactly as specified by the requirements (11).

The system engineers use only Excel to write their requirements. They are also responsible for the content of the SysML models (sequence's architecture and implementation) but these models are developed by dedicated teams (specialists in the use of modelling tools). The process starts by not formalised descriptions ((1) and (3)) to migrate to formalised ones during the development ((2), (4), (5), (6) and (7)). The automated mathematical analysis can discover errors very early in the development, allowing quick loops of errors / corrections.

6 Examples

6.1 Toy example

Let us first demonstrate this approach on a very simplified toy example.

- The launcher is composed of two stages
- The launcher is able to fulfil two missions (i.e. it requires two launcher sequences)
 - Without re-ignition of the second stage engine.
 - With re-ignition of the second stage engine.
- The second stage engine needs a warmup of 10mn period before being ignited (or re-ignited).
- When a stage engine is not used any more, it shall be passivated

The launcher sequence may thus be informally described by

- First stage engine ignition (IGNI1)

- Second stage engine warmup (WARM2)
- First stage passivation (PASS1)
- Second stage engine ignition (IGNI2)
- (only for a mission with two second stage ignitions)
 - Second stage engine stop (STOP2)
 - Second stage engine warmup (WARM2)
 - Second stage engine ignition (IGNI2)
- Second stage passivation (PASS2)

With the following timing:

Time	Events	Remark
T_IGNI1	IGNI1	For all mission
T_IGNI21 - 10mn	WARM2	For all mission
T_PASS1	PASS1	For all mission
T_EWARM2	END_WARM2	For all mission
T_IGNI21	IGNI2	For all mission
T_STOP21	STOP2	Only for a 2 nd ignition
T_IGNI22 - 10mn	WARM2	Only for a 2 nd ignition
T_EWARM2	END_WARM2	Only for a 2 nd ignition
T_IGNI22	IGNI2	Only for a 2 nd ignition
T_PASS2	PASS2	For all mission

Figure 14: Informal specification of launcher sequences

The launcher flight software shall be able to implement the two sequences. A mission data called "Option" is then defined to specify to the flight software which launcher sequence shall be executed ("Option" can take the value "One_Ignition" or the value "Two_Ignitions").

The architecture of these two sequences can be modelled with a SysML activity diagram in the following way.

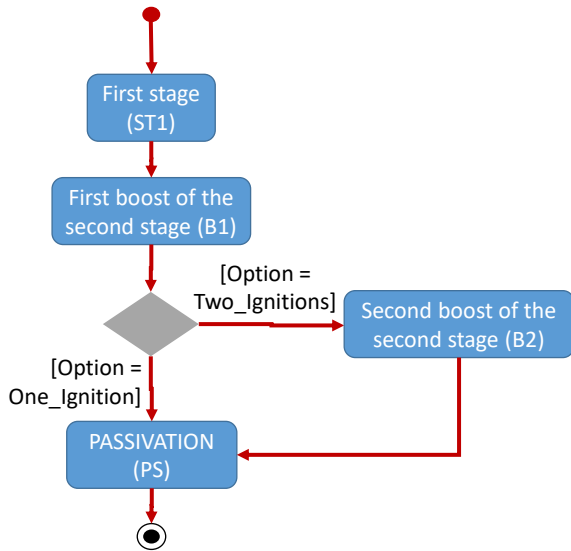


Figure 15: Two simplified launcher sequences

The detailed design of these launcher sequences is provided in an Excel sheet.

Seq	Time	Events	Kind	Accuracy
ST1	T_IGNI1	IGNI1	Out	[0s..0s]
B1	T_IGNI21 - 11mn	WARM2	Out	[10ms..10ms]
ST1	T_PASS1	PASS1	Out	[10ms..10ms]
B1	T_IGNI21 - 1mn	END_WARM2	In	[-10s..0s]
B1	T_IGNI21	IGNI2	Out	[10ms..10ms]
B2	T_STOP21	STOP2	Out	[10ms..10ms]
B2	T_IGNI22 - 11mn	WARM2	Out	[10ms..10ms]
B2	T_IGNI22 - 1mn	END_WARM2	In	[-10s..0s]
B2	T_IGNI22	IGNI2	Out	[10ms..10ms]
PS	T_PASS2	PASS2	Out	[10ms..10ms]

Figure 16: Formalised description of launcher sequences

The two stages are modelled by the following state-charts

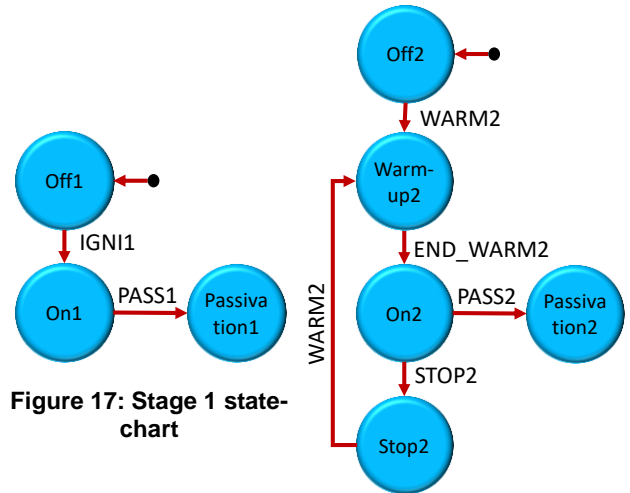


Figure 17: Stage 1 state-chart

Figure 18: Stage 2 state-chart

The flight software is finally automatically generated from the following description using the DSL:

```

-- T_IGNI1 is specified by the mission
at T_IGNI1: IGNI1;
-- T_IGNI21 is computed by the guidance
at T_IGNI21 - DWARMUP: WARM2;
-- T_PASS1 is computed by the guidance
at T_PASS1: PASS1;
-- END_WARM2 is supposed to have already occurred
at T_IGNI21: IGNI2;
if Option = Two_Ignitions then
  -- T_PASS1 is computed by the guidance
  at T_STOP21: STOP2;
  -- T_IGNI22 is computed by the guidance
  at T_IGNI22 - DWARMUP: WARM2;
  -- END_WARM2 is supposed to have already occurred
  at T_IGNI22: IGNI2;
end if;
-- T_PASS2 is computed by the guidance
at T_PASS2: PASS2;

```

Figure 19: Implementation of the toy example

The following equations on mission data are then automatically generated with the option with two engine ignitions (the option with one engine ignition is similar):

- $T_IGNI1 < T_IGNI21 - 11$
- $T_IGNI21 - 11 < T_PASS1$
- $T_PASS1 < T_IGNI21$
- $T_IGNI21 - 1 < T_IGNI21$
- $T_IGNI21 < T_STOP21$
- $T_STOP21 < T_IGNI22 - 11$
- $T_IGNI22 - 11 < T_IGNI22$
- $T_IGNI22 - 1 < T_IGNI22$
- $T_IGNI22 < T_PASS2$

The three equations

- $T_IGNI21 - 1 < T_IGNI21$

- $T_IGNI22 - 11 < T_IGNI22$
- $T_IGNI22 - 1 < T_IGNI22$

are obvious and can be ignored. Therefore, it remains:

- $\max(T_IGNI1) < \min(T_IGNI21) - 11$
- $\max(T_IGNI21) - 11 < \min(T_PASS1)$
- $\max(T_PASS1) < \min(T_IGNI21)$
- $\max(T_IGNI21) < \min(T_STOP21)$
- $\max(T_STOP21) < \min(T_IGNI22) - 11$
- $\max(T_IGNI22) < \min(T_PASS2)$

Any launcher mission shall ensure that the mission data fulfil these equations in order to be achievable.

A simplex algorithm allows computing the minimal values of these mission data satisfying these equations:

- $\min(T_IGNI1) = \max(T_IGNI1) = 0$
- $\min(T_IGNI21) = \max(T_IGNI21) = 12$
- $\min(T_PASS1) = \max(T_PASS1) = 2$
- $\min(T_STOP21) = \max(T_STOP21) = 13$
- $\min(T_IGNI22) = \max(T_IGNI22) = 25$
- $\min(PASS2) = \max(T_PASS2) = 26$

These values allow verifying by tests automatically generated by an in-house tool that the implementation is consistent with the design.

It shall be noticed that these tests are representative only for the verification of the launcher sequence. A real mission may have a real duration of several hours but we have demonstrated that the test with a mission of 26mn is representative enough. This “time” reduction makes the test approach achievable.

6.2 Ariane 6 example

This approach has been used to develop the Ariane 6 launcher sequences:

- Around 20 subsystems
- The maximal number of states of a sub-system state-chart is around 500.
- Several hundreds of launcher sequences (considering the options and the degraded scenarios)
- Around one hundred equations for each launcher sequence

7 Comparison with other approaches

This section compares practically the approach described in this paper with former experiments conducted by ArianeGroup (and is thus not a theoretical academic comparison with an exhaustive state of the art).

7.1 Deductive proof

Deductive proof (such as SPARK [2] or FramaC [8]) is adapted to static properties (such as “a data X is always greater than a data Y”) but not to timed properties (such as “the event E1 occurs always 10 seconds after the event E2”).

ArianeGroup developing its flight software in Ada, the use of SPARK has been experimented. This technic is able now to automatically demonstrate complex properties but not the ones addressed in this paper.

7.2 Timed model checking

Timed model checking (e.g. UPPAAL [3], or model checking based on SysML [4]) is limited by the combinatory explosion of the cases. On case studies provided by ArianeGroup, time model checking was working only on small models (limited to a few key-events).

In addition, if some formalised descriptions usable for formal proof may be quite intuitive (e.g. SysML state-charts), others may be more difficult to understand (e.g. μ -calculus).

Moreover, using such technic requires a model translator from the ArianeGroup models toward the tool formalism. Such model translator may be quite costly to develop.

Such technic relies also on semantics proposed by the model checking tools which is not the semantics of the systems developed by ArianeGroup (asynchronous semantics versus synchronous semantics). The qualification of the model translator may be complicated because it requires an accurate mastering of the source and target semantics.

The demonstration that the used model is a correct abstraction of the software code is thus not straightforward.

7.3 Certified code generator

A more adapted approach would be the use of a toolset providing at the same time a modelling environment, a proof engine and a certified code generator. Such toolset ensures then that the results of the formal proof are applicable to the product.

ArianeGroup has experimented the SCADE Suite [1]. Its main advantages are its formal semantics, the synchronous approach that is compatible with a launcher design and its certified code generator. Several formal proof tools are also available on SCADE Suite models. However, SCADE Suite has been abandoned because the data-flow interpretation of the state-charts is not intuitive and not adapted. The even-driven interpretation of SysML corresponds better to the need.

In addition, the proof tools analysing SCADE Suite models have the same limitation than the timed model checkers (see section 7.2) and are not able to analyse non-trivial models.

8 Conclusion

Excel and even SysML are not formal methods. Anyway, they are often used for the development of complex systems because they are intuitive tools well mastered by the system engineers. This paper has shown how a development can start from high-level informal requirements to be refined to low-level formal requirements which can themselves be translated to simple mathematical equations.

Tests may be used to demonstrate in a quite deterministic way the equivalence between the “Excel model” and the implementation.

The approach described in this paper is less formal than the technics described section 7. However, it is often sufficient even if it is obviously not complete in some cases. Let us consider the following example, close from the ones of section 4.2.

Times	Events
T1	Ev1
T2	Ev2

Figure 20: Simple example

Even if it is not formally demonstrated, we have the feeling that equation $\max(T1) < \min(T2)$ is always sufficient. And the use of the minimal values is also sufficient to demonstrate that the implementation is consistent with the design.

However, a property “ $Ev2 \leq Ev1 + \Delta$ ” cannot be demonstrated correct with this approach.

9 References

- [1] SCADE - <https://www.ansys.com/products/embedded-software/ansys-scade-suite>
- [2] SPARK - <https://www.adacore.com/sparkpro> - See also an example of use in the space domane “Rail, Space, Security: Three Case Studies for SPARK 2014”, C. Dross, P. Efstathopoulos, D. Lesens, D. Mentrée, Y. Moy, ERTS2014
- [3] UPPAAL - <https://uppaal.org/>
- [4] “Formal Model Driven Engineering for Space Onboard Software”, E. Conquet, F-X. Dormoy, I. Dragomir, D. Lesens, P. Nienaltowski, I. Ober, ERTS2012
- [5] SysML - www.omg.sysml.org
- [6] “From functional definition to software code”, D. Lesens, ERTS2016
- [7] “Automatic Test Generation - An Industrial Feedback”, M. Ducamp, D. Lesens, P. Ranoarivony, ERTS 2022
- [8] Frama-C - <https://www.frama-c.com>
- [9] INCOSE Systems Engineering Handbook (INCOSE-TP-2003-002-03.2.2)

Session We.4.C
Model Driven Development

Wednesday 12th June

15:00

–

Pastel Room

Large legacy systems design maintainability through modeling

MOHAMED HABIB ESSOUSSI

Airbus Operations S.A.S
+33 6 08312231

mohamed-habib.essoussi@airbus.com

Paul VIVOT

Airbus Operations S.A.S
+33 6 07601685

paul.vivot@airbus.com

Lucas DELOYE

SII
+33 5 34615959

lucas.deloye.external@airbus.com

Davi Henrique de Sousa Pinto

Airbus Operations GmbH
+49 15111097167

davi-henrique.sousa-pinto@airbus.com

Abstract. Model-Based System Engineering (*MBSE*) and particularly Model-Based Product Line Engineering (*MBPLE*) now stands as the new standard for systems engineering at Airbus Group. Indeed, the Airbus *MBSE* Architecture Framework (*R-MOFLT*)[3] and its feature-based product line engineering framework extension (*MBPLE4MOFLT*) are widely deployed on Research & Technology projects. This paper tackles the applicability of such enablers to large legacy systems. As such, it outlines a proof of concept on redesigning a legacy system using *MBPLE4MOFLT* as a new product line based on several in-service variants definitions that have been designed over the last four decades following document-based ways of working. As such, the interoperability between these ways of working and the new digital assets is essential to achieve this migration on one side and, once migrated, to ensure backwards compatibility with the official process, on the other side. To this aim, besides using existing data hubs between *Cameo Systems Modeler* and *Rational DOORS*, the Airbus *MBSE SysML* profile has been extended with further customizations to fit the new product line design golden rules. Wizards are also proposed to ease authoring and impact analysis. Finally, a new plugin has been developed to automate the variability propagation throughout variable assets and to ensure consistency between the variability handled with *MBPLE4MOFLT* and the requirements applicabilities handled in *Rational DOORS*.

Keywords. MBSE, MBPLE, system engineering, industrial application, avionics.

1. Introduction

Recently, Airbus has launched a new Flight Management System (nFMS) product line to

capitalize on four decades of designs and improvements across Airbus aircraft families.

This product line aims to reduce non-recurring costs and time to market while preparing for the future. With *MBSE* being already deployed in the "Zero Emission" (ZEROe) Demonstrator[2][5], the question was to assess the methodology maturity against the new product line needs. It is crucial in our approach to underscore that unlike ZEROe, nFMS does not commence from a clean state, but rather inherits a substantial legacy. As such, a proof of concept has been conducted, prior to an eventual industrial deployment as an enabler of this development. The proof of concept also questioned the actual need of *MBSE*. It explored how *MBSE* can assist system designers who may not be familiar with it in their day-to-day tasks in the context of a high-pressure delivery rate. The objective was to maintain interoperability with the existing process while preserving the current format for sharing artifacts with suppliers.

Consequently, the primary focus shifted swiftly to the challenge of effectively managing a large volume of textual requirements. The emphasis was on organizing them within a robust specification, prioritizing solid variability management over modeling the system behavior. Model execution hence falls out of the scope.

Nevertheless, we explored enhanced collaborative work methods, digital continuity, document generation, and impact analysis through *MBSE*. This paper delves into these aspects, demonstrating *MBSE*'s role as a key enabler in large legacy systems, using an industrial use case.

The rest of the paper is organized as such: section 2 presents the methodology, from the state of the art down to the tailoring and customizations. Section 3

outlines the current process of designing a flight management system at Airbus. Section 4 is the proposed approach within the framework of the project assumptions. Section 5 exposes the first results with benefits, challenges and limitations then concludes.

2. Methodology

INCOSE's visions for 2020[12] and 2025[13] correctly predicted and anticipated the widespread adoption of *MBSE* as a de-facto way of deploying Systems Engineering for complex systems.

Following the trend, the industry, academia and the systems engineering community have at their disposal several comprehensive frameworks, such as DoDAF¹, MoDAF², UAF³, Arcadia[14]. These frameworks are continuously improved, matured and tailored to achieve business and technical needs.

Airbus MBSE Framework. Airbus' own *R-MOFLT* Architecture Framework[3] has been developed to tackle the organization's challenges regarding SE and *MBSE*. It is based on the SysML language[15] and expands and combines existing frameworks to support Airbus' products, services and industrial systems.

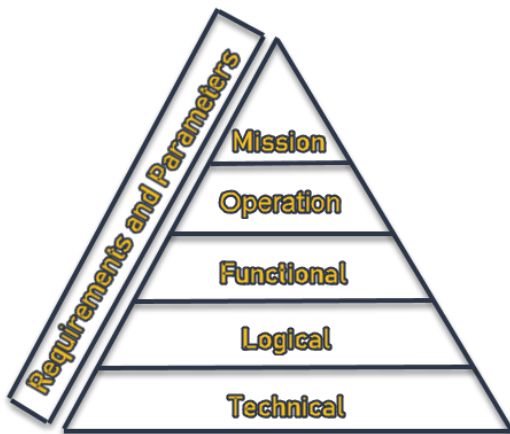


Figure 1: R-MOFLT Architecture Framework Scheme

Airbus MBPLE Framework. As part of providing an integrated solution, an *MBPLE*[1][4] approach based on the principles described in the ISO 26580[18] is being developed. It currently supports variability on architecture and requirements, but its ambition is to

¹ Department of Defence Architecture Framework (U.S. Department of Defence 2010)

² Ministry of Defence Architecture Framework (U.K. Ministry of Defence 2012)

³ Unified Architecture Framework (OMG 2020)

provide holistic product line approach capabilities, by expanding existing digital assets to include variation points, while maintaining a single source of variability, through feature models. For the project described in this paper, two methods from the *MBPLE* framework have been used and tailored:

- **MBPLE4MOFLT.** Expansion of the *MOFLT* framework. It includes variability and is used to represent a product line system architecture (or 150% models). It includes a PLE configurator capable of generating product line variants (or 100% models).

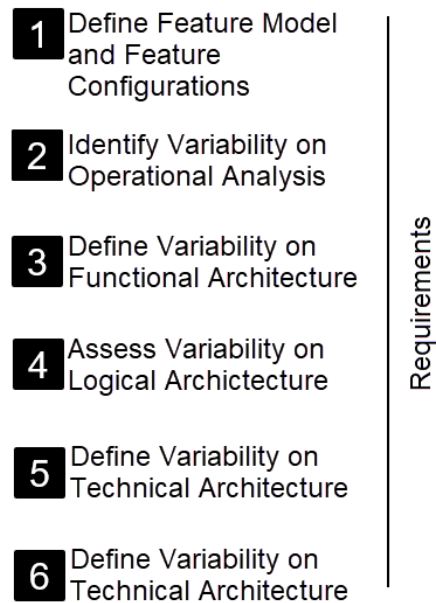


Figure 2: MBPLE4MOFLT scheme

- **MBPLE for Requirements and V&V Management (MBPLE4RVVM).** Expansion of the model-based requirements management methods. It includes existence (meaning a textual exists or not for a certain variant configuration) and parametric variability to textual requirements. This method is integrated and meant to be used with *MBPLE4MOFLT*.

Tailoring. The digitalisation and transition to a model based approach is an even bigger challenge in large and complex organizations such as Airbus. This puts a big emphasis not only on the framework itself, but also on organizational and transitional aspects[17]. Programmes have been running for decades and plenty of data is still stored in older means (hardware and/or software).

In this context, automations and tailoring were developed to interface with “non model-based”

documents and data formats still required by some processes. These automations are better described in Section 4 - Proposed Approach.

3. Analysis of current practices

Within the Airbus framework, the newly developed nFMS adheres to the well-structured Airbus process titled "Define System Requirements and Design." This procedural guideline outlines a comprehensive set of deliverables, encompassing not only the development and process assurance plan but also a collection of artifacts that intricately delineate the system requirements, design specifications, and interface parameters.

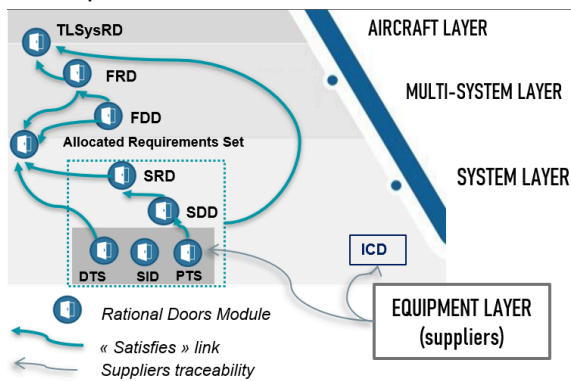


Figure 3: nFMS RDBS

Illustrated in Figure 3, these artifacts are systematically arranged within a Requirements Documents Breakdown Structure (RDBS), with the Detailed Technical Specification (DTS) and the System Interface Documents (SID) serving as the central focal points of this organizational framework at system layer. Requirements within the DTS and SID satisfy upper-level requirements cascaded from the multi-systems function layer such as the Function Definition Documents (FDD) and the Top-Level System Requirements Document (TLSysRD). Subsequently, the requirements embedded in the DTS and SID are disseminated to suppliers, accompanied by the Purchase Technical Specification (PTS) delineating the definition of equipment, encompassing both software and hardware. The seamless interconnection and cascade of these design artifacts is facilitated through the utilization of *Rational DOORS*[22]. Beyond the functional requirements, the system design incorporates the interface definition, meticulously documented within an Interface Control Document (ICD[9]) facilitated by the Albatros⁴ tool. This document comprehensively outlines the

physical characteristics of network messages, digital inputs and outputs labels and buses, as well as analog and discrete inputs and outputs signals.

A critical challenge arises due to the absence of digital continuity among the DTS, the SID, and the ICD. This lack of integration complicates impact analysis processes, introducing inefficiencies and time-consuming evaluations of potential design evolutions consequences.

Moreover, the current methodology falls short in addressing essential product line considerations, specifically in defining variability at the requirement level. This limitation poses a significant hurdle in achieving a comprehensive understanding of the system's adaptability to diverse configurations.

To overcome these deficiencies, the design office has developed ad-hoc golden rules tailored to nFMS as a multi-program and a multi-supplier product line, where the requirements applicabilities are set with the targeted configurations (Airbus aircraft families) and the stakeholders with the suppliers as depicted in the template below:

```

DTS FMS-xxx-xxx-xx-xx
<Requirement Statement>
Rationale:<rationale of the requirement>
Additional Information:<any additional info to support the understanding of the requirement>
Applicability:<aircraft configurations>
Change Rationale:<change rationale with regards to existing products>
Source:<pre-nFMS source>
Stakeholders:<list of suppliers>
Working Status:<working status>
    
```

Before integrating updates into the baseline DTS, the design office initiates a Specification Evolution Request (SER) outlining the changes towards the suppliers. This delta is then assessed for cost implications. Upon agreement between the involved parties, the delta is manually incorporated into the complete DTS.

4. Proposed approach

This section provides a comprehensive exploration of the pragmatic solutions implemented to address the design definition and maintainability challenges outlined in Section 3, employing *MBPLE4MOFLT*. Emphasis is placed on the significance of ad-hoc customizations tailored to deliver substantial value to the project.

a. Project assumptions

The objective of this proof of concept is to showcase the effectiveness of improving the authoring process for DTS and SIDs requirements. The primary focus is on establishing robust linkages between these

⁴ The official interface management tool at Airbus

requirements, facilitating smooth impact analysis. Furthermore, the goal is to forge connections with ICDs parameters, thereby establishing a digital continuity that extends from the functional and interface requirements to the intricate details of physical connectors within avionics computers backplanes.

As depicted in figure 4, to accompany the change smoothly, bridges and hubs (supporting the exchanges depicted as red arrows) are opted over disruptiveness so that interoperability with the current process is maintained. A primary motivation behind the adoption of this new approach is to ensure user-friendliness for designers unfamiliar with MBSE, aiming to seamlessly integrate with existing workflows for authoring whilst taking advantage of new automated time-saving impact analysis features. MBSE serves in our context as a facilitator for elevated requirement-based engineering.

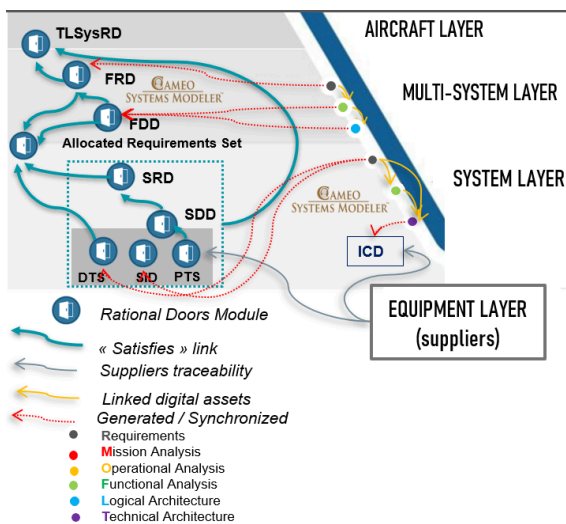


Figure 4: nFMS RDBS with R-MOFLT

b. Product Line Engineering

The management of the product line is facilitated through a rigorous framework comprising a feature model and a feature configuration. The feature model serves to formalize the pivotal factors influencing variability, delineating the software's adaptability. As described in Figure 5, these factors include specific features per supplier, cockpit HMI type, and aircraft type, thus ensuring both software adaptability and portability. The feature configuration, on the other hand, is determined by a triplet encompassing aircraft type (such as A320, A330, A350), aircraft variants (such as Future architecture or Legacy), and suppliers.

Legend							
Select							
FC	ConfigurationSpace	27	27	28	28	28	28
EM	nFMS Product Line Feature Model						
F	F0000_nFMS_Product_Line	✓	✓	✓	✓	✓	✓
F	F0001_Hardware_Adaptability	✓	✓	✓	✓	✓	✓
F	F0010_FMC	✓	✓	✓	✓	✓	✓
F	F0011_FMGE	✓	✓	✓	✓	✓	✓
F	F0023_HMI	✓	✓	✓	✓	✓	✓
F	F0230_MFD	✓	✓	✓	✓	✓	✓
F	F0231_MCDU	✓	✓	✓	✓	✓	✓
F	F0022_Suppliers_Specificities	✓	✓	✓	✓	✓	✓
F	F0220_SupplierB	✓	✓	✓	✓	✓	✓
F	F0221_SupplierA	✓	✓	✓	✓	✓	✓

Figure 5: Extract from nFMS Feature and Configuration matrix

This structured approach enables the transformation of the model from 150% to 100% based on a predefined set of configurations. The 150% model is founded on the principles of clean and direct methodologies, as elaborated upon in greater detail in section 4.c. Moreover, the product line exhibits robustness against configuration evolution, adeptly accommodating changes such as supplier product evolution or the introduction of new aircraft models. Once both feature model and configuration are established, features are articulated as elementary constraints. To mitigate workload and minimize errors, features are categorized at function level and subsequently propagated to requirements, coupled with validation rules. To ensure interoperability with Rational DOORS, the configuration of aircraft and supplier specifications is systematically calculated at the requirement level, with the resulting data automatically integrated into the applicability and stakeholders attributes. This process is facilitated through a specialized plugin and additional customizations, which encompass derived properties implemented by opaque behaviors. The overarching objective of this automated tool chain is to forge a cohesive connection between MBPLE, leveraging digital assets as the definitive source of truth, and traditional Product Line Engineering (PLE) reliant on textual attributes within Rational DOORS.

c. Architecture

Modeling the functional breakdown through SysML Block Definition Diagrams (BDD) and system architecture via SysML Internal Block Diagrams (IBD) presented a strategic opportunity to mitigate complexity in the context of legacy large systems. The approach involved conceiving this extensive system as a hierarchical arrangement of functions, transcending conventional document-based representations. This method facilitated a holistic comprehension of the interconnections among functions and the corresponding functional flows, exemplified by the Airbus MBSE SysML Profile's extension of SysML proxy ports stereotypes.

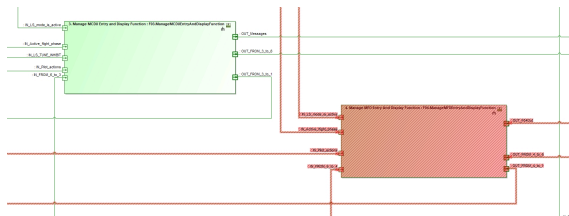


Figure 6: Extract of nFMS 150% functional architecture with a model transformation preview for a particular variant with Pure::Variants[19]

The functional analysis adhered to the direct method (150%, c.f., figure 6), as advocated in [2]. The resultant model enabled the establishment of hyperlinks from specific parts of requirements' textual statements to functional flows. This serves a dual purpose: validating requirement allocations for enhanced organizational structure and enabling metachain navigation essential for impact analysis, thereby enhancing design maintenance (c.f., 4.b). Figure 7 illustrates the process wherein functional flows, originally designated as generic and referenced in the cross-programs DTS requirements, undergo specialization to manifest as specific parameters tailored for each aircraft within the ICD[9]. In this context, the functional interface, initially representing a generic parameter, extends specific functional interfaces defined as variant assets. This relationship ensures that the former strictly inherits the variant functional flow properties of the latter.

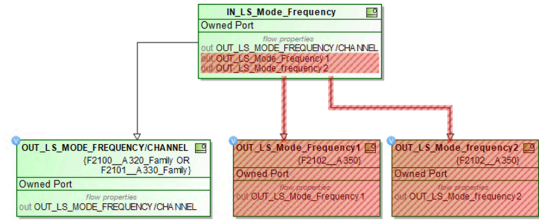


Figure 7: Example of functional flow with associated ICD specific parameters with a model transformation preview for a particular variant with Pure::Variants

As exemplified in figures 8 and 9, these ICD parameters are conveyed then through nested ports which found representation in technical interfaces—conforming to the Airbus MBSE SysML Profile's extension of SysML proxy ports stereotypes—defined within the technical architectures.

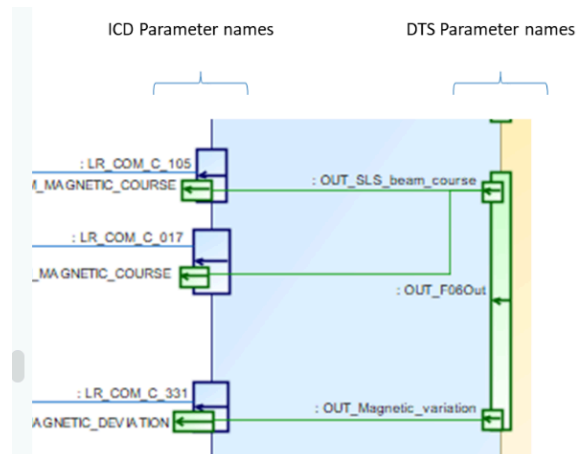


Figure 8: Example of two ICD parameters delegated to one DTS generic parameters and realized by two different digital (A429[10]) labels

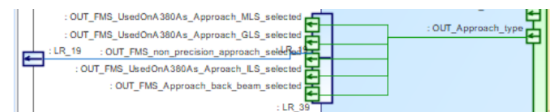


Figure 9: Example of 6 ICD parameters delegated to one DTS generic parameters and realized by one Functional Data Set (FDS) of an AFDX[11] message (avionics network)

Each variant followed the clean method[2]. Its technical interfaces, encompassing system inputs like physical connectors, digital labels, buses, network message structures, ports, and physical links, formed a crucial connection between the Functional (F-layer) and Technical (T-layer) components. This linkage supported digital continuity, allowing for bidirectional impact analysis. Figure 10 shows an example of a specific functional

flow (ICD parameter “LS_MODE_FREQUENCY/CHANNEL”) conveyed by a digital label (A429 label “033”) and transferred on a data bus which is connected to FMGC (Flight Management and Guidance Computer) backplane pins number “LM10A/10B”.

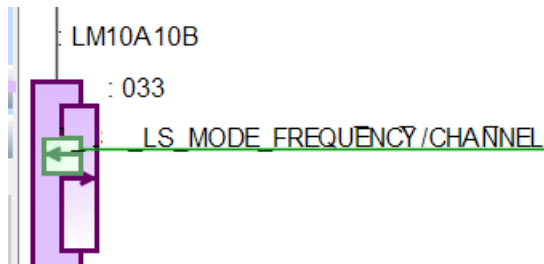


Figure 10: Example of an ICD parameter in F-Layer and T-Layer

d. Enhanced Requirement-based Engineering

Our approach is adaptable to legacy workflows, encompassing existing processes and exchanges with suppliers. Its primary objectives are to enhance engineers' comprehension and to support the 150% specification completeness.

Requirement authoring. The requirements authoring is supported by a wizard to streamline the automatic setup of attributes such as requirement labels (unique identifiers) and versions, ensuring adherence to the nFMS golden rules as sketched in the template defined in section 3. Engineers can focus on product design rather than formatting the specification. Allocation of requirements to functions is achieved through dependency links within traceability matrices. This linkage enables the propagation of function variation points, facilitating the automatic computation of applicability and stakeholder attributes (c.f 4.b). Subsequently, validation rules are applied to ensure that requirements are defined at the appropriate level, verifying that referenced generic functional flows (if any) are indeed utilized by the function to which the requirement is allocated. With this method, we intend achieve a well organized DTS

The maintenance of the requirements set adheres to the SER process outlined in Section 3, facilitated efficiently through the use of branches. The existing baseline resides within the trunk, with each branch dedicated to a distinct SER. Upon validation of functional evolution with suppliers, the branch is merged into the trunk, establishing a new baseline. Only potential conflicts necessitate the expertise of a systems engineer; all other components can be integrated without any loss of information. While

Rational DOORS remains integral to Airbus's operational methodology, the baseline is also shared within its framework. In a branch, the requirement label is constructed in the creation wizard by taking the branch name and associating it with a unique number per branch to identify new requirements. Once merged into the trunk, another wizard is launched on the requirement set to define the final label of created requirements. Each label is built by retrieving the identifier of the section in which the requirement resides, providing a unique number within the section, and retrieving the baseline number of the trunk in *Teamwork Cloud* to construct the version. Branches serve not only to highlight SER disparities with its original baseline but also enable comparison with the current iteration, ensuring comprehensive tracking of system evolution.

Impact analysis. Beyond merely linking requirements across diverse documents as discussed in Chapter 3, our approach involves referencing model elements such as generic functional flows, specific functional flows or technical flows (c.f. 4.c). By leveraging HTML hyperlinks embedded within requirement attributes such as statements, additional information or rationale, facilitating the creation of a relational map spanning different layers of the system architecture. Integration of such hyperlinks in a requirement statement is illustrated in Figure 11. In this scenario, both the “SLS is active” (Satellite-based Landing System), and the “FAS data block” (Final Approach Segment) functional flow interface types are referenced.

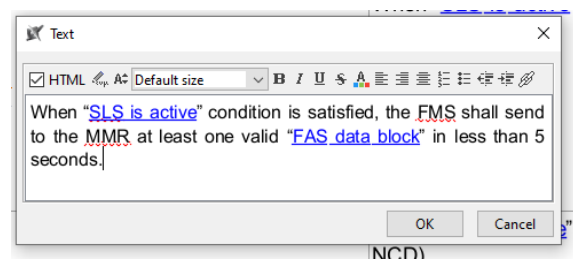


Figure 11: Requirement statement with hyperlinks

Through the utilization of MOFLT profile extensions and opaque behaviors, it is feasible to define a forward impact analysis, as illustrated in Figure 12. This analysis bridges the gap between requirements and functional flow interface types, along with their technical implementations. Notably, only the FAS data block is detailed in technical components as it is an ICD parameter (c.f. 4.c), while the "SLS is active"

Boolean is a generic specification-defined element whose technical definition remains undisclosed to Airbus.

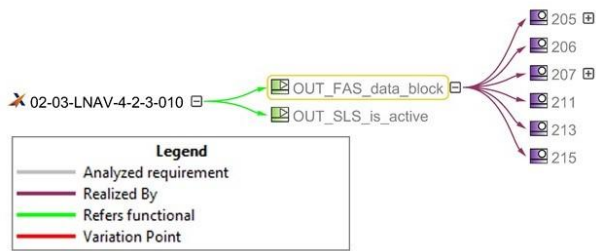


Figure 12: Example of a forward impact analysis (from a requirement)

Figure 13 demonstrates the capability to conduct a backward impact analysis, employing specific functional or technical flow in a manner akin to the forward analysis. For instance, all requirements affected by a system updating one of its interfaces with the FMS are enumerated.



Figure 13: Example of a backward impact analysis (from a technical interface type)

Both analyses are initiated via a customized wizard, wherein the references to SysML and MOFLT[3] are abstracted away from the user, ensuring a streamlined interface. For this purpose, realization and reference attributes are established, employing the MOFLT method, to link model elements between them by extending their profiles.

Document generation. Upon the creation of a new baseline, it is generated and shared with suppliers

after the model undergoes transformation into a 100% model (c.f. 4.c) to uphold confidentiality between parties.

Using *MBSELab@Cameo*, the methodology adopts an ontology-based[6][7][8] approach to maximize agnosticism towards the *MOFLT* framework. Indeed, in our approach, we purposefully separate the functional architecture hierarchy from the DTS hierarchy to minimize disruptions to the existing DTS and to avoid influencing the architecture retro-engineering process based on an unconventional document hierarchy. Within this ontology the DTS abstract elements such requirements, sections, chapters, free text and referenced interface data are defined alongside their attributes (c.f 3) and their relationships. They are subsequently mapped to stereotypes and organized according to the document pattern. Figure 14 displays a combination of ontology elements (green elements) and their corresponding mappings (purple elements). Certain attributes, such as Name and Description, are already established within a higher-level ontology and can be directly utilized in the document pattern.

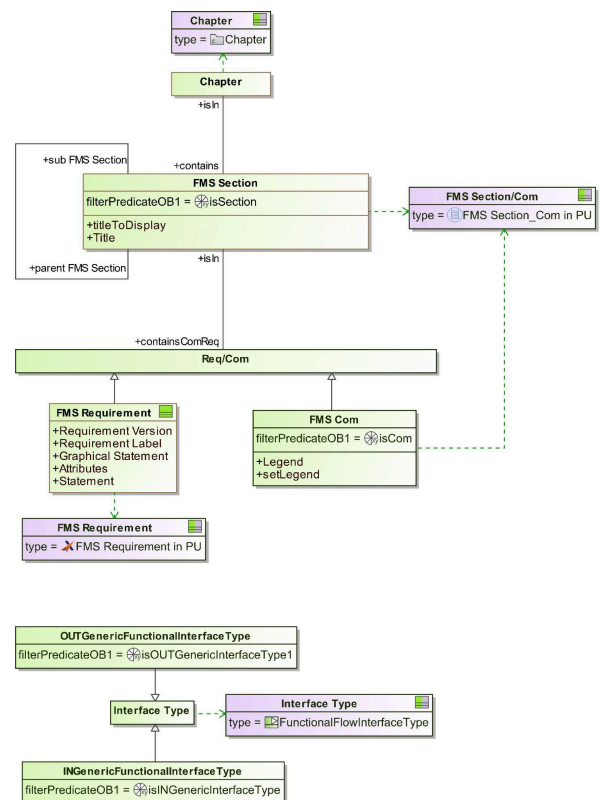


Figure 14: nFMS ontology and mapping diagram

MBSELab@Cameo's enablers facilitate the DTS generation as a henceforth ontology-based

document. Emphasizing PLE, they enable thanks to our ontology the automatic generation of mappings between generic functional interfaces and their specific variants, presented in a comprehensive matrix within the DTS appendix (see section 4.c). These enablers also allow the extraction of model diagrams for visually specific requirements and configuration management data from the *Teamwork Cloud* server. Indeed, in legacy documents, the initial pages are dedicated to cataloging the integrated components within the baseline and providing a brief history of the documents. Utilizing APIs, the baseline numbers, release dates and commit comments are seamlessly retrieved and incorporated into the generated document.

e. Tools

Having a methodology (tailoring of *MBPLE4MOFLT*) and a language (*SysML*) defined, one main challenge was to develop a toolchain able to answer all the needs while being interoperable with already existing tools used for other layers (equipment and multi-systems ones) and being user-friendly to ensure quick and efficient handling by system designers with no specific modeling background. To do so, some of the tools solutions promoted by Airbus Framework have been adopted. All the modeling activities are performed in *Cameo Systems Modeler 21x[20]*, including requirements authoring. Top-level requirements coming from the multi-systems layer and defined in *Rational DOORS* are imported in *Cameo Systems Modeler* thanks to *Cameo DataHub[21]*. Traceability between nFMS requirements and top-level one is performed directly in *Cameo Systems Modeler*. However, as *Rational DOORS* is the official database storing all requirements in Airbus, when a DTS baseline is frozen in *Cameo System Modeler*, requirements and their traceability links towards top-level requirements are exported in *Rational DOORS* through *Cameo DataHub*. Nevertheless, *Cameo Systems Modeler* remains the only source of truth for FMS requirements.

Collaboration between designers is ensured through *Teamwork Cloud*, allowing them to work on common shared models.

For the product line aspect, *SysFM*⁵ plugin is used to define the feature models and all the variation points, and *Pure::Variants* plugin is used for model transformation from a 150% model to a 100% one (for export by supplier for instance). Additionally, the

VPP⁶ plugin has been specifically crafted to propagate variation points from functions to their allocated requirements, while also computing the applicable variants from the variation points set with *SysFM*.

Finally, for cascading the requirements to suppliers (equipment layer), an export is performed directly from *Cameo System Modeler* to *Microsoft Word*.

The overall toolchain is represented in Figure 16.

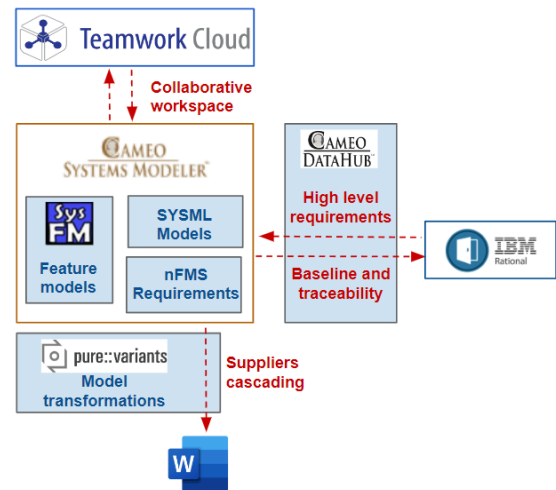


Figure 16: Developed toolchain

f. Customizations

The rationale behind the implemented customizations is to fulfill the nFMS golden rules by extending the *MOFLT* plugin with regards to the desired data model defined in section 3.

In these profile extensions, derived properties have been utilized for the calculation of requirements' stakeholders and applicabilities with respect to variability. These derived properties along with wizards defined in 4.d stand for opaque behaviors scripted in Groovy being well-documented in existing APIs, and with superior performance. Regarding impact analyses, efforts were made to base them as much as possible on *SysML* and *UML[16]* to avoid potential modifications to the existing opaque behaviors within the *MOFLT* framework.

Organizing these profile extensions, opaque behaviors, and wizards as project usage allows for their potential application in other models. Additionally, conducting tests when a new framework or *Cameo Systems Modeler* version is released prior to migrating the nFMS model ensures seamless integration and functionality.

⁵ Airbus SysML profile for feature modeling

⁶ Airbus Variation Propagation Plugin

5. Results assessment and future work

During this proof of concept, all DTS requirements were migrated from *Rational Doors* to *Cameo Systems Modeler*, totaling exactly 19626 requirements without encountering any significant delays in tool usage or project access. This successful migration underscores the scalability of the approach. However, only 0.5% of these requirements were retro-engineered within a functional and technical architecture to conduct impact analysis, though these examples were considered representative of the DTS content. This raises two potential paths forward for the proof of concept: selectively retro-engineering DTS components likely to evolve in future increments or exploring generative AI to address a broader scope. By leveraging *MBSE* for enhanced requirement-based engineering and fostering collaborative work, we anticipate a significant reduction of at least 15% in design lead-time, as we shift focus away from document formatting towards engineering digital assets.

Industrialization of this proof of concept is scheduled for 2025, with over 30 FMS designers expected to be involved. Additionally, exporting generated ICDs to the official interface management tool (Albatros) holds promise for further streamlining design processes and reducing lead times.

However, significant challenges remain, particularly in improving the performance of *MBSELab@Cameo* for DTS generation towards suppliers, which currently falls below industrial standards (1 hour for ~100 requirements). Backup solutions are being currently analyzed to mitigate this risk.

Moreover, a first limitation of the developed toolchain concerns a part of the specification composed of mathematical requirements coming from the aircraft performance equations. The capability of writing mathematical equations in *Cameo Systems Modeler* is limited. Being also implemented in Matlab Simulink by the flight physics team, a bridge between the *Simulink* models and the requirements in *Cameo Systems Modeler* might be considered in a future work.

Furthermore, beyond technical concerns, governance issues have surfaced, including the maintenance of customizations across *MOFLT* plugin releases. These considerations will require careful attention to ensure the long-term success of the project. Finally, it is imperative to develop a customized training program for the nFMS design team.

Conclusion

In this paper, we present a novel approach utilizing *MBSE* and *MBPLE* methodologies to enhance the maintenance of large legacy systems without disrupting procurement processes with suppliers. Our approach emphasizes interoperability with existing processes and official tools while minimizing the training burden on designers.

To accomplish this goal, we introduce a method for linking unaltered textual requirements with a retro-engineered architecture, facilitating impact analysis through a meticulously defined representation of interface parameters. This method, implemented as a semi-automated tool, serves as a bridge between PLE and MBPLE, fostering enhanced collaborative work among stakeholders. This customized and tailored solution is now prepared for full deployment on the current use case.

References

- [1] Ferrogali, M 2019. 'Modelling and Simulation @Airbus a fundamental digital transformation axis across product, manufacturing and support in service'. *Presentation at MODELS Conference 2019 – Industry Day 18 September 2019 | Munich - Germany*.
- [2] Henrique Madeira R, Henrique de Sousa Pinto D, A 2023. 'Variability on System Architecture using Airbus MBPLE for MOFLT Framework'. *INCOSE International Symposium*, vol.33
- [3] Ducamp, C, Bouffaron, F, Ernadote, D, Wirtz, J and Darbin, A 2022. 'MBSE approach for complex industrial organization program'. *INCOSE International Symposium*, vol. 32, no. 1, pp. 839-856.
- [4] Forlingieri, M, Weilkiens, T 2022. 'Two Variant Modelling Methods for MBPLE at Airbus'. *INCOSE International Symposium*, vol. 32, no. 1, pp. 1097-1113.
- [5] Airbus, ZEROe - Towards the world's first zero-emission commercial aircraft, viewed 11 August 2023, <<https://www.airbus.com/en/innovation/zero-emission/hydrogen/zeroe>>.
- [6] D. Ernadote, "Ontology-Based Pattern for System Engineering," 2017 ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MODELS), Austin, TX, USA, 2017, pp. 248-258, doi: 10.1109/MODELS.2017.4.
- [7] D. Ernadote, "An ontology mindset for system engineering," 2015 IEEE International Symposium

on Systems Engineering (ISSE), Rome, Italy, 2015, pp. 454-460, doi: 10.1109/SysEng.2015.7302797.

[8] Ernadote, Dominique. (2020). MB2SE: A Theoretical Foundation for Systems Engineering Habilitation à Diriger des Recherches. 10.13140/RG.2.2.12357.06886.

[9] Miguel Ángel Mozas Pajares, Carlos Murciano Díaz, Ismael Lafoz Pastor, Carlos Fernández de La Hoz. ICD Management (ICDM) tool for embedded systems on aircrafts. ERTS2 2010, Embedded Real Time Software & Systems, May 2010, Toulouse, France. Hal-02267704

[10] Aeronautical Radio Inc., ARINC specification 429-ALL: Mark 33 Digital Information Transfer System (DITS) Parts 1, 2, 3, 2001.

[11] Aeronautical Radio Inc., ARINC specification 664P7, Aircraft Data Network, Part 7, Avionics Full Duplex Switched Ethernet (AFDX) Network, 2005.

[12] International Council on Systems Engineering (INCOSE) 2007, *Systems Engineering Vision 2020*.

[13] International Council on Systems Engineering (INCOSE) 2014, *Systems Engineering Vision 2025*.

[14] Voirin, J.L., 2017. *Model-based System and Architecture Engineering with the Arcadia Method*. Elsevier.

[15] OMG 2018, *OMG Systems Modelling Language (OMG SysML), Version 1.6*, <<https://www.omg.org/spec/SysML/1.6/PDF>>.

[16] OMG 2016, *OMG® Unified Modeling Language® (OMG UML®), Version 2.5.1*, <<https://www.omg.org/spec/UML/2.5.1/PDF>>.

[17] ISO/IEC 2015, *Software and systems engineering-Reference model for product line engineering and management*, (ISO 26550: 2015).

[18] ISO/IEC 2021, *Software and systems engineering — Methods and tools for the feature-based approach to software and systems product line engineering*, (ISO 26580:2021).

[19] Pure Systems. 2023. pure::variants User Manual. Retrieved from <https://www.pure-systems.com/pv-update/additions/doc/latest/pv-user-manual.pdf>

[20] Dassault Systèmes. (2021). Cameo Systems Modeler 2021x Refresh2 User Manual. Retrieved from <https://docs.nomagic.com/display/CSM2021x/2021x+Version+News>

[21] Dassault Systèmes. (2021). Cameo DataHub 2021x Refresh2 User Guide. Retrieved from <https://docs.nomagic.com/display/CDH2021x/User+Guide>

[22] IBM. (n.d.). Rational DOORS and Rational DOORS Web Access 9.6. Retrieved from <https://www.ibm.com/support/pages/rational-doors-and-rational-doors-web-access-96>

Coupling Optimization using Design Structure Matrices and Genetic Algorithm

Sébastien Dubé
Samares-Engineering, Blagnac
sebastien.dube@samares-
engineering.com

Mirna Ojeda
Samares-Engineering, Blagnac
mirna.ojeda@samares-
engineering.com

Jean-Marie Gauthier
IRT Saint Exupery Toulouse
jean-marie.gauthier@irt-
saintexupery.com

Keywords *MBSE, DSM, N2, Genetic Algorithm*

Abstract This article seeks to contribute to a nuanced understanding of the integration of Design Structure Matrix (DSM) and genetic algorithms in the context of Complex Systems modelling described within Model-Based System Engineering approach. By examining coupling minimization as a critical aspect of advanced systems engineering practices, we aim to provide a scholarly exploration, blending theoretical insights with practical applications. The objective is to equip systems architects with analytical tools integrated within their Model Based Systems Engineering (MBSE) environment for exploring the design space of component interactions, facilitating the identification of optimal system architectures.

I. Context and Motivations

Within the domain of Systems Engineering and Model-Based Systems Engineering (MBSE), the efficient management of complex systems necessitates a systematic approach to manage and refine the interactions among their functional components. A fundamental principle within Systems Engineering, emphasized by standards ISO15288:2023 [1] and its associated Systems Engineering Handbook [2], is the imperative to reduce coupling between subsystems for effective management of product complexity. The crux of coupling minimization involves the disentanglement of interdependencies between components.

In most industrial processes, Systems Engineering discipline involves applying Functional Analysis [3] approach and elicit the stakeholder needs using decomposition of system needs into functions and their associated data flow. Then, systems engineering methods propose to distribute these functions over the systems / components implementing the architectural solution. This requires an intermediate "logical architecture" that abstracts function closer to their eventual physical implementations, primarily guided by functional dependencies to ensure coherent integration.

In 2015 a DSM/N² matrix technique [4], has been used for extracting coupling metrics from Functional Architectures, with the capabilities of a genetic algorithm to minimize coupling among logical components. Our investigation addresses this task by employing Coupling Matrices, to assess and reorganize functional dependencies between logical components. The specific emphasis is on coupling minimization as an isolated criterion, accounting for factors such as allocation constraints and timing requirements. Consequently, our method initiates with the identification of functional dependencies, utilizing them to quantify coupling. Then, we propose an optimized allocation of functions to components.

In the subsequent sections, we present our approach consisting in coupling optimization through the application of a genetic algorithm. Following this, we present and discuss the results derived from the application of our approach on two case studies. Finally, we review the relevant literature concerning the application of DSM/N² diagrams in the context of Model-Based Systems Engineering (MBSE).

II. Background on Genetic Algorithm, N2 Matrices and Capella

In this section, we provide an overview of the integration of the DSM/N² matrix technique and genetic algorithm for coupling optimization.

a. Optimization of architecture using DSM/N² Matrices

The N² (N-squared) Matrices, also known as a Coupling Matrix or Design Structure Matrix (DSM), is a graphical representation used in Systems Engineering to analyse and visualize the relationships and dependencies between different components within a system. The primary objective of an N² Matrix is to assess and quantify the coupling or interdependencies between various elements, such as functions, subsystems, or components. The interdependencies within this matrix are defined by functional exchanges connecting two functions, with a binary representation: '1' indicating dependence and '0' signifying independence. Then, the method intends to propose a group of functions (Modules) by successively create groups where the number of interfaces between groups is minimized as illustrated in Figure 1. The sum of interactions outside the module constitutes total interactions.

To perform such grouping, at each iteration, method intends to compute the resulting “coupling” value, which is used to evaluate the resulting complexity of the proposed architecture.

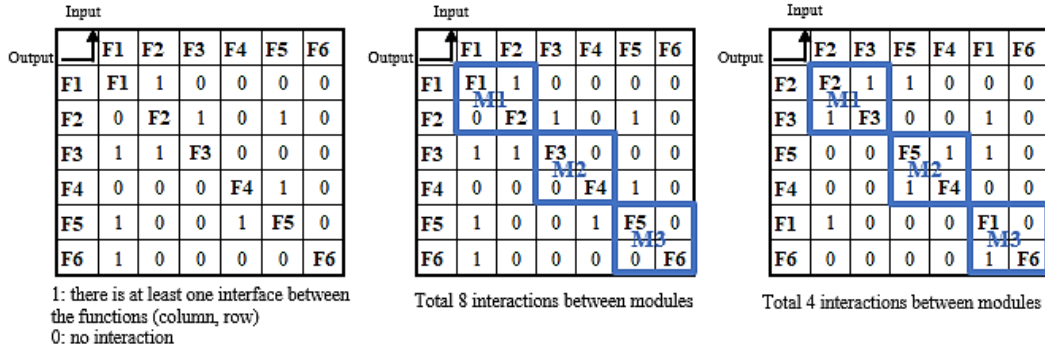


Figure 1. Illustration of the modularization concept and result after minimization of coupling

Using this matrix, we can calculate a coupling value pertaining to the interfaces specified between the logical components, deduced from the interfaces established among functions allocated to these components. The coupling value serves as an assessment of the complexity of coupling between logical components, derived from a formula based on software coupling metrics [5]. For this study, we used the Equation 1 to calculate the coupling value of each individual logical component. Then, the Equation 2 is used to calculate the coupling value of the complete architecture.

$$Coupling(C_{M_k}) = 1 - \frac{1}{d_i + 2 \cdot c_i + d_o + 2 \cdot c_o + \omega + r}$$

Equation 1 - Coupling Value of a Logical Component

$$CouplingValue(C_v) = \sum_{k=1}^n [C_{M_k}]$$

Equation 2 - Coupling Value of the Complete Logical Architecture

Where M_k is the logical component under consideration, d_i is the number of input data parameters, c_i is the number of input control parameters, d_o is the number of output data parameters, c_o is the number of output control parameters, ω is the number of modules called (fan-out), and r is the number of calling the module under consideration (fan-in).

b. Genetic Algorithm

Genetic algorithms, aim to explore the solution space of a given problem to meet predefined criteria. As depicted by the Figure 2, the algorithm initiates by randomly generating an initial population, with each subject representing a potential set of function allocations (1). Subsequently, a fitness function (2), in our case, the coupling equation, assesses each subject, assigning a value or rank that reflects its proximity to the optimal solution. Subjects too distant from the desired solution are then eliminated (3).

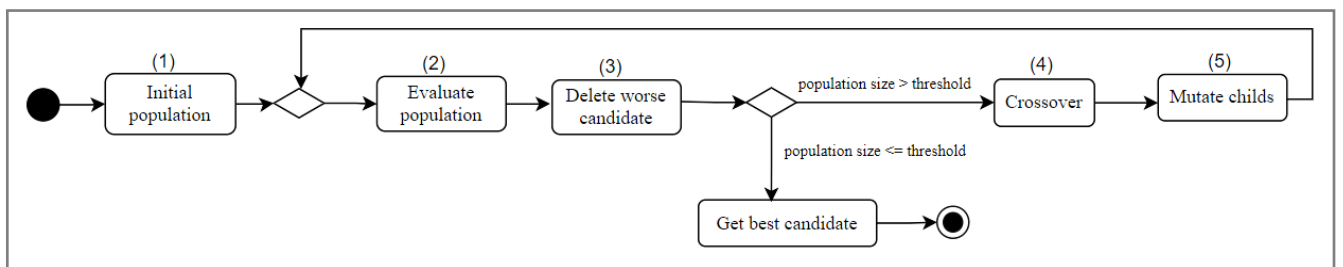


Figure 2. Genetic algorithm processes

The algorithm assesses the remaining subjects, and if the population size is within a specified threshold, it returns the best solution among them; otherwise, it proceeds. The pivotal biomimicry aspect of the genetic algorithm unfolds as the remaining subjects undergo crossover, exchanging genes to generate new subjects (4). These newly created offspring undergo mutation (5), where a portion of their characteristics undergoes random changes. Crossover and mutation serve the purpose of preventing convergence to local optima by dispersing new subjects throughout the solution space.

Genetic algorithms offer configurability through a set of parameters:

- Initial population size, a crucial parameter ensuring sufficient coverage of the solution space at the beginning.
- Max generation number, a parameter determining the algorithm's termination, even as the population grows.
- Percentage of survivors, indicating the proportion of the least fit subjects to be eliminated.
- Percentage of parents, denoting the proportion of subjects participating in crossover.
- Percentage of children to mutate, is the proportion of new subjects subjected to mutation after crossover.

- Percentage of genes to mutate, indicating the proportion of genes to be mutated for each new subject.

As the algorithm is iterative, the conditions for exiting the loop are defined as the number of chromosomes being less than 3 and reaching the maximum number of generations.

To implement programmatically the genetic algorithm, we have been established: the chromosome encoding is the logical component index, the index of chromosomes is the index of each function. The encoded chromosome is represented in 1-dimensional array format where each chromosome has an identifier. An illustration of the GA translation in code becoming in a single gene is shown in the Figure 3. In this example, there are 4 components and 6 functions, as illustration of the allocation mechanism: the function index 1 and 4 is allocated to the component index 2.

Chromosomes	1	2	3	4	2	3
Function identifier	0	1	2	3	4	5

Figure 3. Encoding chromosomes

c. Capella and Arcadia

The ARCADIA method [8] defines systems engineering concepts from needs analysis to architectural solution definition, we propose in Figure 4 the highlight of involved concepts in our current proposal:

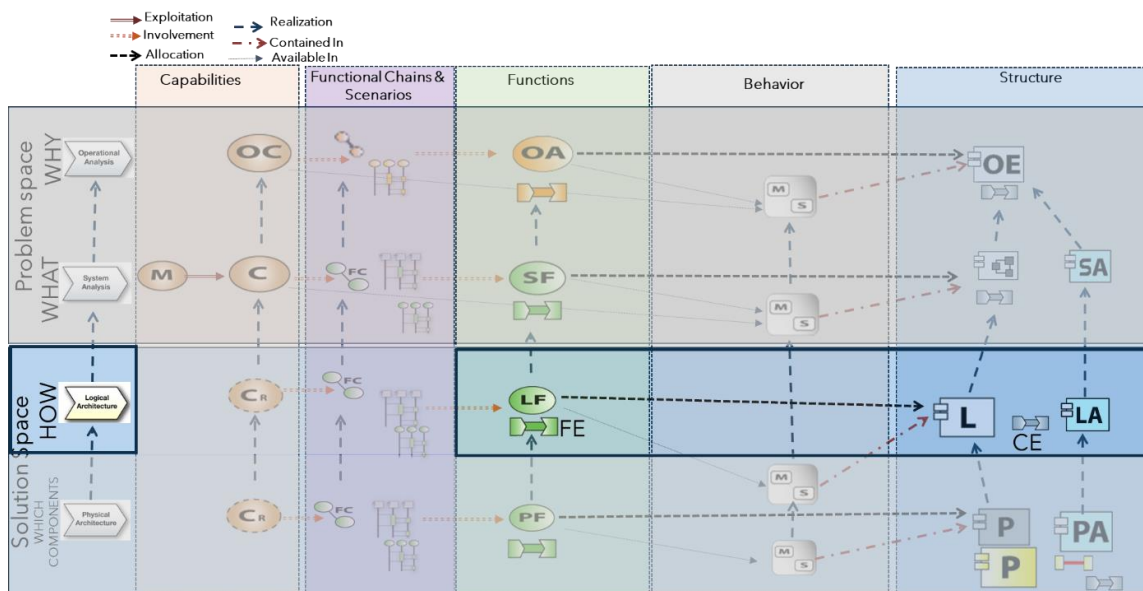


Figure 4. Overview of ARCADIA concepts and emphasis of involved concepts within our proposal

So, we consider in our current implementation the Logical Functions (*LF*) allocation to the Logical Components (*L*). The information exchange between functions is modelled using the concept of Functional Exchanges (*FE*). The Functional Exchanges (*FE*) are allocated to Component Exchanges (*CE*) when these exchanges cross the boundaries of a single component. The Component Exchanges (*CE*) represent the interfaces between components and the objective of our proposal consist of to minimize the number of cross-exchanges between components (and minimize interfaces coupling between systems).

An illustration of simplified logical architecture diagram in Capella is illustrated in Figure 5:

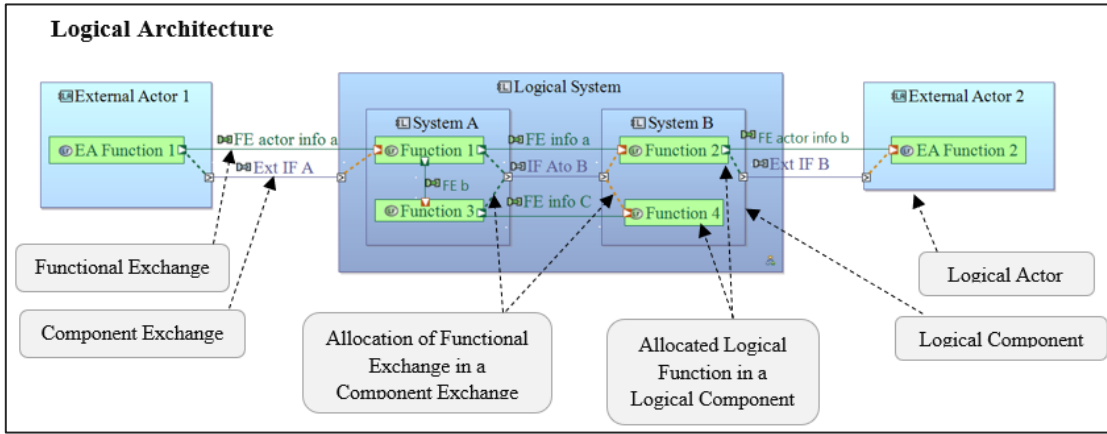


Figure 5. Simplified Logical Architecture Diagram example and associated concepts

III. Implementation and Case Study

In this section, we first present the implementation of our DSM Genetic Algorithm solution. Second, we present the application of our methodology through a case study. Finally, we present and discuss the obtained results concerning the efficiency and the implications of the proposed approach in the domain of Model-Based Systems Engineering.

a. Implementation in Python and Capella

The implementation of our DSM-based optimization has been integrated within different MBSE approaches and tools. This work relies on our previous work in [7] implemented within Cameo tool. This paper intends to focus on the integration of this technique within Capella MBSE tool [6] and offer to systems engineer facilities to evaluate the more optimized way to distribute functions in their logical architecture in order to minimize interfaces between the defined systems. This automation uses the Python4Capella extension [7], which provides a Python API to interface directly with the Capella model. The algorithm overview is presented below in Figure 6.

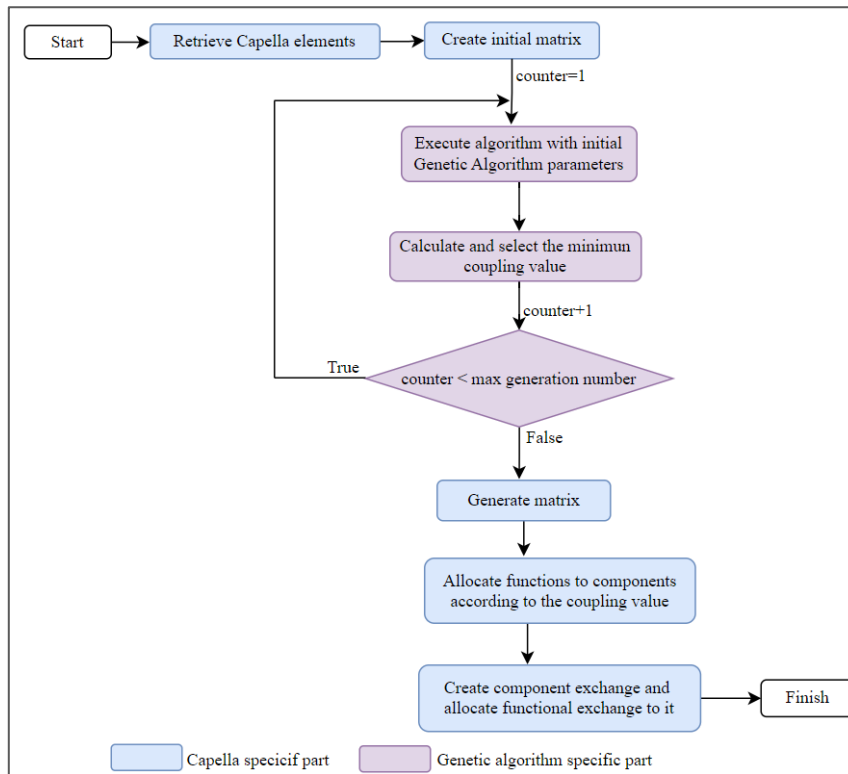


Figure 6. Algorithm structure and adaptation to Capella

Our process begins by employing Python4Capella's capabilities to extract the existing logical architecture. Then, using Python scripts, we generate the DSM to visualize and analyse the interdependencies and extract in excel format the resulting DSM Matrix.

The binary DSM is built based on the presence or absence of functional exchanges between functions. Once the DSM is established, the genetic algorithm, configured with initial parameters begins its optimization cycle. The initial parameters are based on our previous work in [9]. After identifying the allocation of functions to components, our tool automates the allocation process within the Capella model. It ensures that the allocations respect pre-defined constraints, such as pre-allocated functions as show in the Figure 7. available functions and are consistent with the lowest coupling values determined by the genetic algorithm.

The final step involves scripting Python4Capella to interpret the updated DSM and translate component interdependencies into Component Exchanges, in which the Functional Exchanges are allocated. This transformation is executed while filtering out relationships within the same component to maintain a focus on inter-component interactions.

The main differences between our current Capella implementation in regards to our initial implementation in Cameo/SysML environment are:

- In Capella, the algorithm considers the Functional Exchanges (*FE*) between all functions. In Capella/ARCADIA, some functions are assigned to the actors defined as external entities in interaction within the system of interest.
- In our initial implementation in Cameo tool, we have used in our functional architecture specialization for exchanges between functions by distinguishing **Flow of Information, Energy or Matter**. Then, we have created categories of Information Flow: **Data, Event, Enable/Disable** where “Events” are considered differently in the dependency flow. In our current Capella implementation, we have not yet considered the concept of “Exchanged Items” which may be used to distinguish Events and Shared Data.

In order to get the best solution under the parameters given, we observed that the number of functions to allocate is correlated with the initial population size, the maximum generation number and the gene mutation percentage. Indeed, the higher the number of functions to allocate, the bigger is the space to explore. Therefore, it might-be long process to get a suitable solution sufficiently optimized, especially when the number of candidate solutions grows. So, in further work, we plan to explore the way to minimize the time to obtain a coupling value minimized for large architectures, how to configure automatically initial parameters according to the complexity of architectures, and to study other optimization algorithms to find the best solution.

b. Application to AIDA Case Study

To assess the relevance of the proposed approach, we applied the optimization process to some case studies. The approach has been applied on an Aircraft Inspection by Drone Assistant system [10]. The AIDA system is a Remotely Piloted Aircraft System (RPAS) that it is composed of a quadcopter drone, a ground station system and a remote control.

i. Functions allocation constraints

In the reality of our system there are constraints on how functions should be allocated, influenced by factors such as safety or subcontracting, which is taken in our approach. Figure 7 illustrate this initial functions to components “allocations” constraints that should be respected while proposing a final and complete logical architecture. The figure shows that 15 of 23 logical functions have been allocated among 12 logical components.

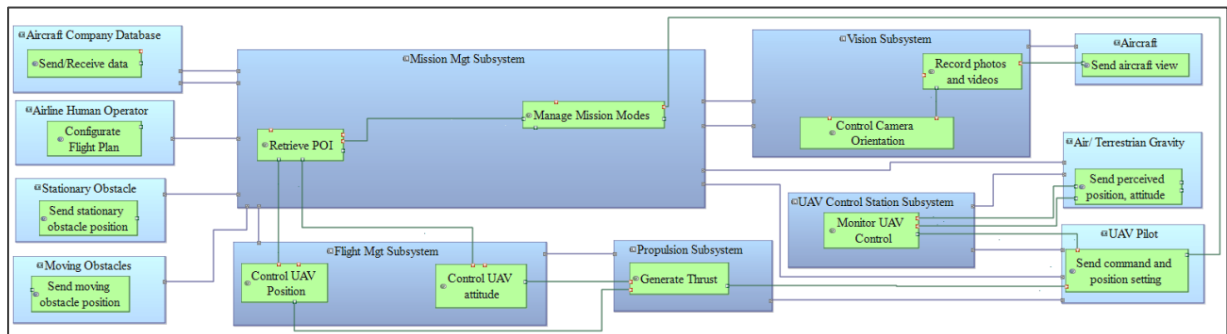


Figure 7. Architecture before algorithm execution

In order to get a suitable solution under the parameters given, we observed that the number of functions to allocate is correlated with the initial population size, the maximum generation number and the gene mutation percentage. Indeed, the higher the number of functions to allocate, the bigger is the space to explore. Therefore, it might-be long process to get close to a coupling value minimized, especially when the number of candidate solutions grows.

ii. Capture of relationships between logical components

The approach produces an initial matrix representing functions and their relationships, as depicted in Figure 8 (derived from the functional architecture and associated functional exchanges). Figure 9 illustrates the resulting matrix after the algorithm's execution, showcasing the functions allocated to 3 out of the 12 components in our use case. These matrices are generated in Excel format by our proposed implementation.

	A	B	C	E	F	G	H	L	N	O	P	Q	R	S	W
1	LogicalFunctionName	Index	0	2	3	4	5	9	11	12	13	14	15	16	20
2	Emergency Landing	0	F1	0	0	0	0	0	0	0	0	0	0	0	0
4	Sense and Avoid Obstacles	2	0	F3	0	0	0	0	0	0	0	0	0	0	0
5	Build FlightPlan Relative to Aircraft Type	3	0	0	F4	0	1	0	0	0	0	0	0	0	0
6	CheckWinForce	4	0	0	0	F5	0	0	1	0	0	0	0	0	0
7	Retrieve POI	5	0	0	0	0	F6	0	0	0	0	0	0	0	0
11	Identify Absolute Aircraft Coordinates	9	0	0	0	0	1	F10	0	0	0	0	0	0	0
13	Manage Mission Modes	11	1	0	0	0	1	0	F12	0	0	0	1	0	0
14	Send Pictures to DB	12	0	0	0	0	0	0	0	F13	0	0	0	0	0
15	Record photos and videos	13	0	0	0	0	0	0	0	0	F14	1	0	0	0
16	Control Camera Orientation	14	0	0	0	0	0	0	0	0	0	F15	0	0	0
17	Manage Photos Recording	15	0	0	0	0	0	0	0	1	1	1	F16	0	0
18	Configure Flight Plan	16	0	0	1	0	0	0	0	0	0	0	0	F17	0
20	Send aircraft view	20	0	0	0	0	0	0	0	0	1	0	0	0	F21

Figure 8. Initial matrix generated

	A	B	C	D	E	F	G	H	I	J	L	M	N	O	
1	LogicalFunctionName	Index	0	1	2	3	4	5	6	7	9	10	11	12	
2	Send aircraft view	0	F21	0	0	0	0	0	0	0	0	0	1	0	
3	Emergency Landing	1	0	F1	0	0	0	0	0	0	0	0	0	0	
4	Identify Absolute Aircraft Coordinates	2	0	0	F10	1	0	0	0	0	0	0	0	0	
5	Retrieve POI	3	0	0	0	F6	0	0	0	0	0	0	0	0	
6	Manage Mission Modes	4	0	1	0	1	F12	0	0	0	1	0	0	0	
7	Build FlightPlan Relative to Aircraft Type	5	0	0	0	1	0	F4	0	0	0	0	0	0	
8	Sense and Avoid Obstacles	6	0	0	0	0	0	0	F3	0	0	0	0	0	
9	CheckWinForce	7	0	0	0	0	1	0	0	F5	0	0	0	0	
11	Manage Photos Recording	9	0	0	0	0	0	0	0	0	F16	1	1	1	
12	Control Camera Orientation	10	0	0	0	0	0	0	0	0	0	F15	0	0	
13	Record photos and videos	11	0	0	0	0	0	0	0	0	0	1	F14	0	
14	Send Pictures to DB	12	0	0	0	0	0	0	0	0	0	0	0	F13	
	LogicalComponentName		Aircraft	Mission Mgt Subsystem							Vision Subsystem				

Component Function

Figure 9. Matrix generated after algorithm execution

iii. Resulting Architecture

The resulting architecture is shown in Figure 10, encompassing the analysis and allocation of the lowest level of the logical component and function. While the current approach proposes a single logical architecture, in case of multiple configurations with the same minimal coupling value exist, one of the relevant solution is selected by our tool. We may propose evolution to generate various architecture alternatives and support trade-off analyses in future work.

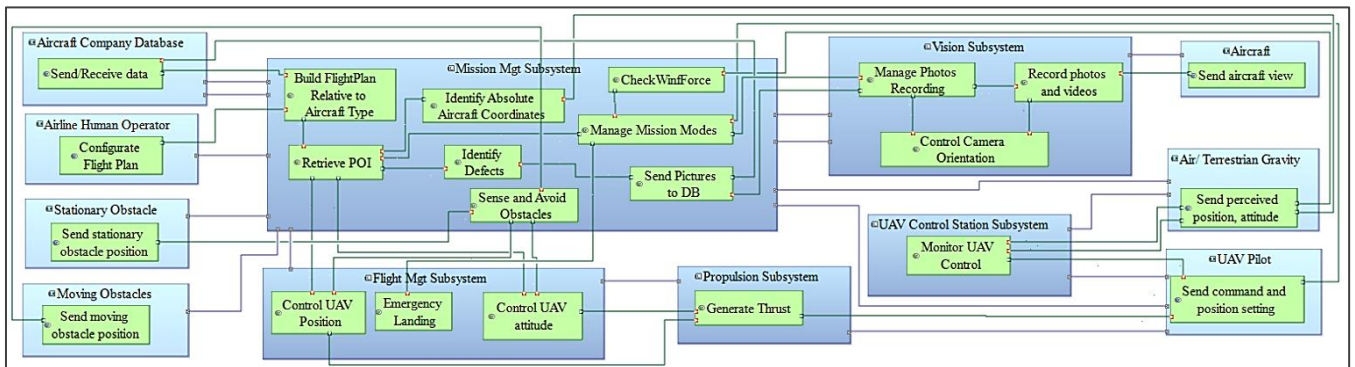


Figure 10. Allocation result after algorithm execution

iv. Genetic Algorithm Parameters Tuning

The setting of the genetic algorithm parameters has been performed currently by iterative experiments. In Table 1., we highlight some experiments we made on AIDA simplified model and a larger model. The genetic parameters have been

set accordingly to obtain some viable solutions and keep a reasonable execution time based on the previous work [9]. The approach has been also executed over another Capella model, AIDA large model with a larger parameter.

The genetic algorithm parameters can be modified in the Capella tool via user interface. During the experiment, it was observed that the population size increase with the matrix size, for matrix 23x23 from 200 population size give a coupling value minimized, the survivor percentage have not significant impact less than 90%, else can stay in the loop , the population mutation and the gene mutation have similar impact , the population mutation until 50% and gene mutation <20% centralize function allocation in few logical components and high percentage (>40%) can be out of solution space remaining in the loop.

Model Name	# Functions	# Components	#Pre-Allocated Functions	Execution time (seconds)	GA Parameters
AIDA light model	23	12	15	1,45	<ul style="list-style-type: none"> Initial population=1000 Maximum generation=50 Survivor (crossover) percentage=70% Parent percentage=20% Population mutation percentage=70% Gene mutation percentage=30%
AIDA large model	47	17	7	120	<ul style="list-style-type: none"> Initial population=2500 Maximum generation=200 Survivor (crossover) percentage=70% Parent percentage=30% Population mutation percentage=40% Gene mutation percentage=70%

Table 1. Execution time of the algorithm and genetic algorithm parameters

v. Comparison to our initial implementation in Cameo

The Table 2. shows comparison results between Capella and Cameo tool for the similar model scope. We can notice that coupling value is larger with Capella model in regards to Cameo SysML model.

Model name	Coupling value	# Interactions
AIDA (Cameo tool)	3.0	23
AIDA light model (Capella)	8.7	21

Table 2. Comparative values in different tool implementation

This can be explained for different reasons. In our initial implementation in the Cameo tool, the model has not pre-allocated functions to external actors, which means they are not included in the analysis. In the other hand, in Capella tool implementation, the external actors have pre-allocated functions, and they are included in the analysis, leading to discrepancies in the coupling values. External actors are considered as constraint for the algorithm in the analysis process, but they are not available for function allocation. The functions available for allocation can only be allocated to the logical components. The number of transitions refer to the interaction between functions allocated in different components of the system

IV. Current Status, Future work, and Perspectives

After reviewing with different parties, the usage of such tool may bring interest when there is still flexibility to distribute functions over the systems, which might be difficult when reusing legacy products or when system architecture is highly constrained by organizational aspects. The generic algorithm is also able to consider already allocated functions as constraints for the space exploration.

Our contribution is currently in the initial stage to give the ability to systems engineer to find a ready to use in Capella MBSE use case. As future work we have identified several possibilities to extend this proposal to:

- Provide implementation of DSM generation within SysML Tools as open-source solution
- Extend the concept to introduce consideration of timing constraints and extend the functions and component exchanges with a time delay property and ensure as a constraint that Time budget allocated to the overall functional chain are fulfilled
- Explore other algorithms than Genetic Algorithm and use optimization techniques proposed in related works to handle large matrices.

- Explore the possibility to generate alternatives of architectures in a same model and exhibit the associated properties (timing, performance, costs, ...) of each. Then extend this with multi-dimensional optimization techniques.

V. Related Works

DSM/N2 matrices are known tools to analyse the architecture complexity of systems and there are existing work which extract DSM/N2 matrices from MBSE methods & tools. A first proposal in [11] exhibits how to extract N2 matrix from model of processes developed using OPM method [12]. In [13], the authors propose a contribution compatible with different MBSE tools using the standardized XMI format. Using DSM and genetic algorithm has been investigated in [14]. In this paper, the authors propose a genetic algorithm tailored for problems characterized by modularity, hierarchy, and overlap within complex systems. In [16], the authors proposed to modify DSM's clustering algorithms to include several design constraints. Robert et al. [17] introduce an approach to DSM clustering, addressing shortcomings in existing clustering techniques by encompassing data acquisition and handling multiple perspectives to a post-processing phase that corrects results for technical feasibility. Finally, in [17], authors proposed and optimized fast clustering algorithm using genetic algorithm to generate DSM matrices for complex system architectures. However, this approach is performed on physical architecture level and is presented as a MATLAB script to support optimization of complex multi-physical systems architecture (with a high number of physical interfaces).

VI. Conclusion

The specificity of our approach is to propose an optimization algorithm of DSM where the clustering algorithm is influenced by the function's allocation to Logical Architecture and is implemented as a genetic algorithm integrated within common MBSE tools (Cameo Systems Modeler, Capella). This allows to address system representation with intermediate complexity using the Logical Architecture concept proposed by several MBSE methodologies such as ARCADIA. As an achievement, we have published our proposal as a Capella addon named **DSM4Capella** in Capella Community within Labs4Capella [18].

Bibliography

- [1] I. S. Organization, ISO/IEC/IEEE 15288:2023 Systems and software engineering System life cycle processes.
- [2] INCOSE, Systems Engineering Handbook - V5, 2023.
- [3] V. Kumar, «Functional Analysis in Systems Engineering: Methodology and Applications,» *JETIR*, vol. 2, n°110, 2015.
- [4] T. R. Browning, «Design structure matrix extensions and innovations: a survey and new opportunities,» *IEEE Transactions on engineering management*, pp. 27-52, 2015.
- [5] H. Dhama, «Quantitative models of cohesion and coupling in software,» *Journal of Systems and Software*, vol. 29, pp. 65-74, 1995.
- [6] «Capella - Open source solution for Model-Based Systems Engineering,» [En ligne]. Available: <https://mbse-capella.org/>. [Accès le 17 11 2023].
- [7] «Python4Capella,» MBSE Capella, [En ligne]. Available: <https://github.com/labs4capella/python4capella>. [Accès le 17 11 2023].
- [8] J.-L. Voirin, Model-based system and architecture engineering with the arcadia method, Kildington, Oxford: Elsevier, 2018.
- [9] Y. Khetan, J.-M. Gauthier et S. Dubé, «Part 5 – Coupling optimization of logical architecture using genetic algorithm,» June 2020. [En ligne]. Available: <https://www.samares-engineering.com/en/2020/07/>. [Accès le 17 11 2023].
- [10] IRT St Exupery, «AIDA architecture,» [En ligne]. Available: <https://sahara.irt-saintexupery.com/AIDA/AIDAArchitecture>. [Accès le 08 03 2024].
- [11] A. Sharon, O. L. De Weck et D. Dori, «Model-Based Design Structure Matrix: Deriving a DSM from an Object-Process Model,» *Wiley Online Library*, 2012.

- [12] D. Dori, *Model-Based Systems Engineering with OPM and SysML*, Springer, 2016.
- [13] S. K. Salas Cordero, C. Fortin et R. Vingerhoeds, «Concurrent Conceptual Design Sequencing for MBSE of Complex Systems through Design Structure Matrices,» chez *International Design Conference*, 2020.
- [14] T.-L. Yu, D. E. Goldberg, K. Sastry, C. F. Lima et M. Pelikan, «Dependency structure matrix, genetic algorithms, and effective recombination,» *Evolutionary computation*, pp. 595-626, 2009.
- [15] F. Borjesson et U. Sellgren, «FAST HYBRID GENETIC CLUSTERING ALGORITHM FOR DESIGN STRUCTURE MATRIX,» chez *ASME 2013 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, 2013.
- [16] K. Sinha, S.-Y. Han et E. S. Suh, «Design structure matrix-based modularization approach for complex systems with multiple design constraints,» *Systems Engineering*, pp. 211-220, 2020.
- [17] H. Robert, Y. Ali et M. Christoph, «Systematic module and interface definition using component design structure matrix,» *Journal of Engineering Design*, pp. 647-675, 2010.
- [18] «Labs4Capella,» [En ligne]. Available: <https://labs4capella.github.io/>. [Accès le 17 11 2023].

Specializing SysMLv2 for Real-Time Safety-Critical Systems – an Experiment with AADL

Pierre Dissaux, Jerome Hugues
pierre.dissaux@ellidiss.com, jhugues@andrew.cmu.edu

Abstract – The future release of OMG SysMLv2 provides a new set of foundational layers to support engineering activities of a large set of systems. SysMLv2 relies on a restricted set of concepts combined with a large library to define building blocks for designing systems. This approach makes it possible to define domain-specific libraries that enrich or specialize SysMLv2 elements. In this paper, the authors show how to build one such specialization for real-time safety-critical systems. Starting from the SAE AADL language elements, we show how to a) extend SysMLv2 constructs with AADL ones, and b) propose guidelines to represent AADL static and dynamic semantics. This development serves as an illustration of SysMLv2 extension capabilities. It also addresses a recurring concern of specializing MBSE for domain-specific engineering activities, ranging from design activities to V&V.

Index Terms – Model-Based Systems Engineering, Embedded Systems, SysMLv2, AADL.

INTRODUCTION

The engineering of software-intensive systems such as embedded systems is a complex endeavor whose scope extends well-beyond software engineering to incorporate systems engineering, safety, or security to name a few. Model-Based Engineering provides foundations to tackle this diversity of concerns through well-defined abstractions to capture system constituents, combined with model transformations e.g., to perform analysis. It is not uncommon to combine multiple modeling notation, picked among a rich portfolio of languages.

In the following, we focus on the engineering of real-time safety-critical embedded systems. A common denominator of these approaches is the need to capture system requirements, system architectures up to some level of details and then further developed. In this context, the OMG SysML v1 [1] standard usually plays a role to build the first abstract set of models covering functional and logical architectures, whereas domain-specific languages like SAE AADL [2] or OMG MARTE allows one to refine a logical architecture expressed in SysML into a physical one in AADL [3].

Yet, these approaches are usually inefficient in that a) there is an overlap between these models: they may share similar interface or component definitions, b) they force the usage of different languages, adding pressure on project cost and schedule.

The future release of SysML v2¹ brings a set of new attractive features that deserve to be analyzed to improve on the current state-of-practice. The most interesting ones are the ability to define domain libraries as a replacement to UML profiles along with a rich textual and graphical syntax that provides a unified type system, regular definitions of model elements based on the notions of definition and usage, along with user-extensible semantics for, e.g., port-based communication or automata description.

This paper is organized as follows: first, we recall the need for combining SysMLv1 with another notation. The authors selected AADL, but this would equally apply to OMG MARTE or AutoSAR. We illustrate some of the existing transformation approaches and their limits. Then, we introduce some of the features of the future SysMLv2 standard and how they can be leveraged to build domain-specific libraries. Taking the SAE AADL language as an input, we show how to build a library for extending SysMLv2 for the engineering of embedded systems and provide an overview of the foreseen tool support. An illustrative example is provided at the end of the paper and we conclude by introducing a current initiative to draft one such library in the scope of the OMG Systems Modeling Community.

STANDARDS FOR MODEL-BASED ENGINEERING

The development of SysMLv1 started in 2004, in coordination with INCOSE, with the goal of providing a model-based solutions to support the ISO15288 Systems Engineering technical processes, ranging from Stakeholder Needs to Requirements, Architectural Design down to Verification and Validation processes. The standard has been revised several times and release 1.7 has been published in December 2022.

The AADL standardization committee has been established in 1999. This committee is hosted by SAE International, that published multiple standards for the avionics community. The first release of AADL has been published in 2004, with the

¹ Draft document available at <https://www.omg.org/spec/SysML/>

latest revision published in April 2022. AADL has been designed to be a formal language with a rich set of semantics, to allow for unambiguous and precise modeling of embedded systems architectures that encompass the software design and runtime architecture, the physical system, and the computing hardware.

In both cases, developing an open standard has been a critical decision to foster an international community of researchers, tool vendors, and industrial users. There is a natural overlap between SysMLv1 and AADL: both support architectural design modeling. Yet, they do so at different levels of abstractions.

1. SysMLv1 considers mostly logical (abstract) system architectures and connects elements of a system to requirements and abstract behaviors. SysML models rely on naming conventions to infer the role of blocks (such as router, bus, etc.) and provide limited mathematically grounded analyses capabilities.
2. AADL has native concepts for describing both the interface of components (devices, buses, etc.), but also to configure these elements (using default property sets) or through AADL annexes and define their run-time semantics.

For instance, AADL models can be analyzed for performance or safety analysis, whereas SysMLv1 models remain mostly descriptive and could be used to drive external simulation or analysis capabilities.

Complex systems require both the descriptive approach supported by SysML and the analytical one supported by AADL. The descriptive paradigm allows for the realization of a “single source of truth” that can be queried, whereas the analytical one allows for automated analysis capabilities, bringing confidence that the system will deliver the right set of performance. From a modeling perspective, an AADL model can be seen as a refinement of a SysMLv1 model: it provides a more accurate description of the system while using similar elements such as interfaces or components topology.

The most challenging issue in such model transformations is the preservation of the semantics between the two languages. This can be achieved by expressing the target model semantics in at least three ways:

1. Within the source model via a metamodel extension (e.g., UML stereotypes).
2. Within the source model via a library of specialized constructs (e.g., SysMLv2 domain libraries).
3. With an independent third-party tool that implements the semantical transformation rules (e.g., ATL, LMP).

The pros and cons of these three approaches can be summarized as follows:

	Pros	Cons
Source metamodel extension	- Robust mapping that can be verified at the most abstract level	- High impact on modeling tools implementation. - Lack of portability often observed
Source model library	- Easily changed and sharable - Low impact on modeling tools implementation	- Risk of lack of mapping standardization
Independent processing rules	- No extension required at metamodel or model levels - Low impact on modeling tools implementation	- Third-party tool and language required

Following the first approach (UML stereotypes), Galois [4] has developed a bi-directional translator between SysMLv1 and AADL to support the navigation between the two model spaces teams. Still, this means two sets of models that must be maintained in parallel, with adequate training for the engineering teams. This adds extra costs and burden on engineering teams.

On the contrary, Ellidiss applied the third approach to implement a SysMLv1 to AADL transformation process using the LMP technology [7], where the mapping between the two languages is formalized by Prolog rules. However, due to the lack of standardized guidelines, this solution requires a high level of customization at corporate or project level.

SysMLv2 now brings an opportunity to experiment the second approach with the definition of a domain library dedicated to AADL.

ABOUT SYSMLV2

SysMLv2 is the successor of the Systems Modeling Language (SysML) series of languages. The “v2” is an important factor in that SysMLv2 is not a simple revision but a complete redesign of the language. SysMLv2 and is not built as an extension/tailoring of UML like SysMLv1. Instead, SysMLv2 is built around a metamodel and model libraries. The

preferred approach to extend the language is to create model libraries. What is the most important is that the language semantics is grounded in mathematical and ontological terms. These should not frighten the end user: it means that the language has been carefully thought to be consistent from the ground up. Although the language is still under finalization, one does not expect the language to change significantly at this stage. Also, it has been heavily analyzed and presented [4, 5]. The general feedback is that SysMLv2 has a set of features that will enable new approaches to support model-based engineering at large:

- Simpler set of language constructs built around the concept of “definition” and “usage”. This separation provides the required rigor to express the various dimensions of a system lifecycle.
- SysMLv2 is built on top of KerML, a small foundational set of concepts such as “occurrences” or “thing”. These ontological definitions bring clarity compared to a collection of stereotypes.
- SysMLv2 provides a standardized API to manage model storage, exchange, and interaction. This allows a strong decoupling between configuration management, model authoring, and model processing.
- SysMLv2 provides both graphical and textual syntax. The latter allows for a leaner approach to modeling.

As of Summer 2023, more than 14 SysMLv2 tools are under development, covering the typical landscape of modeling tools. The textual syntax allowed for the development of plugins for IDE such as VSCode, lowering the barrier to start building models.

ABOUT AADL

The AADL [2] language is an architecture description language for model-based engineering of embedded real-time systems. It defines notations, expressed using both a graphical and a textual syntax, to represent a full system with its software and hardware components in one model (architectural representation of the system).

The AADL components are defined by a type (that mainly contains the component interface elements, called features) and zero or more implementations (that present the component internal structure composed of subcomponents, calls, connections, flows, modes, and properties). The components are grouped in three categories: software components (subprogram, subprogram group, data, thread, thread group and process); hardware components (processor, virtual processor, device, bus, virtual bus, and memory); and system composition component (system). An AADL connection is a linkage established between components that can be port, parameter, or access connections. The AADL language brings

the capability to enrich the model with additional information by a set of standard properties and annexes. Properties are used to complete the component definition and bind the whole system hierarchically.

There is a notional correspondence between AADL concepts and SysMLv2 ones. Component types and implementations are like SysMLv2 parts: they describe the constituents of a system. Similarly, AADL concepts of ports and connections echo SysMLv2 concepts of features and connections, AADL properties and SysMLv2 attributes share the same goal.

However, AADL provides a constrained set of semantics rules for modeling safety-critical systems:

- AADL component categories are representative of building blocks of these systems. They are associated with legality rules that constrain composition of components.
- AADL components have predefined semantics, e.g., for periodic threads, for system startup or model change that prescribes the system lifecycle. Similarly, port-based communication is fully specified in terms of expected observable behavior.
- AADL property sets are representative of the configuration parameters for these systems.

Hence, it is natural to consider AADL as a specialization in which AADL concepts specialize SysMLv2 ones.

SPECIALIZING SYSMLV2 FOR REAL-TIME SAFETY-CRITICAL SYSTEMS – AN OVERVIEW

In the following, we sketch our current work in defining elements to specialize SysMLv2 with AADL concepts.

Our approach aims to address three requirements:

1. Support AADL vocabulary by specializing SysMLv2 concepts to define AADL ones.
2. Ensure that SysMLv2 models respects AADL static semantics, such as typing rules, e.g., a thread can be a subcomponent of a process, but not vice-versa. AADL defines 200+ rules to define the notion of model validity.
3. Support for AADL dynamic semantics. Whenever possible, we want to specialize SysMLv2 semantics to describe the execution semantics of a model. Let us note this would constitute a nice addition to the original AADL specifications: currently, the AADL semantics is formally described as a collection of hybrid automaton and English text. Leveraging SysMLv2 state machines would make the description less ambiguous and machine processable.

Figure 1 illustrates how to support the first requirement: AADL component categories are defined using corresponding SysMLv2 part specialization. From the end user perspective, they have access to the original concepts of the language defined in a library imported (see “import AADL::*”).

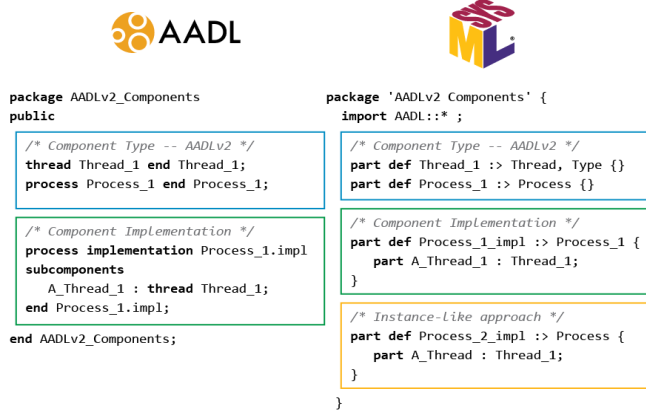


Figure 1 AADL and SysMLv2 with AADL concepts.

The library itself defines each specialization and associate rules to these specializations, covering the second requirement (see Figure 2). Each AADL component category specializes a common ancestor that specializes SysMLv2 part. This specialization specifies static constraints to be respected. In this example, we state which elements can be subparts (subcomponents in AADL vocabulary) and which features can be used (e.g. a process may not use hardware features). SysMLv2 makes it easy to specify those constraints using iterators and basic logic. We note this modeling approach is equivalent to the definition of a meta-model. The library model used by SysMLv2 allows one to hide the way constraints are defined, or the inheritance tree.

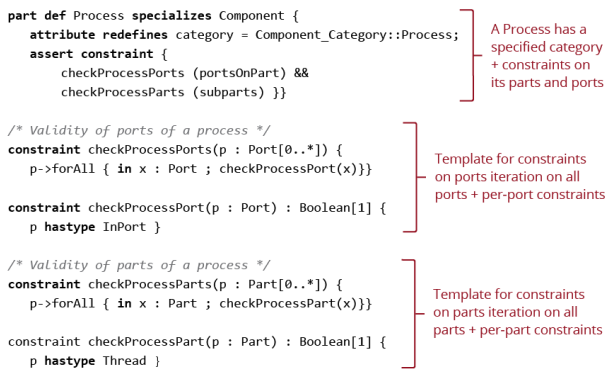


Figure 2 AADL Process as a SysMLv2 part specialization.

An alternative approach that addresses the second requirement with a more complete specialization hierarchy was also explored in [6]. It has the advantage of being easy to check by a SysMLv2 tool, but does not cover some specific

cases (e.g., for assessing the compatibility of connections), Constraints addresses this issue, but makes the validation of a model more costly, as constraints must be evaluated. Currently, there is no available implementation of SysMLv2 that support either their evaluation on-the-fly or on-demand.

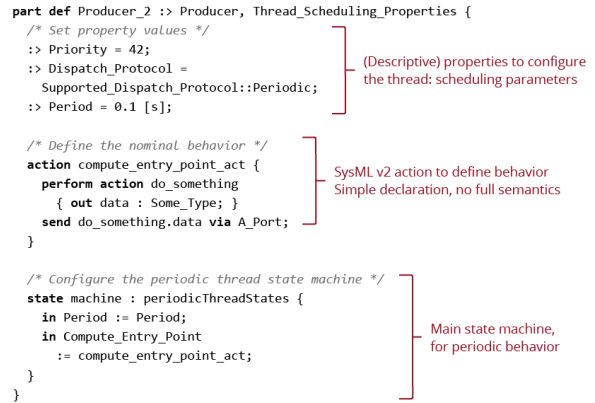


Figure 3 AADL process with execution semantics.

Finally, figure 3 shows a more elaborated example to cover the third requirement. We defined a collection of attributes to capture some configuration parameters for scheduling. We also instantiate a state machine to represent the expected periodic behavior of this entity. Similar specializations have been defined for port-based communication or other scheduling policies.

DETAILS OF THE AADL/SYSMLV2 MAPPING

The strategy that has been chosen to describe in detail the representation of AADL constructs in SysMLv2 is to follow the structure of the AADL standard document (SAE AS-5506D). Indeed, one of the short-term objectives of this work is to make it as easy as possible for AADL users to use the new language.

The AADL standard document is organized as follows:

- Chapters 1 to 3 provide informal description text about the scope, references, and a summary of the language.
- Chapter 4 gives a more formal description of the main structural elements of the language, i.e., packages, components, and annexes.
- Chapters 5 to 7 describe each specific category of component.
- Chapters 8 to 10 address components interactions in terms of features, connections, and flows.
- Chapter 11 contains the definition of the AADL properties language.
- Chapters 12 to 14 deal with the elaboration of a complete operational model and support of layered architectures.

- Chapter 15 is dedicated to the textual language lexical rules.
- Appendix A contains the complete definition of the predefined AADL properties that apply to the various AADL constructs.
- Appendix B to F provide summaries of terms, textual syntax, graphical notation, and other ancillary information about the language.
- Annexes A to F are extensions to the core language that are specified in separate documents. They address more specific topics such as data modeling and code generation, interaction with the ARINC 653 and FACE standards, as well as real-time behavior and safety sub-languages.

The ultimate goal of the AADL/SysMLv2 mapping is to cover all the topics addressed by the SAE documents that are relevant to enable operational use of SysMLv2 to perform AADL modeling and verification activities.

This initial version of the mapping addresses chapters 4 to 11 only. The current state of this work is described below in this paper. Additional work will be required to address chapters 12 to 14 and the annexes. Chapter 15 must also be addressed for the management of AADL and SysMLv2 reserved words and their difference in case sensitivity.

MAPPING RULES FOR AADL STANDARD CHAPTER 4

The first part of chapter 4 of the AADL standard document (4.1 and 4.2) deals with high level organization of a model. An AADL specification is the collection of AADL packages and AADL property sets that contribute to the definition of all the required modeling elements. The AADL specification defines a global namespace, and each package or property set defines its own namespace. Access to definitions of remote packages may be insured thanks to a with statement that can be optionally enforced by a renames statement to avoid the use of full qualified names. An AADL package may optionally define a private section.

Corresponding constructs in SysMLv2 are very similar. A SysMLv2 root namespace defines a global namespace that owns packages. Each SysMLv2 package defines a new namespace. Access to remote declarations can be obtained with an import statement, that automatically allows for the use of short names instead of qualified names. Local renaming is supported by the definition of aliases. There is no dedicated notion of property set in SysMLv2; as explained below, specially featured SysMLv2 packages will be used instead. We can also note that both AADL and SysMLv2 share the same syntax for defining qualified names.

AADL	SysMLv2
<i>specification</i>	<i>root namespace</i>
<i>package</i>	<i>package</i>
<i>public/private sections</i>	<i>public/private elements</i>

<i>property set</i>	<i>package</i> (see chapter 11)
<i>with/renames</i>	<i>import/alias</i>
<i>:: sep. for qualified names</i>	<i>:: sep. for qualified names</i>

Table 1 mapping for packages.

The second part of chapter 4 of the AADL standard document (4.3, 4.4, 4.5, and 4.6) is related to the AADL component model. An AADL component definition is done within an AADL package and can be split into a mandatory component type declaring its externally visible features, and zero or several component implementation(s) containing the internal realization(s), and in particular a list of subcomponents. This section also contains the definition of a particular kind of AADL components, referred to as abstract. An AADL abstract component is a generic placeholder used in early phases of the design process, that must be later refined into one of the concrete categories of AADL components.

The concept of type and implementation does not exist in SysMLv2. Two possible mapping options can be considered depending on the intent of the modeling process. The first option consists in ignoring this two-fold definition and including all the owned elements inside a single SysMLv2 part definition. Necessary split or merge operations can be supported respectively by SysMLv2 to AADL and AADL to SysMLv2 model transformation rules. The second option supposes the declaration of one part definition for the component type and as many additional part definitions as there are component implementations. The part definitions representing the component implementations must then specialize the one representing the component type. The AADL legality rules specifying what a component type and a component implementation may contain can then be managed by two abstract part definitions owning the relevant SysMLv2 constraints, one for the component types and the other for the implementations. The concept of AADL abstract component is easy to manage in SysMLv2. Indeed, it could be directly mapped to the native SysMLv2 part definition construct. In practice, it will be processed in the same way as the other AADL component categories in the SysMLv2 domain library.

AADL	SysMLv2
<i>component type</i>	<i>abstract part definition</i>
<i>component implementation</i>	<i>abstract part definition</i>
<i>subcomponent</i>	<i>part usage</i>
<i>abstract component</i>	<i>part definition</i>

Table 2 mapping for components.

The last part of chapter 4 of the AADL standard document (4.7 and 4.8) is about component parametrization and language extensions. AADL prototypes are used to define parametric components, i.e. components having some of their features that can be customized while being instantiated as subcomponents. AADL annexes are the way to extend the AADL language definition under the form of guidelines,

libraries, or sublanguages. Some of these annexes have been included into the SAE standard package whereas others are user or tool specific, such as Resolute, Agree or LAMP.

SysMLv2 does not provide a dedicated construct to support parametric part definitions highlighting which elements can be customized. An almost similar result can be obtained using general SysMLv2 specialization and redefinition mechanisms. Regarding AADL annexes, the mapping will be specified on a case-by-case basis. Although these have not been explored in detail yet, we can expect that the AADL behavior annex can leverage native SysMLv2 constructs such as states, actions and the KerML expression language. The AADL error annex may better rely on the future adaptation to SysMLv2 of the existing OMG RAAML standard. User defined annex sublanguages can be embedded inside a SysMLv2 model thanks to textual representations. Finally, annex libraries can be represented by SysMLv2 library packages.

AADL	SysMLv2
<i>prototype</i>	<i>part usage redefinition</i>
<i>behavior annex</i>	<i>states, actions, expressions</i>
<i>error annex</i>	<i>RAAML ?</i>
<i>other annex sublanguages</i>	<i>textual representation</i>
<i>annex libraries</i>	<i>library package</i>

Table 3 mapping for prototypes and annexes.

MAPPING RULES FOR AADL STANDARD CHAPTERS 5 TO 7

Chapters 5 to 7 of the AADL standard document provide a detailed specification of each category of component. This includes software components (chapter 5), execution platform components (chapter 6), and system composition (chapter 7).

For each category, the AADL specification consists of informative text, a set of more formal naming, legality, and consistency rules, as well as the list of relevant AADL properties. All this information contributes to the definition of the static and dynamic semantics of the AADL language and makes it appropriate for the support of real-time embedded critical system and software development.

The first step of this work aiming at representing an AADL specification with a corresponding SysMLv2 model, focuses on the selection of the most appropriate SysMLv2 construct or combination of construct for each AADL one. At that stage, we do not address in depth the definition of all the specific AADL static and dynamic semantics under the form of SysMLv2 constraints or metadata for instance. That's why, in this paper, we will not perform a precise analysis of each individual category of AADL component and its SysMLv2 counterpart. Instead, we present a general strategy to support the generic concept of AADL category that does not exist explicitly in SysMLv2.

The currently proposed solution is to add to each component a dedicated attribute typed by an enumeration listing all the possible AADL category names. Most of the AADL static semantics rules can be expressed in terms of SysMLv2 constraints. This is in particular the case for the rules defining which categories of features and subcomponents can be embedded in each kind of AADL component.

AADL	SysMLv2
<i>component category</i>	<i>abstract part definition with a category attribute usage</i>
<i>list of categories</i>	<i>enum definition</i>
<i>allowed feature</i>	<i>constraint</i>
<i>allowed subcomponent</i>	<i>constraint</i>

Table 4 mapping for categories

Note that there are still ongoing discussions with the SysMLv2 standardization committee to check other possible mapping options requiring some clarifications about the definition of the semantics of the language.

MAPPING RULES FOR AADL STANDARD CHAPTERS 8 TO 10

Chapters 8 to 10 of the AADL standard document specify in order component features, connections between components, and logical dataflows across one or several components. The last item (chapter 10) has not been studied yet.

AADL features represent interaction points for a component. As such, they are part of an AADL component type. They can be either an AADL abstract feature, an AADL port, or an access point to a remote shared AADL subcomponent. AADL also specifies feature groups that encompass a collection of features. AADL connections are defined for each kind of feature.

The general way to represent AADL features with SysMLv2 is the port. However, it must be enriched by additional constructs to better match either AADL ports or access features. Indeed, an AADL port is typed by a specific data component which can be represented by an owned item of a SysMLv2 port. In a similar way, an AADL access feature requires a binding between a SysMLv2 port and a ref usage representing a proxy of the remote component. Feature groups can be managed natively with SysMLv2 port nesting. Finally, AADL connections can be represented in a general case by SysMLv2 connections.

AADL	SysMLv2
<i>port and abstract feature</i>	<i>port with a type item</i>
<i>provided access feature</i>	<i>out port + ref + bind</i>
<i>required access feature</i>	<i>in port + ref + bind</i>
<i>feature group</i>	<i>nested ports</i>
<i>connection</i>	<i>connections</i>

Table 5 mapping for features and connections.

Chapter 11 of the AADL standard document deals with the property sublanguage. Properties are one of the ways to enforce the semantics of each category of AADL elements. They are also commonly used as a language extension mechanism. Finally, they have sometimes been overused to implement constructs that would have probably been better described at architectural level. This is typically the case for binding, entry points, and modeling properties. In the following, we will distinguish the mapping for the general case, and the specific mapping for binding properties. The other modeling properties have not been studied in depth yet.

AADL properties are declared in property sets containing lists of property types, property definitions, and property constants. A group of predefined property sets is part of the normative standard and its annexes, and user defined nonstandard property sets can be added to fit project or tool specific needs. Within the core of an AADL model, these properties can be used as property associations to assign a value thanks to a property expression statement.

The mapping in the general case consists in using a SysMLv2 package to represent AADL property sets, attribute definitions for property types and property definitions, attribute usages for property associations and property constants, and the KerML expression language for AADL property expressions. The validation of these mapping rules will consist in implementing the existing predefined property sets in full. This task has not been carried out yet.

AADL	SysMLv2
<i>property set</i>	<i>package</i>
<i>property type</i>	<i>attribute definition</i>
<i>property definition</i>	<i>attribute definition</i>
<i>property constant</i>	<i>attribute usage</i>
<i>property association</i>	<i>attribute usage</i>
<i>property expression</i>	<i>KerML expression</i>

Table 6 mapping for properties (general case)

The proposed mapping for AADL binding properties does not rely on SysMLv2 attributes, as the language offers better constructs to define structural relationships between components that are not directly related to data or control flows. An AADL binding establishes a many-to-one hosted/allocated relationship between components using resources and those representing resources. In particular, it maps application components to execution platform components. In practice, AADL bindings are used to define either mappings from functional architectures to system architectures (function binding), or software to hardware deployments (processor and connection bindings).

For these three special cases of AADL properties, the binding construct proposed by SysMLv2 does not match as it

expresses an equivalence instead of an allocation of separate entities. Discussions with the SysMLv2 standardization committee are still in progress to select the most appropriate construct for representing AADL binding properties. Moreover, AADL also defines allowed binding and allowed binding class properties, whose mapping with SysMLv2 has not been explored yet.

AADL	SysMLv2
<i>actual binding property</i>	<i>allocation</i> (to be confirmed)
<i>allowed binding property</i>	(to be defined)
<i>allowed binding class properties</i>	(to be defined)
<i>entrypoint properties</i>	(to be defined)
<i>Matching and substitution rules</i>	(to be defined)
<i>implemented as property</i>	(to be defined)
<i>is hardware property</i>	(to be defined)

Table 7 mapping for properties (special cases)

ONGOING TOOLING EXPERIMENTS

These definitions are supported by several ongoing efforts led by the authors.

First, we defined a SysMLv2 library defining AADL concepts that we presented. This library supports the AADL syntactic elements (component and feature categories, property sets) and draft of the static semantics. The latter is currently being refined as SysMLv2 addresses the last pending issues during its finalization.

This library can be used in multiple ways. The SEI has released two translators between AADL and SysMLv2. The AADL-to-SysMLv2 translation builds on OSATE AADL meta-model to translate AADL models to SysMLv2. The SysMLv2-to-AADL translation builds on the SysMLv2 reference implementation of the API to query SysMLv2 model elements and translate them to AADL. These translators serve two purposes: first, to validate the proposed mapping rules. Building a translator is a good strategy to ensure the completeness and correctness of the mapping rules. Furthermore, they allow to either translate legacy AADL models to SysMLv2, but also to apply existing AADL toolsets, for instance for performance analysis or model checking, on SysMLv2 models.

For its part, Ellidiss Technologies is working to take SysML v2 into account in its tool offering. In the short term, this will follow two tracks. The first one aids with textual edition of AADL and SysML v2 models as well as prototyping the bi-directional transformations thanks to the Visual Studio Code extensions developed by the company.

The second realization is more focused on providing advanced model analysis features for SysML v2 users and will leverage the existing AADL Inspector tool and its Prolog

based LAMP plugins [7]. Foreseen enhancements of this tool are the realization of user customizable AADL to SysML v2 and SysMLv2 to AADL model transformation rules, as well as an implementation of the SysML v2 constraints defined to enforce the AADL static semantics, into Prolog.

AN ILLUSTRATIVE EXAMPLE

This section provides an illustration of the SysML v2 domain library for AADL through a simple example that is representative of the benefit brought by the proposed approach.

We first present the illustrative example in its AADL form, showing its graphical representation, the complete corresponding AADL text, and a typical possible timing analysis that can be performed on such a model. The textual AADL model was automatically generated by the Stood for AADL design tool. The timing simulation was performed by the AADL Inspector tool.

In the second part, we introduce fragments of a representation of the example in SysML v2, and the current state of the AADL library for SysML v2. The corresponding SysML v2 textual files were tested with the OMG System Modeling Community Reference Implementation tool prototype, in the version that was available at the date this paper was written.

THE EXAMPLE IN AADL

The proposed example describes a minimalistic AADL real-time system that includes a sufficient level of details to enable scheduling analysis and simulation. As shown in Figure 4, this system is composed of a software application represented by an AADL process and its hardware execution platform described by an AADL processor with a real-time scheduler characterized by a scheduling protocol property.

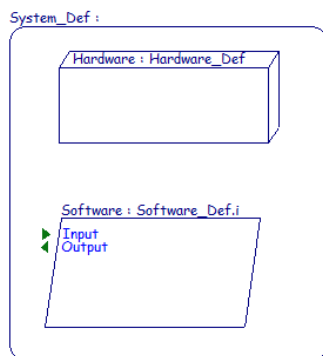


Figure 4 a minimalistic AADL real-time system

The software application is composed of two periodic threads exchanging integer values via AADL data ports. Figure 5 shows the two connected threads and the connections up and down the encompassing process. Note that this graphical representation focuses on architectural design and thus does

not show all the modeling details that must be handled by other features of the AADL authoring tool.

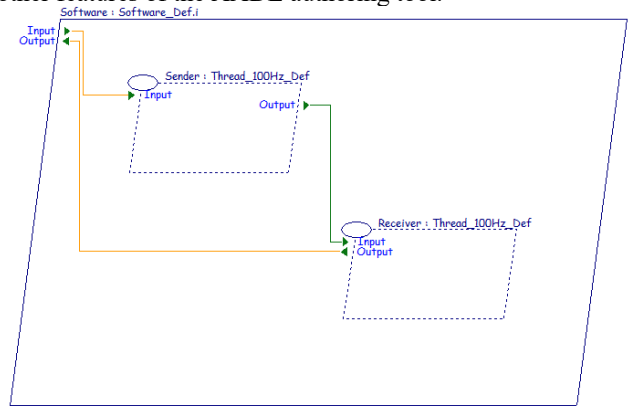


Figure 5 Software architecture of the example

The corresponding complete AADL code is shown below.

```

package AADL_Example
public
with Data_Model;

system System_Def
end System_Def;

system implementation System_Def.others
subcomponents
  Hardware : processor Hardware_Def;
  Software : process Software_Def.i;
properties
  Actual_Processor_Binding =>
    (reference(Hardware))
  applies to Software;
end System_Def.others;

processor Hardware_Def
properties
  Scheduling_Protocol => (HPF);
end Hardware_Def;

process Software_Def
features
  Input : in data port Int;
  Output : out data port Int;
end Software_Def;

process implementation Software_Def.i
subcomponents
  Sender : thread Thread_100hz_Def {
    Compute_Execution_Time => 1 ms..2 ms;
    Priority => 2;
  };
  Receiver : thread Thread_100hz_Def {
    Compute_Execution_Time => 2 ms..2 ms;
    Priority => 1;
  };
connections
  cnx1 : port Input -> Sender.Input;
  cnx3 : port Receiver.Output -> Output;
  cnx2 : port Sender.Output ->
    Receiver.Input;
end Software_Def.i;

thread Thread_100hz_Def
features
  Input : in data port Int;
  Output : out data port Int;

```



```

properties
Dispatch_Protocol => Periodic;
Period => 10 ms;
end Thread_100Hz_Def;

data Int
properties
Data_Model::Data_Representation =>
Integer;
end Int;

end AADL_Example;

```

The real-time behavior of such a concise model is fully defined by the standard AADL run-time specification. It is thus possible to execute the AADL model, as shown in the simulation trace in Figure 6.

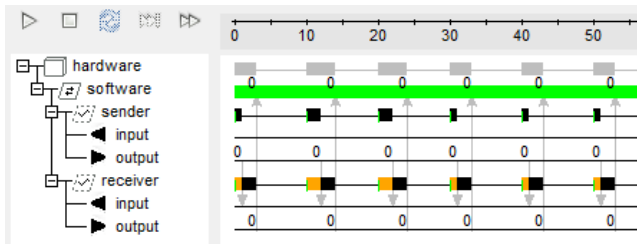


Figure 6 Simulation trace of the example

THE EXAMPLE IN SysMLv2

In this sub-section, we provide a possible representation of the same example in SysML v2 that is based on the current state of the mapping that is presented above in this paper. It is important to recall that the dynamic aspects of SysML v2 language have not been studied in depth yet, so only the static architecture semantics have been addressed for now.

The SysML v2 source text shown below describes the applicative part of the example and we can observe that its structure is very similar to the AADL one. Apart from the obvious change in terms of syntax, the most notable difference is the list of imported libraries whose role is to implement the mapping between the two languages.

```

package AADL_Example {
import AADL::*;
import AADL_Project::*;
import AADL_Project::Time_Units::*;
import Deployment_Properties::*;
import Thread_Properties::*;
import Timing_Properties::*;

part def System_Def :> System {
part Hardware: Hardware_Def;
part Software: Software_Def;
allocation SwToHw:
Actual_Processor_Binding
allocate Software to Hardware;
}
part def Hardware_Def :> Processor {
attribute :>> Scheduling_Protocol = HPF;
}
part def Software_Def :> Process {
port Input: DataPort

```

```

{ in :>> type: Int; }
port Output: DataPort
{ out :>> type: Int; }
part Sender: Thread_100Hz_Def {
attribute :>> Priority = 2;
attribute :>> Compute_Execution_Time =
1 [ms]..2 [ms]; }
part Receiver: Thread_100Hz_Def {
attribute :>> Priority = 1;
attribute :>> Compute_Execution_Time =
2 [ms]..2 [ms]; }
connection ToSe: PortConnection
connect Input to Sender.Input;
connection SeToRe: PortConnection
connect Sender.Output
to Receiver.Input;
connection ReTo: PortConnection
connect Receiver.Output to Output;
}
part def Thread_100Hz_Def :> Thread {
port Input: DataPort
{ in :>> type: Int; }
port Output: DataPort
{ out :>> type: Int; }
attribute :>> Dispatch_Protocol =
Periodic;
attribute :>> Period = 10 [ms];
}
part def Int :> Data {
attribute :>> value:
ScalarValues::Integer;
}
}

```

The SysML v2 domain library for AADL is currently organized in such a way that it follows the structure of the AADL standard document. Only small fragments of these libraries are shown in this paper.

The first library corresponds to the AADL core definitions. It implements the mapping defined between the modeling constructs of the two languages. The fragment shown below provides a generic representation for AADL components, ports and connections, as well as the specific specialization for threads.

```

standard library package AADL {
/* references:
* SAE AS-5506D AADL chapters 4 to 10
* OMG SysML v2 release 2024-02 */

enum def ComponentCategory {
enum Data;
enum Process;
enum Processor;
enum System;
enum Thread;
/* ... (fragment) */
}
abstract part def Component {
attribute category : ComponentCategory;
}
part def Thread :> Component {
attribute :>> category =
ComponentCategory::Thread;
attribute Compute_Execution_Time :
Timing_Properties::Compute_Execution_Time;
attribute Period :
Timing_Properties::Period;

```



```

    attribute Dispatch_Protocol :
      Thread_Properties::Dispatch_Protocol;
    attribute Priority :
      Thread_Properties::Priority;
  }
  abstract port def Feature;
  port def DataPort :> Feature {
    inout item type : Component;
  }
  abstract connection def Connection :>
    Connections::BinaryConnection;
  connection def PortConnection :>
    Connection {
      end source : Feature;
      end target : Feature;
    }
  abstract attribute def Property;
  /* ... (fragment) */
}

```

The second library provides a SysML v2 implementation of the AADL_Project property set. It contains a list of common declarations for AADL properties. These declarations may be customized to fit project specific requirements.

```

standard library package AADL_Project {
/* references:
* SAE AS-5506D AADL appendix A8
* OMG SysML v2 release 2024-02 */

enum def Supported_Scheduling_Protocols {
  enum RM;
  enum DM;
  enum HPF;
}
alias HPF for
  Supported_Scheduling_Protocols::HPF;
/* ... (fragment) */
}

```

Then, for each appendix of the AADL document that specifies the predefined property sets, a dedicated SysML v2 library is proposed. As an example, the library for AADL deployment properties is shown below.

```

standard library package Deployment_Properties {
/* references:
* SAE AS-5506D AADL appendix A1
* OMG SysML v2 release 2024-02 */
import AADL_Project::*;

allocation def Actual_Processor_Binding {
  end part sw : AADL::Component;
  end part hw : AADL::Component;
}
attribute def Scheduling_Protocol :>
  AADL::Property,
  Supported_Scheduling_Protocols;
/* ... (fragment) */
}

```

```
}

```

LESSONS LEARNT AND WAY FORWARD

In the previous sections, we sketched elements of the development to specialize SysMLv2 with AADL concepts. These snapshots of our current development show that SysMLv2 provides interesting abstractions that cover many use cases of AADL. At the time of writing, we covered most static semantics rules, and the most basic cases for the execution semantics.

To foster SysMLv2 adoption, the OMG recently initiated the Systems Modeling Community (SMC). The primary goal of the SMC is to gather user communities interested in specific usages of SysMLv2. The authors are currently engaged in a specific working group aiming at pushing forward this development. Our primary goal is to capture AADL semantics as SysMLv2 constructs. This would address the issues of having two languages that have some overlap. This would also equip systems engineers with a domain-specific library with consistent semantics rather than developing ad hoc solutions. A longer-term goal is to close the gap between MBSE and domain specific engineering for other classes of embedded systems (e.g., robotics) or standards (e.g., OMG MARTE or AutoSAR). The maturity of these standards or practice makes it possible to build such a community that could deliver reusable libraries for support model-based engineering activities.

REFERENCES

- [1] OMG. "OMG Systems Modeling Language (OMG SysML) Version 1.7." OMG, 2022. [ptc/22-08-02](https://www.omg.org/spec/SysML/1.7/)
- [2] SAE AS-2C "AS5506D Architecture Analysis & Design Language (AADL)." SAE International. April, 2022. <https://doi.org/10.4271/AS5506D>.
- [3] W. Zhe, J. Hugues, J.-C. Chaudemar, and T. LeSergent, "An Integrated Approach to Model Based Engineering with SysML, AADL and FACE," in *Proceedings of Aerospace Systems and Technology Conference 2018 (ASTC'18)*, 2018. [10.4271/2018-01-1942](https://doi.org/10.4271/2018-01-1942)
- [4] Nico Jansen, Jérôme Pfeiffer, Bernhard Rumpe, David Schmalzing, Andreas Wortmann, "The Language of SysML v2 under the Magnifying Glass", *Journal of Object Technology*, Volume 21, no. 3 (July 2022), pp. 3:1-15, doi:10.5381/jot.2022.21.3.a11.
- [5] Hans Peter de Koning, "SysML Version 2 – Final Stretch," ESA MBSE2022 Workshop, 22-24 November 2022, Toulouse, France
- [6] Jean-Charles Roger, Pierre Dissaux, "AADL modelling with SysML v2" ADEPT Workshop, 17 June 2022, Ghent, Belgium, in *Ada User Journal* Volume 44. Number 1
- [7] Pierre Dissaux, "LAMP: to shed light on AADL models" ADEPT Workshop, 16 June 2023, Lisbon, Portugal,

Session We.5.A
ML/AI for Critical System 2

Wednesday 12th June

17:00

–

Auditorium

Certified ML Object Detection for Surveillance Missions

Mohammed Belcaid*, Eric Bonnafous*, Louis Crison* Christophe Fauré*, Eric Jenn†, Claire Pagetti‡

* CS Group, Toulouse, France † IRT Saint-Exupéry, Toulouse, France ‡ ONERA, Toulouse, France

ABSTRACT

In this paper, we present a development process of a drone detection system involving a machine learning object detection component. The purpose is to reach acceptable performance objectives and provide sufficient evidences, required by the recommendations (soon to be published) of the ED 324 / ARP 6983 standard, to gain confidence in the dependability of the designed system.

I. INTRODUCTION

The ever increasing traffic of UAVs (Unmanned Aerial Vehicles) in the airspace represents a new threat for safety and security. In this context, we are developing a surveillance system aimed at detecting and localizing intrusions of UAVs in sensitive areas.

A. System description

The drone surveillance system is composed of two main parts: a *sensing sub-system* and a *machine learning (ML)-based detection and localization sub-system*. The sensing sub-system is composed of a radar and a camera. The radar scans continuously the area under surveillance and can detect and classify (as *UAV*, *bird* or *other*) multiple objects simultaneously within a range of 5 km and an angle of view of 120 degrees. For small objects, the performance of the radar detection and classification being low [14], the camera is used to confirm the type of the detected object on the basis of the objects locations provided by the radar.

In this work, we only consider the camera-based detection and localization functions. Therefore, we make no hypothesis on the position of detected objects in the images. This is coherent with the uncertainty inherent to the information provided by the radar. Indeed, assuming the drone to be in specific position in the image (e.g. always in the center), thanks to the radar localization, would possibly lead to miss the presence of an intruder.

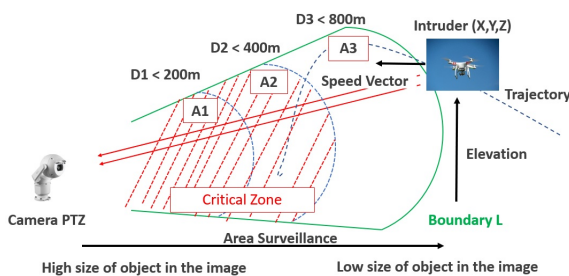


Figure 1: Surveillance Area

Figure 1 shows the surveillance area that is delimited by a conic boundary (in green on the figure). In this area, any object of at least 0.5m and at most 800m must be

detected. This area is partitioned into three sub-areas (A1, A2, and A3) in order to adapt the detection performance and latency requirements of the system to the distance to the intruder. Indeed, the closer the intruder, the faster the detection should be and the higher the quality of the detection should be. Moreover, depending on the size of the drone in the image (in pixel²), the system could execute different object detection models, the performance of which has been *optimized* with a range of object size.

For confidentiality reasons, no precise performance requirements can be given for the system. For area A3 for instance, detection performance must be higher than 80% and detection latency must be lower than 50ms. In addition, detection performance must be achieved in a large range of environmental conditions including various *backgrounds* (landscape, city,...) or weather conditions (sunny, cloudy, ...). Finally, coverage (i.e., ratio of false negatives) must be lower than 20% in order to prevent false alarms and the unnecessary triggering of the interception action (for instance).

In order to reach a high level of reliability and availability, we choose to follow the recommendations promoted in the aeronautics domain, and specifically the guidance [8] released by the EASA (European Union Aviation Safety Agency) and the soon to be published ED 324/ARP 6983 recommendations currently being developed by the SAE G34/EUROCAE WG114 working group [9]. Another aeronautical standard, named the SORA [17], has been published to regulate drone flight to ensure safe operations in air traffic and environments. However, it does not address the integration of ML models into safety-critical systems. Applying the SORA could be considered in a further step to integrate an ML-based detection function in an interceptor drone (which is a better way to treat intrusion in sensitive area rather than an on-ground system).

B. ED 324/ARP 6983

The ARP 6983 is a Process Standard for Development and Certification/Approval of Aeronautical Safety-Related Products Implementing AI¹. As of the date of redaction of this article, this document is still a work-in-progress, and a first public version is expected in Q2 2025. The ARP 6983 provides guidance that can be used as means of compliance for embedded AI. It complements existing practices to cover the specific issues raised by the introduction of AI/ML. Insights on the expected contents of the standard can be found in [11].

The ARP 6983 covers a significant part of the engineering activities for a AI/ML system, from the system/-

¹See <https://www.sae.org/standards/content/arp6983/>

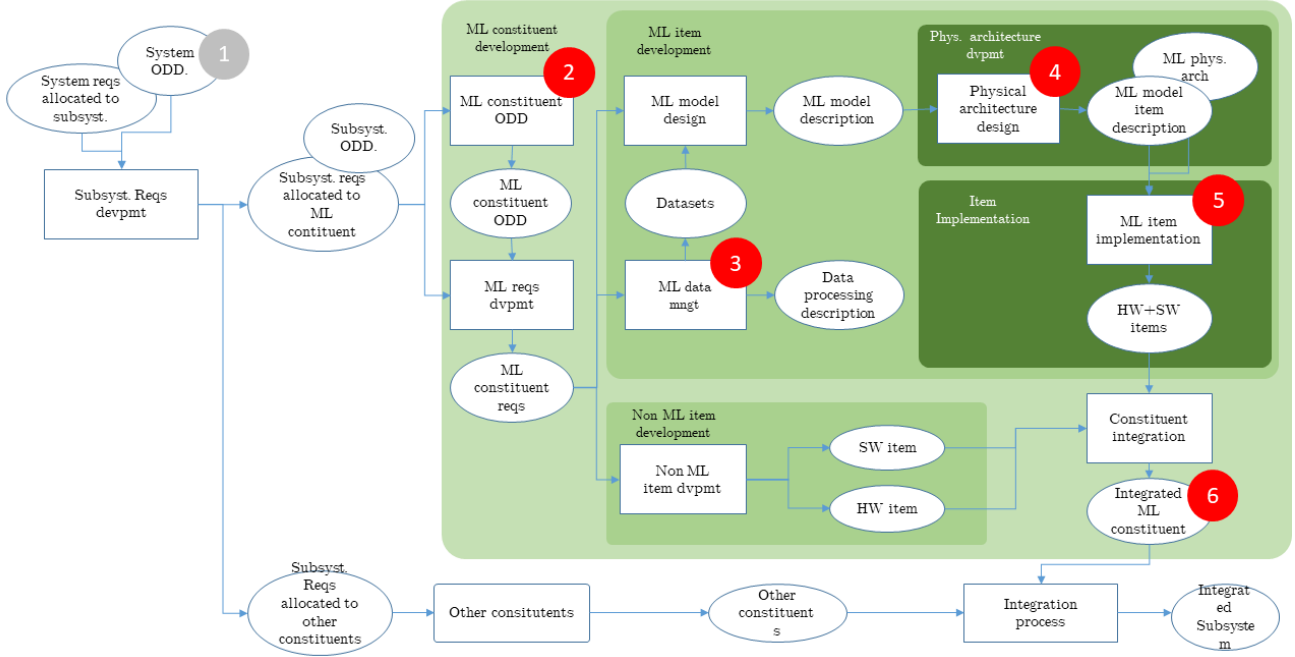


Figure 2: ARP 6983 simplified development workflow (adapted from [11])

subsystem level down to the hardware and software items levels, through the ML constituent level². An overview of the overall process is given on Figure 2. Note that this diagram refers to an interim version of the standard that still may change before the official release of the standard is published.

In this paper, the focus is placed on the activities of the standard labelled with a red tag (those in grey are mentioned for completeness). More precisely: the system Operational Design Domain (①, §II-A) and the other subsystem requirements are used to develop the ODD (Operational Design Domain) of the ML Constituent, or MLCODD (②, §II-B) and, finally, the input dataset (③, §III-B). The ML model designed to comply with the ML constituent requirements (④) is refined into one or several ML Model Item Description(s) (MLMID, ⑤), which are implemented and deployed on the target hardware (⑥, §IV).

C. Contributions

Our main contribution is the description of a partial process compliant with ARP 6983 applied in the development of an actual industrial system. The main phases of the development process addressed in the paper are the following:

- in phase 1, the system-level Operational Design Domain is specified and *propagated* to the ML *constituent* of the system,
- in phase 2, the dataset is built (selected, augmented) in compliance with the ODD and the MLCODD,
- in phase 3, the ML model is designed (selected, adapted) so as to comply with the functional (ML performance) and non functional (memory footprint and latency) requirements,

²A ML constituent is defined in the certification guideline as a constituent containing the ML model(s) and its associated data processing.

- finally, in phase 4, different implementation paths are investigated considering traceability and latency concerns.

In addition from applying this process, we also propose the following technical contributions:

- Biases on the input dataset have been identified and corrected by data augmentation;
- Compliance with detection performance and inference latency requirements has been addressed by conjointly (1) improving detection performances by (1.a) preventing information loss due to image redimensioning, (1.b) achieving good sub-image overlap, and (2) improving implementation efficiency thanks to (2.a) a quantized representation of the ML model (FP16 and INT16) and (2.b) an efficient implementation of the GEMM matrix multiplication operator.

There is a significant and increasing number of publications addressing the usage of ML in safety critical systems. When it comes to certification aspects, we can for instance mention the work on the ACAS-Xu in [5]. However, this work considers a very specific problem with a narrow operational domain (5 scalars). More recently, papers address the certification vision-based landing such as [7]. But none of them covers the full spectrum as we do and none addresses the drone intrusion detection problem. In that sense, our case study is representative of a (new) broader class of problems for which ML is considered useful.

The paper is organized as follows: Section II presents our approach to define the Operational Design Domain of our ML constituent; Section III describes the dataset design compliant to the ODD; Section IV and Section V describe respectively the model design and deployment process; Section VI concludes the paper.

II. ODD SPECIFICATION

The first phase of our process is to define the Operational Design Domain (ODD). The ODD of a system is the allocation of the Operational Domain (OD) requirements to the system, the OD being a “specification of all foreseeable operating conditions under which an end-product is expected (and should be designed) to fulfill its missions” [20]. The ODD is a crucial element for the development of any ML-based system. Below, we first specify the ODD at system level and then refine it at ML constituent level, where the ML constituent only includes the ML model and its associated processing.

A. System ODD

The system ODD has been developed from a set of operational scenarios provided by domain experts. This set is deemed to cover the complete range of conditions in which the system must operate. For defining the operational scenarios, we use some terminology of the ISO 34503 [19], a standard that proposes some concepts and requirements to enable the definition of an ODD of an automated driving system. In particular, an operational scenario determines (i) the environmental conditions, (ii) the set of *dynamic elements*, also referred to as *objects* (intruder or drone, birds or other), (iii) the set of *scenery elements* (spatially fixed elements) of the system environment (“landscape”, “sun”, etc.) that must be considered and (iv) the set of attributes characterizing those (dynamic and scenery) elements (e.g., “position in the sky” for the “sun” element, “type of drone” for the “intruder” element, etc.). Here is an example of such a scenario.

Operational scenario 1: is defined by:

- Environmental conditions: *Time = 2 pm. Season = winter. Location: Europe. Atmosphere (nebulosity = none – meaning that the weather is clear).*
- Scenery elements: *The system is installed in an urban area with background buildings of high below 15m.*
- Dynamic elements: *A 50cm x 50cm x 20cm drone arrives on the hand left side of the surveillance area (with orientation = (10°, 25°, 3°)) at a distance of 450m from the system, moving with a straight trajectory, in the direction of the system, at a constant speed of 1m/s. Sun is visible (on the left hand side of the image).*

The set of operational scenarios encompass many *elements* and associated *attributes* that are translated as a set of constraints that in fine defines the ODD. Those constraints can be numerous, highly complex or non tractable by human. It may that an additional simplification step is necessary to aggregate some constraints into simpler ones, that can be more easily interpreted at the image level. For instance, the “position of the sky” and “presence of clouds” could be folded into a single attribute called “lightning conditions”. In addition, the experts should define the *realistic* distribution of elements (together with their attributes). For the drone intrusion detection, some of the constraints obtained to define the ODD are:

- The *type of intruders* is in {Quadrotor, Birotor};
- The *size of intruders* is within [0.5m, 1m];
- The *type of area* is in {urban, sub-urban, country side};

- The *time of the day* is in within [6am, 10pm];
- The *lighting condition* is the range [sunny, slightly cloudy];
- etc.

To complete the ODD, the EASA guidelines [8] [20] (*Anticipated MOC DM-01-1*) and the ARP require to identify particular conditions that need to be specified explicitly 1) so as to be surely taken into account (edge-cases) during training and testing; or 2) on the contrary to be removed from the ODD (outliers). Those edge-case conditions may refer, for instance, to specific relations between attributes of the environment elements. The condition where an object has the same color as some other elements of the environment (e.g. painted intruder in green to reproduce the grass and hide easily) is such an example in our context. Edge-case conditions could be generated by randomly sampling values of the various attributes identified in the ODD, but with a very low probability.

For example, the ODD given above allows some other objects such as distant airplanes or helicopters to appear in the surveillance area where they would be hard to distinguish from a drone (to some extent, some drones may be seen as a “small airplanes”). These situations are considered rare and as outliers. Actually, they can be made as rare as necessary by forbidding to install the system in areas close to airports, for instance. The ODD must clearly address those outlier situations.

Finally, let us remind that the definition of the ODD is an iterative process. For instance, the latter constraints on edge-cases and outliers have to be “reinjecte” in the ODD to complete it. The model design could also lead to refine the ODD.

B. ML Constituent ODD

According to [9], the ML constituent is the “defined and bounded set of either hardware item(s) and/or software item(s) that implement ML”. In our case, the ML constituent is a software component (running on some piece of hardware) that takes as input images provided from a camera and generates as outputs data representing bounding boxes of objects detected in the image along with their classification. The ML constituent, figure 3, contains three main software components (the pre/post-processing and the ML model implementation). The pre-processing is in charge of translating the raw image into a format expected by the ML model (e.g. resize high resolution images into 640x640). Typical object detection algorithms are presented in [22]. The post-processing is in charge of computing the localisation of the intruder (if any) in the image and providing the absolute position on the area.

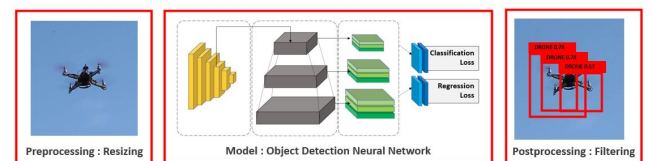


Figure 3: ML Constituent

The ML constituent ODD (MLCDD) specifies “the foreseeable operating conditions under which an ML Constituent is expected to work” [9]. In this paper, focus is

placed on the image processing elements, so the MLCODD of interest refers to constraints related to the processed images. Some of those constraints represent the “projection” of the system-level ODD constraints to the image domain; some others are related to the technical solutions used to implement the ML components. Examples are:

- The *size of objects* is in the range $[20 \text{ px}^2, 400 \text{ px}^2]$ with 95% of them in $[20 \text{ px}^2, 100 \text{ px}^2]$;
- The *position of objects in the image* is uniformly distributed in the 2 (geometrical) axes of the image;
- The *main frequency components of the image*³ is in the wavelength greater than 20 px (covering a range from e.g., a solid clear sky background to a complex grass background);
- The *mean brightness of the image* computed as the mean of the V value for the image coded in HSV (Hue Saturation Value) should belong to a pre-defined range;
- etc.

III. DATASET DESIGN

The second phase of our process is to create the dataset that is used during the training, validation and test phases. The dataset is built so as to comply with the definition of the MLCODD (see section II-B). In our case, we exploit private and public existing data sources including [10], [24].

A. Biases analysis

For the system to behave with the expected level of performance in operations, the dataset used during the training phase must reflect the distribution of situations that will be actually encountered during operations. These distributions and constraints are defined by the ODD. Let us take as an example the position of the objects in the image. The ODD states that their position is uniformly distributed on the camera image plane⁴

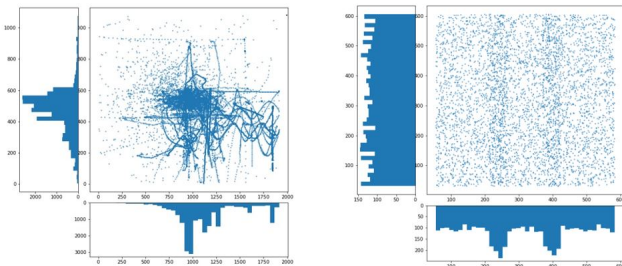


Figure 4: Object positions in the original dataset on the left and the augmented dataset with unbiased position on the right

Figure 4 left hand side shows the spatial distribution of the objects in our initial dataset: 70% of the objects are located in a very narrow area centered in the image. This is clearly not representative of the actual operational conditions (and thus not compliant with the MLCODD), for it would mean that most objects would fly towards the system from a far distance and remain centered on the camera axis. This is clearly a bias that may have

³https://en.wikipedia.org/wiki/Frequency_domain

⁴As stated earlier, the system is also fitted with a radar that would normally place the camera axis in the direction of the object. Here, we consider that this feature is not reliable and ignore it.

a significant negative impact on the capability of the system to detect objects in operating conditions such as the early stage of intrusion (i.e., when the objects enter the surveillance area from the side of the observation cone) and when the drone tries to leave the camera’s view. Having such a biased dataset during the learning phase would potentially lead the network to *erroneously* correlate the presence of a target to its position in the image. This would also significantly reduce the detection accuracy for objects located at the edge of the image.

By displaying the size of the bounding boxes (Figure 5), we observe that the dataset is compliant with the ODD with respect to object sizes. Indeed, almost 95 % of the boxes are smaller than 100x100 pixels.

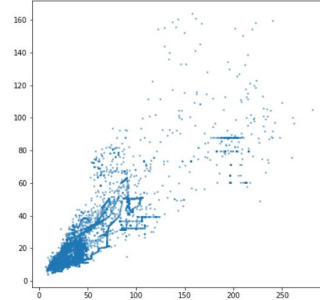


Figure 5: Distribution of size of object in the dataset

B. Dataset augmentation

In order to remove the biases and improve the representativeness of the dataset with respect to the MLCODD, the dataset must be enriched. There are several strategies to enrich the dataset among which making real data collection in real environment. A real data collection is always a challenge because it is expansive and it can hardly reach the quantity / distribution / independence requirements. For example, ensuring a large variety of intruders would imply to have access to many drones and program them to make several types of intrusion. Collecting several images of one drone intrusion corresponds to one intrusion and dependent data. Moreover, reaching a uniform drone position on the camera would be highly challenging. This is the reason why, augmentation with image processing is often used, in addition to real data collection, to reach compliance with the ODD. We have thus applied the following data augmentation techniques:

- generating images with objects at various positions and with various sizes;
- generating inlaying objects in various backgrounds (e.g., sky or urban background);
- generating new versions of existing images with modified brightness;
- generating images with various numbers of objects (from 0 to 4).

To unbiased the dataset, we develop the algorithm 1 that is detailed hereafter. The idea of the algorithm is to create new images from the original dataset by performing image transformations. A possible transformation consists in transforming an 1080x1920 image with a drone at coordinate (486,921) (i.e. in the center) into an 640x640 image with the drone at coordinate (486,510) (i.e. bottom right). The spatial distribution of the new unbiased dataset

with respect to the object position is shown on the right hand side of Figure 4. Thanks to data augmentation, the objects are uniformly distributed in the 2-axes.

Algorithm 1 Spatial Coverage Algorithm

- 1: Let D be the set of all 1080x1920 images
 - 2: Let s be the size of the target image
 - 3: **for** i : Image **in** D **do**
 - 4: Get the coordinate of the object in i
 - 5: Collate i with 7 background images along each side of i .
 - 6: Get the coordinate of the object in the new reference frame
 - 7: Generate a random position of a window where the object is present and then crop the window
 - 8: **end for**
 - 9: Save all new pictures in the dataset P with 640x640 images
-

The algorithm 1 allows to generate new images with a specific size and a new object position. Let us explain the algorithm via an example shown in the Figure 6. First an image is picked (yellow in the picture), then is replicated nine times to produce a larger image (red). The yellow image can contain an intruder or can be a background. A window (green) with a size of (640,640) is randomly selected and cropped. The resulted image (in green on the bottom) is added to the dataset P . The algorithm does not resize the objects and allows to off-center them.

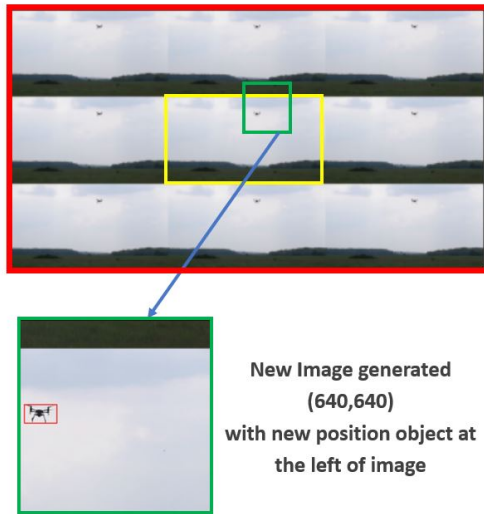


Figure 6: Algorithm 1 in action

In addition to algorithm 1, we create a “mosaic dataset” to increase the number of samples with different contexts. Figure 7 shows a mosaic image of size (640,640) which is an aggregation of 4 samples of size (320,320) with slight variations of brightness and image geometry. This allows to address the brightness and main frequency components constraints of the MLCOOD.

C. Compliance with the MLCODD

We have analysed the initial dataset D along the dimensions (e.g. size of object, position in the image, number of objects ...) identified by the MLCODD and compared with



Figure 7: Mosaic dataset creation

the constraints set by the MLCOOD. As several constraints were not satisfied, we have extended D so as to comply with the definition of the MLCODD. In this paper, we illustrate some of those dimensions (at the end of section II-B) and provide the algorithm 1 to illustrate some image transformations to reach compliance. In particular, we have unbiased the dataset with respect to the objects’ positions by adding images with objects relocated at random places. We have also increased 1) the coverage of drone attitude thanks to rotation and symmetry transformations (in the mosaic approach); 2) the representativeness of contexts with a large diversity of backgrounds and variation of brightness; 3) the situations to have from 0 to 4 intruders on images. The final dataset P is more than three times bigger than the initial one D .

The last step of the data management process is the splitting of the dataset into three subsets: training, validation and test datasets, each compliant with the MLCOOD.

IV. MODEL DESIGN

The third phase of our process is to select and adapt a detection and localization algorithm in order for the ML constituent to perform the intended function. The model design was done following state-of-the-art machine learning approaches and an ad hoc tiling strategy to optimize the accuracy.

A. Requirements

The ML constituent, and the ML model, are designed to realise the intended function (detect safely and quickly intruders in the sensitive area) in all operational scenarios defined during the ODD design. In addition to the (MLC)ODD definition, the design phase has also identified requirements to be fulfilled. There are several types of requirements including, but not restricted to, *functional performance*, *output format* compatible with the intended function and *real-time performances*.

MLC Requirements 1 (Functional Performance): The first type of requirements concerns the detection capacity of the model.

- The ML model shall classify objects with an accuracy greater or equal to 90% in areas A1 and A2, and greater or equal to 80% in area A3.
- The ML model must have a false alarm rate of less than 20%.
- The ML model must have a non missed UAV rate of less than 20%.

The ML constituent is expected to output the image coordinate of the intruder(s) if any, to provide the classification,

to crop of the detected intruder(s) and to compute their absolute position. All of these information are used for the decision-making (e.g. interception).

MLC Requirements 2 (Output format): The ML model shall localize object in the image with a bounding box and should output the bounding box coordinates, the classification and the confidence level.

The last type of requirements applies to the deployment and implementation of the ML model on the target.

MLC Requirements 3 (Real-time performances): The ML model shall be deployable on an Nvidia Xavier AGX target with the minimal use of COTS software and libraries. Moreover, the ML model shall detect and localize objects in at most 50 ms.

B. Model selection

Object detection, classification, and localization tasks are usually done using deep learning neural networks. We experimented two different models: a one-stage model (YOLOv3 [21]) and a two-stage model (Faster RCNN [13]). While the accuracy was similar for both models, the inference latency of the two-stage model was incompatible with the *real-time performance* requirements. In addition, the YOLOv3 model has the capability to detect objects at different scales, and this maps nicely to the different areas considered in the system. We choose the YoloV3 tiny [1] model which is a refinement of the lighter YoloV3 model with optimized feature scaling, making it more efficient in terms of latency.

Another important design choice is to rely on a smart pre-processing. During the design of the model, focus was placed on improving the detection of small objects, while keeping the architecture of the YOLOv3 component. To achieve this goal, we choose not to rescale the 1920x1080 camera image in order to preserve its information content. Instead, we decompose the high resolution image into several 640x640 tiles with some overlap, as described in the next paragraph and shown on Figure 9. This solution reveals to be a good trade-off between detection performance, latency, and memory footprint.

C. Tiling Strategy

The transformation of a high resolution 1920x1080 image into multiple smaller images (or *tiles*) without image compression can be done in several ways. Such tiling strategy depends on the tile size (here fixed at 640x640) and the targeted overlap between tiles [23]. Each tile must then be analyzed by the YoLo model leading to several model inferences to cover the full high resolution image. This has a direct impact on the real-time performances (and the latency). In terms of implementation latency, having no overlap is the best solutions since there will be less tiles. However, in that case, the detection algorithm only gets a partial view of objects located at the boundaries in each tile. Moreover, models are known to badly detect objects (even if complete) on boundaries due to phenomena of blind spots [2] and spatial bias [25]. So we must choose the size of the overlap to force the intruder(s) to be in at least one optimal inference area, as shown on Figure 8.

As a consequence, the objective of our tiling strategy is to find an *optimal decomposition trade-off* that allows



Figure 8: Optimal Inference Area

some overlap for the detection accuracy and that produces reasonable number of model inferences for the detection latency. We have defined two strategies depending on the detection areas since they do not share the same functional and real-time requirements. Indeed, in areas 1 and 2, the drone size in pixel is larger than in area 3 (thus the detection performance is easier) but the latency is shorter. We thus select for those areas the configuration of Figure 9 left hand side composed of three 1080x1080 tiles with an overlap of 50%. The 4 blue points represent the input space to cover. Because the YoLo model expects 640x640 tiles, a resize processing is applied on each 1080x1080 tile. Since the drone size is large, the resizing will not reduce it too much and the YoLo performance remains in the range of acceptable models. The overlapping is important (50%) to increase the accuracy on the frontier between the left and right parts of the image. Finally, having 3 tiles leads to a reduced latency.

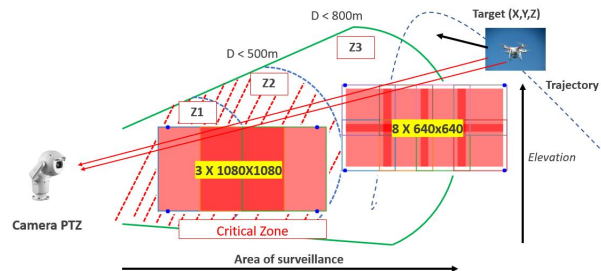


Figure 9: Tiling Strategy according to the Area

The case of area 3 is different since the drones are far away and their size in pixel is small. The tiling scheme for this area is shown on Figure 9 right hand side. It is composed of eight 640x640 tiles with an overlap of 30%. Resizing would degrade too much the performance thus the input image is just decomposed. Having more than 8 tiles is challenging for the implementation and the latency constraints. The two tiling schemes, summarized in Table I, meet latency and memory requirements and provide similar object detection performance for each area.

AREA	A1 and A2	A3
Tiling	(3,1080,1080)	(8,640,640)
Overlap	50%	30%
Resizing	56% on one axis	None

Table I: Tiling configuration for inference optimization

D. Model optimization

In order to lower the computational footprint of the YOLOv3 algorithm while maintaining the detection performances, we modified its backbone by replacing the 7 2D-convolution layers with depth-wise separable convolution (DSC) layers. The use of DSC dramatically reduces the number of operations and the memory footprint. Such layer

is in particular used in the MobileNet model to support efficient object detection on embedded devices [18].

V. MODEL IMPLEMENTATION AND DEPLOYMENT

The last phase of our process is to implement and deploy the design model. The implementation must ensure that all the requirements listed in section IV-A are satisfied. In addition, we also consider ARP 9683 development assurance concerns, including the capability to demonstrate traceability between the ML model and its implementation.

A. Implementation approach

The target as already mentioned is the NVIDIA Xavier AGX platform that comes with 8 Carmel ARM-core, a GPU and a NVDLA (NVIDIA Deep Learning Accelerator). We selected the Darknet implementation framework [21] that supports many YoLo object detection and classification algorithms, including the YoLov3 tiny that we selected. Darknet is open-source, which means that it can be analysed and possibly assessed for traceability and semantic preservation analysis. It has two back-ends: C code for CPU and CUDA code for NVIDIA GPU targets. The training of the model was done with the Keras framework. We then manually describe the model architecture of Keras in a ".cfg" textual file (layers and operators) and export the parameters of each layer in a ".weights" binary file (coded in the IEEE754 format and little endianness). These two files must be consistent with each other and correspond to the ML model description (MLMD) provided at the end of the model design. These files are loaded as input by Darknet to generate the (C or CUDA) code, see figure 10.

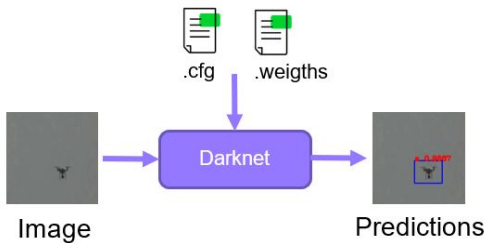


Figure 10: Darknet Framework

B. C code generation for CPU

We first investigate a CPU-only implementation and port the executable on one ARM code of the Xavier. The generated C code is similar to the neural network description which facilitates traceability activities. Several tests were made and the behaviour on the target was similar to one observed in the learning framework Keras. Unfortunately, the real-time requirements are not satisfied.

In order to compensate the limited performances of the CPU, we investigate the use of fixed-point arithmetic, on particular on the convolution layers. Darknet generates Generalized Matrix Multiply (GEMM) [4] based implementation. We optimized GEMM operator with ideas inspired by [12], [3], [16]. Figure 11 shows the latency figures for layer 24 of the YOLOv3 model, using 32-bit floating point arithmetic (left side) and 16-bit fixed point arithmetic (right side). We observe a 50% latency drop between the two implementations, a significant reduction

of the measurement dispersion, and a reduction of 29% of the Observed Worst-case Execution Time (OWCET). Unfortunately, this performance level still does not meet our requirements.

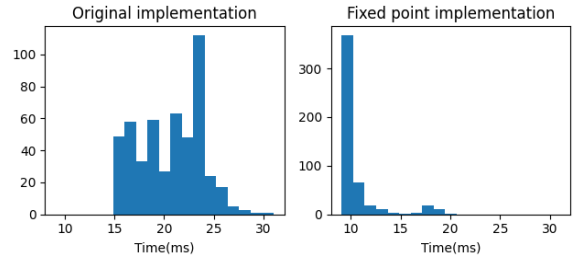


Figure 11: Original Darknet GEMM implementation vs. fixed point modified version (latency for layer 24)

C. Cuda code generation for GPU

We then try the Cuda code generated by Darknet on the NVIDIA GPU. In this case, layer 24 of the FP32-encoded YOLOv3 model is executed in around 1.2ms, which represents around 7% percent of the latency measured on the CPU. Another optimization was achieved by replacing the classical convolution operator used in the initial model by a depth-wise separable convolution (DSC) (already discussed in Section IV). This optimization led to a reduced inference time for each tile of (640,640). The Table II summarizes the latency on the NVIDIA Xavier AGX platform.

Model (FP32)	YOLOv3	YOLOv3 DSC
Inference time	26ms	20ms

Table II: YOLOv3 inference

Finally, a second level of optimization was done by modifying Darknet to generate Cuda code in half precision (i.e. FP16) to reduce memory footprint and latency. By applying FP16 quantization to the entire neural network, the total inference time decreases to 15 ms, a 25 % improvement over the initial FP32 model. Note that we also benchmarked different batch schemes to process the 8 tiles of the full resolution image (see section IV). We did not gain any latency benefit with such an approach.

D. Optimized Cuda code for GPU

The CUDA FP32 and FP16 implementations meet the latency requirements, but they rely on the closed-source cuBLAS library⁵. Using COTS libraries in safety-critical systems can be discouraged because traceability analysis is difficult and static WCET analysis could be unattainable. This is the reason why, we tried our own GEMM operator implementation inspired by the article of⁶. In particular, we applied some optimizations, taking into account the specific structure of the network and the GPU platform.

We remind that to execute a convolution with a GEMM operator, the input 3D tensor is translated into a 2D matrix. A 3D tensor is defined by its three dimensions (H,W,C) where H refers to the height, W the width and C the number of channels. The classical algorithm to translate a 3D tensor into a matrix for GEMM is the `im2col` method [4] shown

⁵<https://docs.nvidia.com/cuda/cublas>.

⁶<https://siboehm.com/articles/22/CUDA-MMM>

in Figure 12. Note that nothing needs to be done for the kernel.

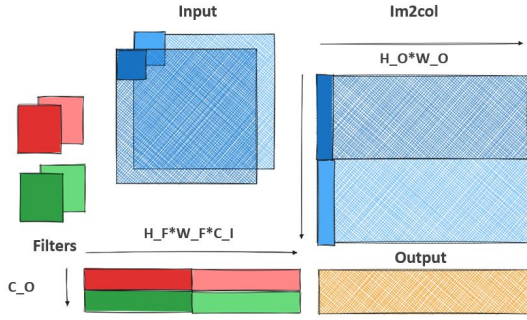


Figure 12: im2col tensor

There is a huge variability in the tensor sizes that are exchanged within the layers of the YoLo architecture. Indeed, the first layers are made up of large images with few channels (large (H,W), small C), while the last layers are made up of small vignettes with a large channel depth (small (H,W), large C). We thus have two main configuration types:

- C1 : First Layers : (Large (H,W), Small C)
- C2 : Last Layers : (Small (H,W), Large C)

These two types of tensor impact directly the size of the 2D matrix produced by the im2col function. The CUDA GEMM operator then necessitates different optimizations depending on the size of generated matrix.

To perform the matrix multiplication $A \times B$, GEMM decomposes the matrices A , B and C into blocks A_s , B_s and C_s . Those blocks contain another level of decomposition (micro-panels and tiles) [15], [6] that we refer to here as chunk. Each chunk of the two matrices A and B is loaded in a shared memory array and a thread block on an SM calculates the element of the chunk of matrix C . Note that the C chunk is partially computed and needs to be accumulated with several block operations. Figure 13 shows how the global matrix multiplication is decomposed using small blocks. The block parameters are:

- M, N and K: dimensions of A (M,K) and B (K,N);
- BM: Number of rows in blocks C_s and A_s ;
- BN: Number of columns in blocks C_s and B_s ;
- BK: Number of A_s columns and B_s rows;
- TM: Number of elements of C computed by each thread.

A GPU is composed of multiple SM (Stream Multiprocessor) where a SM is a general purpose processor (with cache, shared memory, etc.) able to execute several thread blocks in parallel. Each thread in a thread block is executed on the same SM.

We propose to adapt the GEMM algorithm by adjusting the matrix block parameters as efficiently as possible according to the tensor configurations C1 and C2. To find the optimal chunk decomposition, we have (i) maximized the occupancy of the SM (we set the registers to 32 per SM to increase the number of 4 blocks used per thread to reach 100 % of occupancy), (ii) reduced the data movements on shared memory by optimizing the number of element calculated per thread so as to reduce the load of elements of A_s , B_s . In addition, the reproducibility of the results was also verified. All experiments are made in float32.

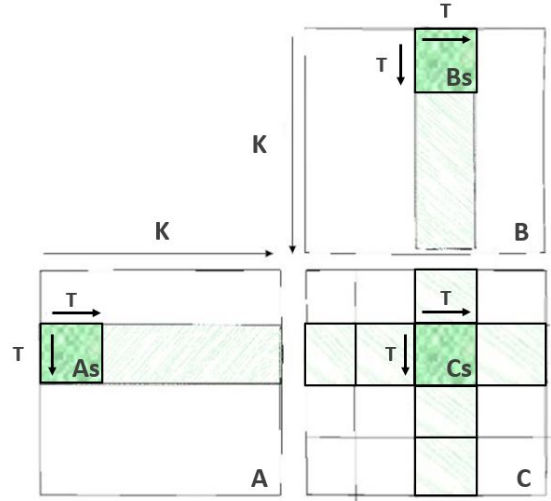


Figure 13: Matrix multiplication accumulation

As shown in Figure 13, matrices A and B are decomposed into blocks. These blocks “traverse” the rows of matrix A and the columns of matrix B to compute the blocks of the output matrix C . In order to maximize performances, the size of these blocks is chosen so that they fit into shared memory. Then we optimize the thread in a thread block to load multiple elements in A and B to reduce data movement in shared memory. Moreover, the im2col operation provides a contiguous mapping of data in memory that enables to optimize the calculus in the same block of threads. In our case, the values of those parameters are given on Table II.

GEMM Tiling	First Layer (C1)	Last Layer (C2)
M	16	1024
N	409600	400
K	27	4608
BN	16	64
BM	16	64
TM	8	8
BK	BN/TM=2	min(BN/BM)/TM=8

Table III: GEMM block decomposition parameters

Performances of our implementation with respect to CuBLAS’ is given on Figure 14. The performance remains lower than the highly optimized version implemented in CuBLAS by NVIDIA’s engineers, but it is sufficient to meet our latency requirement.

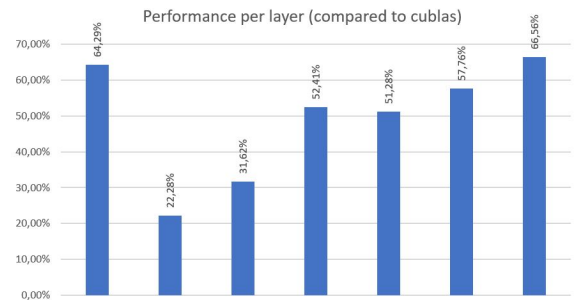


Figure 14: Performances of the GEMM implementation

VI. CONCLUSION

The purpose was to design a surveillance system to detect and localize intrusions of UAVs in sensitive areas.

To support the design, we decided to follow part of the ED 324/ARP 6983 guidance in order to help us increase the confidence and reliability of the design. We have addressed, in a pragmatic way, some important issues raised when integrating a ML component in a system performing critical, real-time functions. We have proposed a partial process compliant with the ARP 6983 that helped us reaching a much higher quality / confidence in the system. The process focuses on:

- the ODD and MLCODD definition. This was possible with the definition of operational scenarios and their translation into constraints that can be mapped in the image domain;
- the dataset design. We have shown how biases on existing datasets have been detected and corrected via an appropriate data augmentation. The verification of the dataset compliance with the MLCOOD is done by providing insights of the dataset, such as the distribution of scenery elements with respect to the constraints identified for the ODD;
- the ML model design in order to reach the expected functional performances while integrating implementation constraints. This results from a classical precision vs. latency trade-off by preventing the loss of useful information thanks to the use of a tiling strategy instead a simple image resizing;
- the implementation with optimizations. Again, we addressed conjointly performance and dependability, by developing an ad-hoc implementation of the most demanding operator (GEMM) that allows to reach an acceptable performance level with a full traceability of the source code to the input model.

As stated previously, we have only addressed a small subset of issues raised by embedding ML components. In future work, we will extend our work in several directions including in particular (i) a more precise and complete definition of the ODD in order to improve the quality of the datasets and, possibly, detect the out-of-ODD conditions in which the system cannot operate safely, (ii) a more traceable implementation of the model (currently, traceability ends with the NVIDIA drivers and hardware comes into play).

Finally, we will consider architectural means to alleviate the remaining and irreducible concerns raised by ML. In particular, we will consider the addition of system-level monitoring and mitigation means. Finding independent (e.g., non-ML) solutions remains a challenge considering that the function essentially relies on image processing.

ACKNOWLEDGEMENTS

This work has benefited from 1) the AI Interdisciplinary Institute ANITI, funded by the “Investing for the Future – PIA3” program Grant agreement ANR-19-P3IA-0004; 2) the PHYDIAS 2 project funded by the French government through the France Relance and the NextGenerationEU programs; 3) ARCHEOCS project funded by the French Research Agency (ANR) and the partners of the IRT Saint-Exupéry Scientific Cooperation Foundation.

REFERENCES

[1] P. Adarsh, P. Rathi, and M. Kumar. Yolo v3-tiny: Object detection and recognition using one stage improved model. pages 687–694, 03 2020.

[2] B. Alsallakh, N. Kokhlikyan, V. Miglani, J. Yuan, and O. Reblitz-Richardson. Mind the Pad – CNNs can Develop Blind Spots. *arXiv*, 2010.02178, 2020.

[3] D. Ben Khalifa and M. Martel. An evaluation of POP performance for tuning numerical programs in floating-point arithmetic. In *4th International Conference on Information and Computer Technologies, ICICT 2021, Kahului, HI, USA, March 11-14, 2021*, pages 69–78. IEEE, 2021.

[4] K. Chellapilla, S. Puri, and P. Simard. High Performance Convolutional Neural Networks for Document Processing. In G. Lorette, editor, *Tenth International Workshop on Frontiers in Handwriting Recognition*, La Baule (France), Oct. 2006. Université de Rennes 1, Suvisoft. <http://www.suvisoft.com>.

[5] M. Damour, F. Grancey, C. Gabreau, A. Gauffriau, J.-B. Ginestet, A. Hervieu, T. Hureau, C. Pagetti, L. Ponsolle, and A. Clavière. *Towards Certification of a Reduced Footprint ACAS-Xu System: A Hybrid ML-Based Solution*, pages 34–48. 08 2021.

[6] I. De Albuquerque Silva, T. Carle, A. Gauffriau, V. Jegu, and C. Pagetti. A predictable SIMD library for GEMM routines. In *IEEE 30th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2024.

[7] E. Denney and G. Pai. Assurance-driven Design of Machine Learning-based Functionality in an Aviation Systems Context. In *42nd AIAA/IEEE Digital Avionics Systems Conference (DASC 2023)*, 2023.

[8] EASA. EASA Concept Paper: First usable guidance for Level 1 and 2 machine learning applications, 2023.

[9] EUROCAE WG-114/SAE joint group. ED 327/ARP 6983 – Certification/approval of aeronautical systems based on AI, 2024. on going standardization.

[10] S. Fujii, K. Akita, and N. Ukita. Distant bird detection for safe drone flight and its dataset. In *2021 17th International Conference on Machine Vision and Applications (MVA)*, pages 1–5, 2021.

[11] C. Gabreau, A. Gauffriau, F. D. Grancey, J.-B. Ginestet, and C. Pagetti. Toward the certification of safety-related systems using ML techniques: the ACAS-Xu experience. In *11th European Congress on Embedded Real Time Software and Systems (ERTS 2022)*, Toulouse, France, June 2022.

[12] F. Geyer, J. Freitag, T. Schulz, and S. Uhrig. Efficient and mathematically robust operations for certified neural networks inference. *ArXiv*, abs/2401.08225, 2024.

[13] R. B. Girshick. Fast R-CNN. *CoRR*, abs/1504.08083, 2015.

[14] J. Gong, J. Yan, D. Kong, and H. Hu. Interference of radar detection of drones by birds. *Progress In Electromagnetics Research M*, 81:1–11, 01 2019.

[15] K. Goto and R. A. v. d. Geijn. Anatomy of high-performance matrix multiplication. *ACM Trans. Math. Softw.*, 34(3), 2008.

[16] R. Goyal, J. Vanschoren, V. van Acht, and S. J. J. Nijssen. Fixed-point quantization of convolutional neural networks for quantized inference on embedded platforms. *ArXiv*, abs/2102.02147, 2021.

[17] H. Habibi, D. M. K. K. V. Rao, J. L. Sanchez-Lopez, and H. Voos. On SORA for high-risk uav operations under new eu regulations: Perspectives for automated approach. *arXiv*, 2303.02053, 2023.

[18] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017.

[19] ISO. ISO 34503:2023. Road Vehicles — Test scenarios for automated driving systems — Specification for operational design domain, 2023.

[20] F. Kaakai, S. S. Adibhatla, G. Pai, and E. Escorihuela. Data-centric Operational Design Domain Characterization for Machine Learning-based Aeronautical Products. *arXiv*, 2307.07681, 2023.

[21] J. Redmon and A. Farhadi. Yolov3: An incremental improvement. *arXiv*, 1804.02767, 2018.

[22] Shayere Sarkar and Baishali Mullick and Zhenwei Liu and Dhruv Vashisht. Lecture on Anchor-free Object Detection using Multi-Level Feature Pyramids, 2020.

[23] F. O. Unel, B. O. Ozkalayci, and C. Cigla. The power of tiling for small object detection. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 582–591, 2019.

[24] J. Zhao, G. Wang, J. Li, L. Jin, N. Fan, M. Wang, X. Wang, T. Yong, Y. Deng, Y. Guo, et al. The 2nd anti-uav workshop & challenge: methods and results. *arXiv preprint arXiv:2108.09909*, 2021.

[25] Z. Zheng, Y. Chen, Q. Hou, X. Li, P. Wang, and M.-M. Cheng. Zone evaluation: Revealing spatial bias in object detection. *arXiv*, 2310.13215, 2023.

How to design a dataset compliant with an ML-based system ODD?

Cyril Cappi*, Noémie Cohen[†], Mélanie Ducoffe[†], Christophe Gabreau[†], Laurent Gardes*, Adrien Gauffriau[†], Jean-Brice Ginestet[‡], Franck Mamalet[§], Vincent Mussot[§], Claire Pagetti[¶], David Vigouroux[§]
*SNCF, [†]Airbus, [‡]DGA, [§]IRT Saint Exupéry, [¶]ONERA

Abstract—This paper focuses on a Vision-based Landing task and presents the design and the validation of a dataset that would comply with the Operational Design Domain (ODD) of a Machine-Learning (ML) system. Relying on emerging certification standards, we describe the process for establishing ODDs at both the system and image levels. In the process, we present the translation of high-level system constraints into actionable image-level properties, allowing for the definition of verifiable Data Quality Requirements (DQRs). To illustrate this approach, we use the Landing Approach Runway Detection (LARD) dataset which combines synthetic imagery and real footage, and we focus on the steps required to verify the DQRs. The replicable framework presented in this paper addresses the challenges of designing a dataset compliant with the stringent needs of ML-based systems certification in safety-critical applications.

I. INTRODUCTION

Artificial Intelligence (AI) is rapidly becoming a cornerstone technology in various sectors, including transportation. In aeronautics, AI promises efficiency enhancement and operational cost reductions, yet its adoption remains complex. This is primarily due to the stringent certification process these systems must undergo to meet the rigorous safety standards of the domain. Thus, this paper delves into the challenges of certifying AI in aviation, focusing on the design of a dataset that would comply with the Operational Design Domain (ODD) of an AI-based system.

A. Certification guidelines

EUROCAE WG-114/SAE G-34 is a joint working group on the certification of ML-based systems that will release the ED-324/ARP-6983 soon. Even if not yet published, there are several publications [1], [2], [3] that highlight the objectives and activities expected by the Aerospace Recommended Practice (ARP). In parallel, the European certification authorities – EASA – published their concept papers [4], [5] that aim at guiding applicants introducing AI (Artificial Intelligence) / ML (machine learning) technologies into systems intended for use in safety-related or environment-related applications.

Both guidelines rely on the existing standards as much as possible. From an airborne perspective, this means using the ED-79/ARP-4754A [6] guidance whenever possible to integrate the ML-based function at subsystem level and using the ED-12C/DO-178C [7] and the ED-80/DO-254 [8] when it comes to the deployment of the ML models onto respectively software and hardware items. The change of paradigm that comes with a data-driven development method entails a new

process that covers the whole spectrum of ML-based system development.

In this work, we only focus on part of the new process called *data management* to produce a dataset whose internal features match the intended function and its operating environment. Practically, the intended function must be defined with its Operational Design Domain (ODD). Then, the question is how to design a dataset compliant with such an ODD.

B. Motivation

To support our work, we have selected a safety-critical application, namely visual-based landing (VBL). Indeed, increasing the level of autonomy of aircraft will ease the flying in case of pilots' cognitive load and would therefore improve the safety in civil aviation. In a future where it is envisaged to fly with only one pilot on board, a single pilot may not be in capacity to assume all tasks required during the landing phase (especially the final ones). Thus, vision-based landing systems could circumvent such a need and would be in charge of computing the position of the aircraft from the position of the runway within an image taken during the landing phase of an aircraft. We particularly focus on the sub-task that consists of detecting the runway in the image.

As no open-source use cases (and no dataset) were available at the beginning of our work, we first had to define what a visual-based landing system should be and how machine learning could help. This has led us to develop the Landing Approach Runway Detection (LARD) open-source dataset [9]¹. Among the important features of this dataset, we can cite the capacity to generate easily new data thanks first to synthetic data generators based on Google Earth Studio and Microsoft Flight Simulator, and second the selection of Youtube channels (such as ²) with real landing footage video from which we can label new data easily. The Figure 1 reproduces an image recorded during a flight and with our generators. Although the weather conditions differ between the images, we note a great similarity in the runway's environment.

C. Approach

We propose a preliminary approach to apply the aeronautical certification guidelines. We have drawn an overview of our interpretation of the ED-324/ARP6983 workflow, see figure

¹<https://github.com/deel-ai/LARD>

²<https://www.youtube.com/user/TheGreatFlyer>

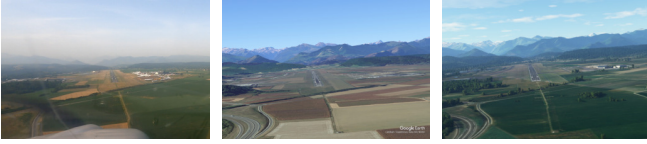


Fig. 1: Illustration of the quality of the synthetic images - Comparison of a real landing footage (left) with synthetic replicas (Google Earth Studio center, Microsoft Flight Simulator right)

2, to design a dataset from a system-level ODD provided by the ED-79/ARP-4754 and the system CONOPS. For the VBL system, the system-level ODD, presented in section II, consists of the landing approach geometry of an aircraft and the standardised runway markings. The ED-324/ARP6983 introduces an intermediate level of engineering (called ML Constituent – MLC) between the system and item layers. From a system perspective, the MLC is an item (or a container of items). The MLC contains at least one ML model and its implementation should provide the necessary items to support the ML model(s) inference. The VBL constituent, presented in section III, contains three stages, among which an object detection stage for which we design the dataset.

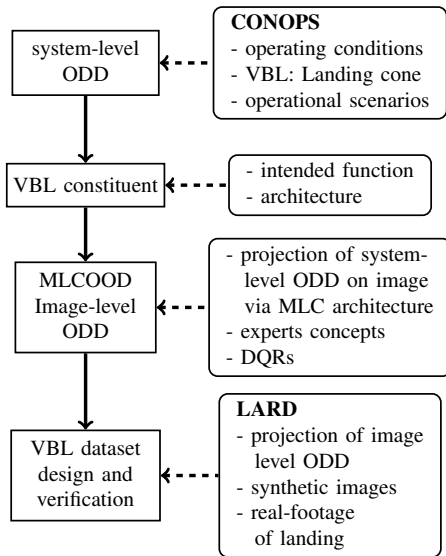


Fig. 2: ODD Design Workflow

The guidance advocates a refinement of the ODD that starts at the system level with the definition of the operating environment of the VBL system (landing approach conditions) and continues at the MLC level (MLCODD or image-level ODD in this use case) with some expected properties on the ML tasks. This activity is highly complex since we have to project the geometry of a landing on the possible images observed by the camera. We propose 3 activities, see section IV, to define the image-level ODD: 1) the geometric projection of a landing on the image; 2) the definition with the help of domain experts of *expert concepts*; and finally 3) the definition of *Data Quality Requirements* (DQRs) associated to

the VBL. These DQRs are imposed by the ED-324/ARP6983 as a result of the ODD characterization activity. Based on this image-level ODD, we developed a strategy for constructing the dataset to encompass those DQRs and we evaluated the compliance of the LARD dataset on some of them. The LARD dataset definition and evaluation are detailed in section V. Note that the workflow is highly iterative. Indeed, depending of the ability to refine the ODD into image-level ODD or the capacity of the dataset to comply with the requirements, it may be necessary to revise the system-level ODD or the ML constituent architecture.

II. SYSTEM-LEVEL ODD

The system-level ODD regroups the requirements that must flow from the ED-79/ARP-4754 and the specific *Concept of Operations* (CONOPS) of the system under development down to the dataset and model designs.

A. ODD concept

The concept of Operational Design Domain (ODD) originated in the automotive industry as a way to define the specific operating conditions under which automated driving systems are designed to function. The ODD concept was first introduced in the SAE (Society of Automotive Engineers) J3016 standard [10] to define levels of driving automation for on-road motor vehicles. Their definition of ODD was "*the operating conditions under which a given driving automation system or feature thereof is specifically designed to function, including, but not limited to, environmental, geographical, and time-of-day restrictions, and/or the requisite presence or absence of certain traffic or roadway characteristics*". The ISO-21448/SOTIF (Safety Of The Intended Functionality) standard [11] focuses on the safety considerations within automotive autonomous vehicles and directly integrates the concept of ODD from the SAE J3061 standard.

A more recent automotive standard, the ISO 34503 [12] proposes some concepts and requirements to enable the definition of an ODD for an automated driving system. The document, in particular, distinguishes the ODD and the *Target Operational Domain* (TOD) that refers to the real-world conditions that a system may encounter. As the TOD is not *specifiable*, it can be seen as a superset of the ODD. It is up to the system design to specify the *optimal ODD* to be as close as possible to the TOD, but it is also their responsibility to ensure that the system is used on the ODD solely and deactivated otherwise. The standard also promotes the definition of *operational scenarios* on which the safety assessment should rely to evaluate the final system.

For aeronautical applications, the European Union Aviation Safety Agency (EASA) has adapted the ODD concept from SAE J3016. The ODD is defined in EASA Artificial Intelligence Concept Paper Issue 2 [5] as "*the operating conditions under which a given AI/ML constituent is specifically designed to function as intended, including but not limited to environmental, geographical, and/or time-of-day restrictions*". While the EASA definition is similar to the

SAE J3016 definition, it differs in that it applies specifically to AI/ML constituents within a larger system, rather than to the system as a whole. This reflects the importance of defining constraints and requirements on the data used during the learning process, implementation, and inference in operations for AI/ML constituents. Nonetheless, the ODD concept remains an important tool for ensuring the safety and reliability of automated systems in both aeronautical and automotive applications.

B. ED-324/ARP6983 – operating environment

The data management requires an upstream process at the system level of engineering to define the operating environment of the VBL system. This definition is developed from the expertise of *subject matter experts* (SMEs) who have a deep knowledge of the Concept of Operations (CONOPS) and who can define the operational envelope of the system, i.e. the system operating conditions and environment where the system is supposed to operate correctly. An accurate definition of the operating environment is a prerequisite for AI scientists to start the data management process and to define the MLCODD (image-level ODD).

C. VBL system-level ODD

Defining such an ODD is highly complex [13] and of vital importance. For a vision-based system, it details in particular the environmental and weather conditions (e.g. temperature, wind, visibility, precipitation, types of sensor noise); operational terrain (e.g., runway slope, runway roughness); operational infrastructure (e.g. fixed obstacles) and many other information. Such a list could be infinite depending on the level of details. Making this problem tractable in practice is generally accomplished by constraining the operational environment to a subset of all possible situations that could be dealt with by a human.

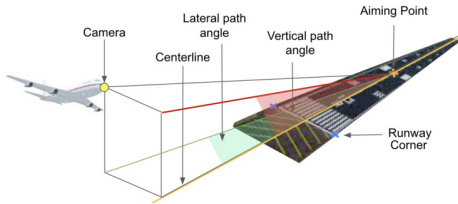


Fig. 3: Geometry of a landing

The application is designed for civil aircraft landings. Therefore, we start by defining a *generic landing approach cone* based on the documentation provided by aeronautical standards. Figure 3 illustrates the different positions / angles / distances / markings involved in the geometric description of a landing. Runway markings are standardized [14] and appear in most cases as follows: a first line at the start of the runway, called *landing threshold*, represents the underline limit of the runway. It is usually followed by a pattern of stripes (the *piano*) and then the runway identifiers. The target of an aircraft during landing is the *Aiming Point*, located 300

Parameter	range
Along track distance	[0.08, 3] NM
Vertical path angle	[-2.2, -3.8]°
Lateral path angle	[- 4, 4] °
Yaw	[-10,10] °
Pitch	[-8,0] °
Roll	[-10,10] °

TABLE I: Parameters of the generic landing approach cone

meters beyond the landing threshold, between two rectangular markings visible on each side of the runway *centerline*³.

The position of the aircraft with respect to the runway is defined by 3 parameters: the along-track distance which corresponds to the distance between the projection of the aircraft nose on the centerline of the runway (on the ground) and the Aiming Point. The lateral (resp. vertical) path angle which corresponds to the angle formed by the centerline and the line defined by the Aiming Point and the plane nose projection on the ground (resp. plane orthogonal to the ground going through the centerline). On the other hand, the attitude of the aircraft is defined by its rotation angles (denoted respectively as pitch, roll, yaw). The yaw angle is relative to the runway heading⁴ whereas pitch and roll are relative to the horizontal plane.

These 6 parameters allow to define a generic landing approach cone (Definition II-C) corresponding to a realistic aircraft trajectory during landing, as well as an envelope for the aircraft attitude that encompasses typical aircraft orientations during approaches on a runway.

Definition 1 (Generic landing approach cone): A generic landing approach cone is the set of all pairs (positions, attitude) within the ranges of the 6 parameters of Table I.

In addition to the approach cone specification, it is also relevant to define some *operational scenarios* that describe some usual trajectories observed in the real world. Such scenarios can represent complex landing situations (e.g. crab and de-crab manoeuvres in the presence of wind) or can be constructed by collecting real traffic observations [15], [16]. These scenarios will help in assessing the performance reached by of the ML constituent and the safety of the complete system.

Operational Design Domain 1 (of VBL): The VBL system must permit the landing as long as the aircraft is in the generic landing approach cone.

III. VBL ML CONSTITUENT

The constraints of the system-level ODD expressed on the Operating Environment must be propagated to the components of the system. It is then mandatory to specify the ML constituent architecture and its associated intended function. The VBL constituent is expected to realise the following intended function:

Intended function 1 (VBL intended function): The *intended function* is the pose estimation of the aircraft with respect to

³An imaginary line going through the middle of the runway

⁴For instance a yaw of 0° indicates that the aircraft faces directly the runway, regardless of the runway orientation.

the airport runway when the aircraft flies within the generic landing approach cone. The pose is estimated from several sensors, including a camera positioned at the aircraft’s nose and directly facing the runway during the landing.

The ML constituent architecture should fulfil the intended function. The one we propose is directly inspired from the 3-stage architecture of Daedalaen AG [17] as shown in Figure 4. The first stage is based on an object detection step that is in charge of computing a bounding box around the detected runway. The image is then cropped around the bounding box and a second stage is in charge of computing the 4 corners of the runway. From this identification, the pose estimation of the aircraft is done with a non-ML approach by the last stage.

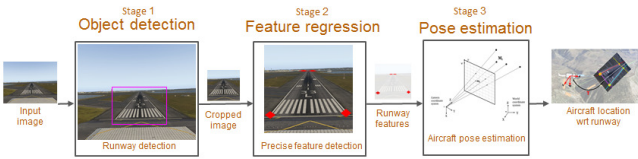


Fig. 4: VBL constituent architecture with 3 stages

Subsequently, we solely focus on the first stage of the architecture that we call the *VBRD* (for Vision-Based Runway Detection). This component comes with its own intended function and its associated performances. The contribution of all MLC components intended functions and their associated performances should ensure the MLC intended function and should fulfil the system requirements. How to derive the requirements on each component and how to define the rules to combine the contributions of each component is out of scope of the paper. These activities are non trivial and should benefit from a meticulous work.

The *VBRD* is a pure ML component and is in charge of realising an *ML object detection task*. The family of object detection models covers a large range of applications and offers much more capacities than we need. Indeed, such a model must be able to detect several types of objects (e.g. pedestrians, vehicles) and several objects appearing in a high resolution input image. In our case, the model should identify one or more runways and as a consequence, there is a unique class (runway). Moreover, we expect the output of the stage-1 to select the runway on which the aircraft must land. An object detector, without any further support, will hardly be capable of selecting the target runway, in particular when there are parallel runways. To simplify the task, we decided to restrict the intended function and thus the ODD to have a unique runway on the image.

Operational Design Domain 2 (of VBL): We restrict the operating conditions of the intended function 1.

- 1) The aircraft is landing on airports with a *piano*;
- 2) There exists only one runway for which current position is considered within the approach cone⁵;
- 3) The runway is fully visible on the image (no occlusion).

⁵Another runway can still be visible, but the aircraft should not be in its approach cone

Such restriction must be fed back to the system level for negotiation and update on the system architecture / operating conditions. In terms of object detection, the model belongs then to the category of single class single object detection.

Task 1 (Single object detection task): The detector must localize the object within the image by providing *bounding box* candidates surrounding said object. An *acceptable* bounding box should include the complete runway with a margin of x pixels on each side (top, bottom, left, right).

The *VBRD* takes as input an image and outputs a bounding box around the runway it contains. Therefore, propagating the system-level ODD down to the components consists of characterising the appropriate constraints at the image level. At this stage, we can only define a very basic and un-exploitable image-level ODD a first *characterisation of the ML constituent restricted to the object detection task ODD*.

Operational Design Domain 3 (of task 1): The ODD is the infinite set of all images that could be seen during a landing on an (extremely large) set of airport runways.

IV. IMAGE-LEVEL ODD

To define properly an exploitable image-level ODD, we need to make more activities.

A. ED-324/ARP6983 – MLCODD Characterization and validation

The MLCODD is defined from the operating environment identified at the system level to specify all foreseeable operating conditions under which the MLC is expected to work. Roughly speaking, we can see the MLCODD as a set of *parameters* (or *features*). For instance, a *parameter* could be the weather conditions selected at system-level (e.g. the *VBL* should function correctly from -5°C up to 40°C in the presence of mist or in a perfect sunny day). The parameter must then be translated as an image-level *parameter*. For instance, the weather conditions *parameter* can be translated into several parameters such as contrast or brightness on the images. However, this translation into MLC inputs must be supported by some rationale and its impact on system-level parameters should also be examined. The MLCODD characterisation is instrumental in specifying the inputs of the data management process, i.e. the capture of all the requirements necessary to produce and verify the dataset.

In any case, *parameters* must be representable with a recognised nomenclature and understandable by a human. This entails that parameters must belong to well-typed elements (e.g. continuous parameters, set of nominal values for discrete/categorical parameters). After this preliminary identification of parameters, it may be that the number of image-level parameters is too high to be tractable. In that case, the MLCODD is refined by identifying potential interdependence between the parameters and applying a reduction strategy on the parameters to reduce the complexity and the dimension of the MLCODD. In this paper, we consider geometric strategies and expert concepts identification to identify the parameters.

In addition, the MLCODD parameters are also characterised by some Data Quality Requirements (DQRs). Such a DQR can specify some ranges of values and a distribution where applicable. Regarding the example of the weather conditions *parameter*, contrast or brightness parameters must be defined with a reachable range covering all the supported weather conditions and distribution among the range. Such a distribution can be conditioned on the airport (e.g. Toulouse and Montreal airport weather conditions distributions differ). In this paper, we consider a subset of identified DQRs: Source Suitability, Completeness, Representativeness and Accuracy. These properties will be detailed in Section IV-B3.

The ODD is not only limited to nominal situations. Indeed, the system must ensure safe behaviour in all foreseeable situations. As a result, the ODD must encompass more general cases. The ED-324/ARP6983 has defined its own taxonomy of data types for non-nominal data (outliers, edge/corner case, singularity, novelty) that should be considered with the appropriate stopping criteria.

B. Approach to design an image-level ODD

In addition to the ED-324/ARP6983 considerations, it is worth looking at other works in the literature and other domain existing standards to help the designer in such a complex activity. For instance, the ISO 34503 [12] encourages the designer to consider, in addition to parameters mentioned previously, *elements* that correspond to main parts of an image. There are 2 categories of elements:

- *scenery elements* that refer to the spatially fixed elements of the operating environment relative to the aircraft;
- *dynamic elements* that refers to moving elements (e.g. other aircraft).

To define the *parameters* and the *elements* of the VBRD, we propose an approach based on 3 activities, that is generic enough to be applied to other object detection ML constituent.

1) **Geometry parameters:** For now, the only usable constraints from the system level are the ones expressed with precise ranges on the geometry of the landing, represented by the definition of the *Generic landing approach cone* see section II-C. The majority of the constraints on the image space will, therefore, be related to the position and attitude of the aircraft, but the focal length⁶ of the camera will have to be taken into account as well. Thus, using the geometry of the landing, we can derive the possible positions of the runway on the image space. This activity should be supported by image processing methods. The book [18] recalls the basics of geometry for images and is a good basis for deriving some properties of the position/shape and other geometric considerations of the runway (or any other scenery element) depending on the range of attitude of the aircraft. Among the transformations, we can cite the projection from the real world to the image-based coordinate system, which is done using two standard matrices [19]:

⁶The resolution of the image and the expected position of the runway will depend on this parameter combined with the distance to the runway.

- The Extrinsic matrix whose role is to get the coordinates of the corners in the camera-centered coordinate system.
- The Intrinsic matrix whose role is to project the 3D coordinates expressed in the camera-centered coordinate system into the 2D image.

2) **Domain-specific concepts:** Human making-decision process on an image relies on the identification of *concepts*. We propose the following partitioning of concepts, depending on their utility and relevance to the task of object detection:

- **Primary concepts:** refer to elements (or landmarks) that are considered fundamental by a human for fulfilling the task. The absence of only one of the primary concepts would imply that the object considered is not an object of interest. For the detection of a runway, we can typically consider the shape of the runway (the typical 4-sided polygon), the clear demarcation with the external area, and the main markings (the target, the piano, the runway number)⁷.
- **Secondary concepts:** refer to elements that may reinforce a decision, but the absence of which is not prohibitive in the identification of the object. For runway detection, we consider the surrounding elements such as the airport traffic control tower, secondary markings which are not always present (for instance the displaced thresholds), other surrounding runways, other aircraft in parking phase, taxiways parallel to the runway, etc...
- **Tertiary concepts:** The presence or the absence of the elements considered in this category should not have any impact on the detection of an object. We can identify here the colours of the areas surrounding the runway, due to vegetation or seasonal changes, as well as the environment around the airport itself, such as the presence of buildings, mountains or water bodies, etc...

A first analysis of these concepts highlighted that they could be different according to the distance to the runway. Indeed, the details on the image are not necessarily equivalent when the runway is a few kilometres away and when it is seen from a few hundred meters. In the first case, we may rely on secondary concepts like the overall airport and the traffic control tower, and for detecting the runway, we will rely in priority on the typical geometric shape of the runway and its visible markings (the target for instance). In the second case, we may consider the details of the markings, like the piano and the runway number, as well as parts of the secondary concepts like the markings on the surrounding taxiways.

We want to point out that these defined concepts could be used and extended in the validation phase of the Model. Indeed, if we consider a correct detection from an ML model, it is possible to use explainability methods such as [20], [21] to identify concepts which were used for this decision. These concepts can be categorised using the partition presented above as follows:

- The concept belongs to the category of primary concept, and it should be added to the list of primary concepts already

⁷This will be challenged when considering the distance at which the runway may be detected.

identified.

- The concept corresponds to a secondary one, which may or may not be used by humans. For instance, for detecting runways, we identified aircraft tyre marks as a valuable indication that is, consciously or not, used by humans when attempting the same task. However, since this concept is secondary, it is crucial that the model remains robust to its removal or its absence, as all runways are not heavily marked by tyres. Another example would be the difference in colours between the runway and its surroundings, or the possible texture of the runways, which can hardly be seen with the naked eye.
- The concept belongs to the tertiary ones, which means that it should not have been used for the decision-making process. In that case, it undoubtedly represents a bias of the model that should be eliminated, possibly by making changes to the training data, or by working on the retraining of the model.

For now, we only mentioned the correct detections of the ML model. Besides, in an ideal world, the list of primary concepts should be precise enough so that the absence of one concept would allow us to discriminate a true positive from a potential false positive. However, as mentioned earlier, this is not always true, typically due to the variability in the distance to the runway, which may lead to some runway features disappearing. Indeed, if we compare a highway segment to a runway from a relatively short distance, certain primary concepts such as geometry and shape, as well as the clear demarcation with the outer area are present, but the markings are very different. Nevertheless, at a certain distance, these markings are likely to be invisible, making it hardly possible to rely solely on these primary concepts to distinguish true positive detections from false positive ones. In that case, we can consider the problem in reverse: the false positive observed in the decision of a model could help us build a fourth list of concepts corresponding to well-identified biases leading to these potentially incorrect decisions.

Quaternary concepts: For objects identified as false positives by a model, we could include in the fourth list the set of elements that are primary concepts for this specific object (e.g. a highway) but not for our object or interest (e.g. a runway). For a highway, we could typically add the presence of cars, a central road divider, road markings, traffic signs, etc... This fourth list of concepts will, therefore, correspond to the elements which, if they are present, invalidate a detection.

3) **DQR for VBL:** The guidelines [22] gives a set of recommendations to build and manipulate the datasets used to develop and/or validate machine learning models.

The Data source suitability [23] “*refers to the appropriateness and relevance of a data source for a specific purpose or context, particularly in relation to its ability to provide data satisfying specified data quality attributes for a given task or analysis.*”

DQR 1 (Data source suitability for VBRD): This property is critical for the choice of generator of synthetic images which should be compared to the images captured by a camera in

real conditions.

Completeness [23] is “*the extent to which a dataset covers, according to the specified criteria, the ODD for the intended application.*”

DQR 2 (Completeness): In our case, guarantees should be provided regarding the coverage of the operating conditions and the operational scenarios defined in system-level ODD.

Representativeness [23] states that “*a dataset is representative if it covers the full ranges of the parameters that define the ODD and the distribution of each parameter matches the specified distribution.*”

DQR 3 (Representativeness): This crucial property should motivate extensive testing of the data distribution regarding each parameter of the image-level ODD.

Accuracy/Correctness [23] “*Measures the faithfulness to the real value and depends on data gathering/generation. It also measures the degree of ambiguity of the representation of the information.*”

DQR 4 (Accuracy/Correctness): In our case, this is highly related to the choice of labels and their precision for the task that must be fulfilled.

These requirements are not guidelines to produce a dataset but must be kept in mind when designing it, ensuring a successful verification in later stages of the process.

V. DESIGN OF THE DATASET

The objective is to design a dataset for the VBRD component compliant with image-level ODD defined by the activities of section IV-B.

A. ED-324/ARP6983 – Data management and verification processes

The objective of the *data management process* is to produce the dataset that matches the characterized ODD. The first activity of the data management process is to identify the sources of such data. Then data are collected, prepared and split into datasets in order to deliver trustworthy training, validation and test datasets which will be used to design, implement and integrate the ML inference model that meets the functional and operational requirements.

Once the ODD is defined, it should be validated. That is the purpose of the *ML data validation process* that is intended to provide assurance that the ODD and its DQRs are correct and complete with respect to the intended function supported by the MLC. The high quality of the datasets (and at least the test dataset for low critical applications) should be demonstrated. If this verification is not properly performed, then the trained model might exhibit unintended behaviour (e.g. make incorrect decisions, fail to generalize to new or unseen situations) that could be detrimental to its intended use and/or the safety objectives that have been assessed at system level. To this purpose, the ED-324/ARP6984 proposes several activities:

- **ODD/datasets bi-directional traceability** to guarantee that the complete ODD is covered and eliminate any undesirable data

- *Data quality analysis* to demonstrate the datasets compliance to the DQRs.

B. Strategy to generate the dataset

LARD is composed of both synthetic and real footage images. Synthetic images were generated via a generator pipeline presented in Figure 5. The two inputs (in gray) are the airport database and the configuration file to be filled by the user, setting which runway they want to generate images from and other parameters (e.g. number of images). Then, the first script (in white) generates a scenario file that can be provided as an input for the synthetic image environment (either Google Earth Studio or Microsoft Flight Simulator). This virtual globe tool can then generate the corresponding images, together with an information file (here in json format). Finally, the last module of our generator associates the 'labels' to each image, in particular the scaled position of the four corners on the picture. The output in gray contains the images, the labels and the metadata.

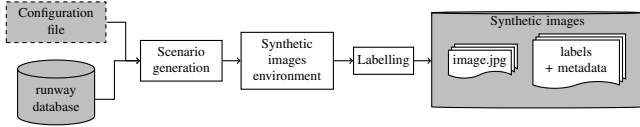


Fig. 5: Generator pipeline

Overall, this generator allows to produce an infinity of images with various camera angles and positions, where the annotation is automatically propagated, which drastically reduces the labelling cost. In parallel, we have access to numerous Youtube channels from which we can extract images and label them.

This framework allows us to generate the dataset for our intended function. The question that we will have to tackle is how to generate sufficient representative data that fit the *development of the data quality requirements*. As an example, the risk of having thousands of images per runway or more is the high similarity of resulting positions in the cone and the low independence between each image, which may lead to overfitting models, while collecting only a few dozen images per runway limits the possibility to encounter edge cases for each parameter and increases the need for manual annotation of runway corners to fulfil the high volume of data required.

C. Adequation between image-level ODD and DQRs

The verification of the DQRs is a fundamental step to provide a first estimation of the quality of the proposed dataset. We detail in the following the quality analysis performed on the data and the results of the verification activities performed for each of the requirements defined in Section IV-B3.

1) *Source Suitability*: The task targeted by the ML component is the detection of runways on images during landing. However, the cost of labelling real images in a sufficient volume for ML training is prohibitive, which leads us to choose a tool for generating synthetic images instead. We

selected Google Earth Studio, which supports trajectories of positions (defined within our landing approach cone) as input and allows us to produce a variety of high-quality images relatively close to reality. However this tool came with restrictions such as the absence of adverse weather simulation or realistic night images. As these constraints were not considered critical, we had to propagate them to the system-level ODD, producing a third refinement defined as follows:

Operational Design Domain 4 (of VBL): We further restrict the operating conditions of the intended function 1.

1) Optimal conditions: clear daylight and no adverse weather conditions (clouds, precipitations...).

The quality analysis presented in this paper were performed on this refinement of the ODD. However, in a recent extension of our generator, we integrated the capability to generate images with both Google Earth Studio and Microsoft Flight Simulator, making this last restriction obsolete. Figure 6 illustrates this comparison, while Figure 7 shows some image variability supported in Flight Simulator.

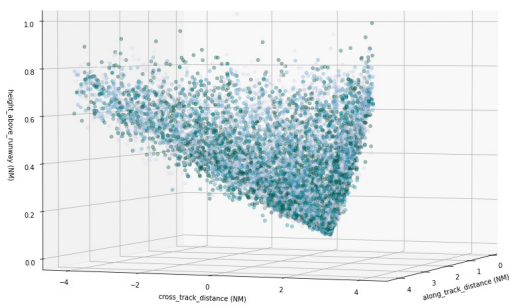


Fig. 6: Comparison between Google Earth Studio and Flight Simulator

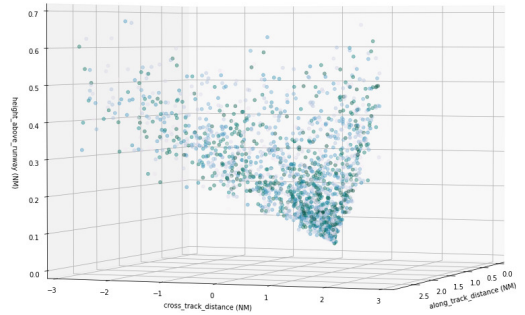
To fulfil DQR 1, another complex question is the quality and the representativity of the synthetic images vs real images that will be observed at operation. There should be rationale and verification activity to accept synthetic images in the dataset. This is considered as future work.

2) **Completeness**: Considering the ODD of the VBRD, associated with the intended function 1, an adequate dataset, i.e. that satisfies DQR 2, should not only cover a variety of airports all around the world but also span a wide range of positions inside the approach cone, to ensure a comprehensive coverage of all possible operational scenarios.

Coverage of landing scenarios: Figure 8 illustrates the distribution of aircraft positions in the training and the test set. In this figure, the z-axis corresponds to the *along track distance*, but the other two axes are also distances (*cross track distance* and *height above runway*), computed from the angles provided in Table I (*Lateral path angle* and *Vertical path angle*). For the training set in Figure 8a, the randomly sampled points span the whole approach cone corresponding to II-C.



(a) Training set



(b) Synthetic test set

Fig. 8: 3-dimensional visualisation of aircraft positions in the training set and the test set

Moreover, while the synthetic test set contains less data, it still covers a variety of positions in the cone, as illustrated in Figure 8b.

Airport distribution: Figure 9 plots the distribution of airports from all around the world which were used to build the LARD dataset. Indeed, obtaining a great variety of images is a fundamental aspect for verifying the generalization capabilities of the models, and current distribution of airports presents the following benefits: first, it ensures a diversity of runway visuals, with different surface types⁸ and various runway length, width and markings, even if the runway standardization reduces the variability for this aspect. Second, it allows for a variety of surrounding terrain and landscapes such as grass, snow, dirt, but also city architectures, water bodies or mountainous reliefs.

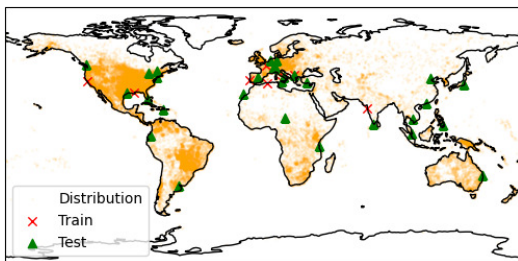


Fig. 9: Distribution of airports used for the training set and the test set

3) *Representativeness:* For each parameter considered in the definition of the image-level ODD, we need to ensure that the distribution of the corresponding image-related features is thoroughly verified, as required by DQR 3. We consider here that the test set is a faithful representation of our image-level ODD, and we compare the distribution of some of its features against the training set.

Runway center positions: The plot of runway centers positions of Figure 10 shows an even distribution both for the training set and for the test set, located primarily around the center of the images. Nevertheless, a large area in the top and

⁸Asphalt and concrete are typically used for runway surfaces

⁸Subset of the test set containing only images from real footage.

the bottom contain little to no points, which is the result of two main factors:

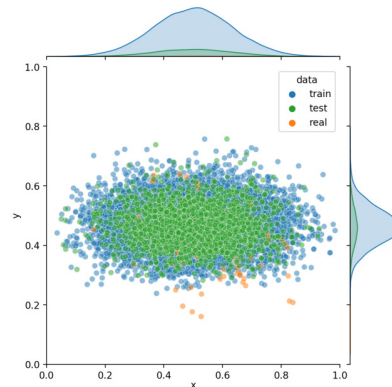


Fig. 10: Normalized positions of runway centers in train, test and real⁹ subsets.

(i) the presence of the watermark, which is expected to be removed from the images before usage by cropping 300 pixels from the top and the bottom of the pictures, and (ii) the ranges of the *pitch* parameter defined in the Table I which prevent the runway to appear at the very top or bottom of the image. Additionally, the real images of the test set appear to be slightly biased towards the bottom-right, which seems to result from the positions of the cameras in the cockpits.

Bounding box fill ratios: The aspect ratio of the objects bounding boxes is a sensitive aspect for a detection task, as elongated objects in one or the other direction may not exhibit recognizable features. Figure 11 illustrates the aspect ratio variability, and highlights how the majority of the bounding boxes in all three subsets have an aspect ratio between 0.5 and 1.5, indicating that most images are suitable for the targeted detection task.

The histograms of Figure 12 illustrate the relationships between the runways, their bounding boxes and the global images. Figure 12a shows comparable distribution for the training and the test set, where most of the runways fill between 20% and 80% of their bounding boxes. This also indicates that bounding boxes should in general contain enough

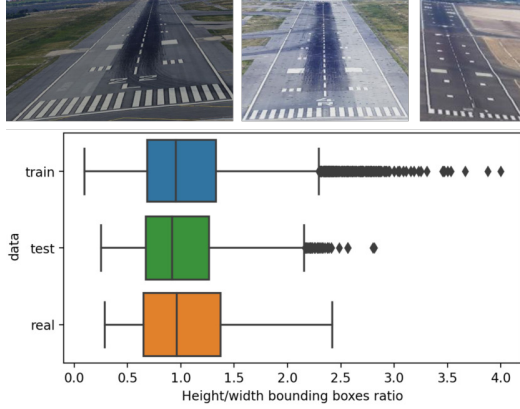


Fig. 11: Top - Illustration of different aspect ratios of the bounding boxes. Bottom - Distributions of bounding boxes height over width ratios for the train, test and real subsets.

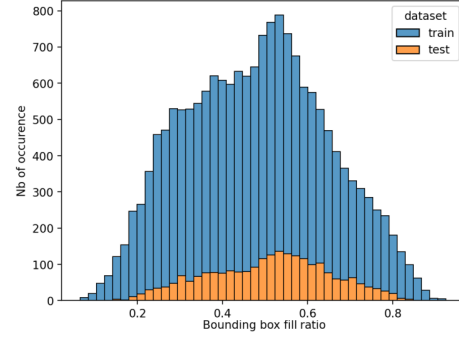
runways pixels for the detection task to be applicable and consistent. Additionally, Figure 12b, which illustrates how the areas of the bounding boxes cover the whole images, shows that the training set and the test set follow approximately the same distribution. This provides a certain level of guarantee that the bounding boxes will look similar between the training and the test set. Moreover, the figure shows that the vast majority of bounding boxes areas are over 25×25 pixels, which makes them large enough for a runway to be detected by humans. On the other hand, the dataset contains only a few examples of bounding boxes with large size, which may bias the learning process when the aircraft is close to the runway and should be further investigated.

Slant distance: The synthetic images and the real images do not contain the same metadata. The distance between the aircraft and the runway is given for synthetic images as the *slant distance*, however it is not available for real images, for which a value called *time to landing* is provided instead. This value can be used as a proxy for the distance to the runway, considering that planes have comparable speed during landing phase.

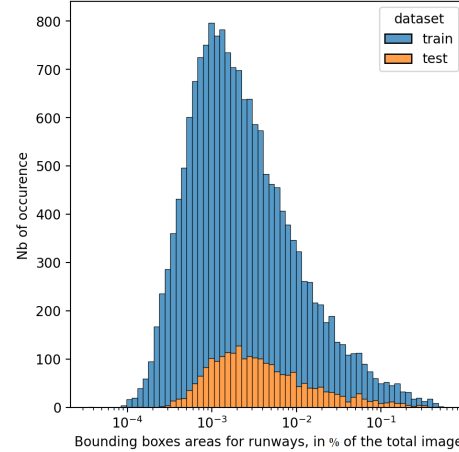
Figure 13 shows how the distributions of *slant distance* (for synthetic images) and *time to landing* (for real images) relate to each other¹⁰. It indicates that for both sources of data, the test set contains an important part of the images close to the runway while a non-negligible number of pictures were taken at longer distances from the runway, in a nearly evenly distributed manner, despite the limited number of real images.

4) **Accuracy/Correctness:** As specified by the ODD, the images of the dataset must always contain fully visible runways. Any label associated to an image should allow to define the runway inside of it in an unambiguous way whether the data is synthetic or real footage, according to DQR 4. There are several approaches for delimiting a runway, the

¹⁰Only the shapes of the distributions should be compared as the *slant distance* was re-scaled to fit the diagram



(a) Distribution of bounding box fill ratios (percentage of the bounding box that correspond to pixels belonging to the runway itself)



(b) Distribution of bounding boxes areas (areas in logarithmic scale)

Fig. 12: Comparison of bounding box characteristics between training and test sets

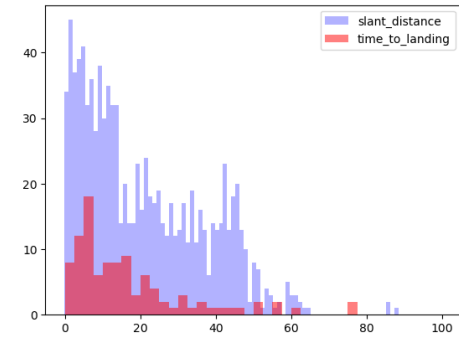


Fig. 13: Comparison of distance estimation between real images and synthetic images in the test set

most usual being contours, ground marking, corners, horizon line or any other semantics specific to a runway. We chose to encode the runway position by the pixel coordinates of its four corners in the image.

As pointed out in [24], representing the runway by its four corners poses some concerns such as instability in the presence of runway occlusion and sensitivity to the aircraft position estimation. In practice, none of the synthetic images (both in the training and the test sets) present situations of

corner occlusions, but they may occur in real images, typically when the camera is placed inside the cockpit. However, these drawbacks are outweighed by the advantages of the corner representation, as this approach is easily applicable to any image, does not require camera-related information (typically camera angles), and is compatible with both image detection and image segmentation approaches, two of the most widely used approaches for locating and identifying objects in image.

Finally, our labelling tool relies on the automatic projection of the runway corners onto the image, which ensures high precision of their position and directly satisfies the requirement for label accuracy.

VI. CONCLUSION

Considering a well-defined vision-based landing task, we have presented a comprehensive approach for designing the system-level ODD of this intended function. We then focus on a specific ML component of the system and refine the ODD at the image level. Using specific tooling based on Google Earth Studio, we illustrate the generation of a dataset designed to fulfil the targeted task. In the process, we establish a link with relevant standards relative to the introduction of ML-based systems in the industry. In particular, we focus on the definitions of Data Quality Requirements and show how they can be verified on the dataset defined for the intended function.

The process leading to the definition of the ODD and the design of the dataset is an iterative process that also benefits from the model design. Indeed, the verification of the model may exhibit some lacks in the dataset, e.g. insufficiency of images for certain airports or attitudes, that should be fed back to the data process management. We will investigate the whole process by pursuing the VBRD design following the ARP 6983 guidelines.

Moreover, our approach is complementary to [3], which defines the ODD by characterizing the distribution of samples it may contain (*in* or *out-of* ODD, edge-cases, corner cases...). We plan to leverage this formal definition to generate dedicated datasets for each of the parameters we identified in this paper, to investigate how the coverage of the ODD can be ensured.

Finally, the quality of a model solely trained on synthetic data must be carefully estimated. The addition of multiple data sources and more image variability should help us address this complex question in the future, but it would require a clear methodology that has yet to be written.

ACKNOWLEDGEMENTS

This work has benefited from the AI Interdisciplinary Institute ANITI, funded by the “Investing for the Future – PIA3” program Grant agreement ANR-19-P3IA-0004 and from the PHYDIAS 2 project funded by the French government through the France Relance program, based on the funding from and by the European Union through the NextGenerationEU program. The authors gratefully acknowledge the support of the DEEL project¹¹.

¹¹<https://www.deel.ai>

REFERENCES

- [1] Christophe Gabreau, Beatrice Pesquet-Popescu, and Fateh Kaakai. EUROCAE WG114 – SAE G34: a joint standardization initiative to support Artificial Intelligence revolution in aeronautics, 2023. Keynote of SafeAI – https://safeai.webs.upv.es/wp-content/uploads/2021/02/AAAI_Keynote_202102_v1.0.pdf.
- [2] Christophe Gabreau, Adrien Gauffriau, Florence De Grancey, Jean-Brice Ginestet, and Claire Pagetti. Toward the certification of safety-related systems using ML techniques: the ACAS-Xu experience. In *11th European Congress on Embedded Real Time Software and Systems (ERTS 2022)*, Toulouse, France, June 2022.
- [3] Fateh Kaakai, Shridhar "Shreeder" Adibhatla, Ganesh Pai, and Emmanuelle Escorihuela. Data-centric operational design domain characterization for machine learning-based aeronautical products, 2023.
- [4] EASA. Concept Paper: First usable guidance for Level 1 machine learning applications - Proposed Issue 01, 2021.
- [5] EASA. Concept paper: First usable guidance for Level 1 & 2 machine learning applications - Proposed Issue 02, 2023.
- [6] SAE/EUROCAE. Aerospace Recommended Practices ARP4754a/ED-79A- Development of Civil Aircraft and Systems, 2010.
- [7] RTCA/EUROCAE. DO-178C/ED-12C - Software Considerations in Airborne Systems and Equipment Certification, 2011.
- [8] RTCA/EUROCAE. DO-254/ED-80 - Design Assurance Guidance For Airborne Electronic Hardware, 2000.
- [9] Mélanie Ducoffe, Maxime Carrere, Léo Féliers, Adrien Gauffriau, Vincent Musso, Claire Pagetti, and Thierry Sammour. LARD - Landing Approach Runway Detection - Dataset for Vision Based Landing. working paper or preprint, April 2023.
- [10] SAE. J3016 Levels of Automated Driving, 2019.
- [11] ISO. ISO 21448:2022. Road vehicles — Safety of the intended functionality, 2022.
- [12] ISO. ISO 34503:2023. Road Vehicles — Test scenarios for automated driving systems — Specification for operational design domain, 2023.
- [13] Philip Koopman and Frank Fratrick. How many operational design domains, objects, and events? In *SafeAI@AAAI*, 2019.
- [14] FAA. Airport Marking Aids and Signs. https://www.faa.gov/air_traffic/publications/atpubs/aim_html/chap2_section_3.html, 2022.
- [15] Matthias Schäfer, Xavier Olive, Martin Strohmeier, Matthew Smith, Ivan Martinovic, and Vincent Lenders. Opensky report 2019: Analysing tcas in the real world using big data. In *2019 IEEE/AIAA 38th Digital Avionics Systems Conference (DASC)*, 2019.
- [16] Xavier Olive and Pierre Bieber. Quantitative assessments of runway excursion precursors using mode s data. *arXiv preprint arXiv:1903.11964*, 2019.
- [17] Daedalaen AG. Neural Network Based Runway Landing Guidance for General Aviation Autoland, 2022.
- [18] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.
- [19] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Nature, 2022.
- [20] Thomas Fel, Agustin Picard, Louis Bethune, Thibaut Boissin, David Vigouroux, Julien Colin, Rémi Cadène, and Thomas Serre. Craft: Concept recursive activation factorization for explainability. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2711–2721, 2023.
- [21] Amirata Ghorbani, James Wexler, James Y Zou, and Been Kim. Towards automatic concept-based explanations. *Advances in neural information processing systems*, 32, 2019.
- [22] Cyril Cappi, Camille Chapdelaine, Laurent Gardes, Eric Jenn, Baptiste Lefèvre, Sylvaine Picard, and Thomas Soumarmon. Dataset definition standard (DDS). 2021.
- [23] EUROCAE WG-114/SAE joint group. ED 327/ARP 6983 – Certification/approval of aeronautical systems based on AI, 2023. on going standardization.
- [24] Giovanni Balduzzi, Martino Ferrari Bravo, Anna Chernova, Calin Cruceru, Luuk van Dijk, Peter de Lange, Juan Jerez, Nathanaël Koehler, Mathias Koerner, Corentin Perret-Gentil, et al. Neural network based runway landing guidance for general aviation autoland. Technical report, United States. Department of Transportation. Federal Aviation Administration ..., 2021.

Authors index

- Abella, Jaume, 197, 305
Alcaide, Sergi, 197
Alecú, Lucian, 159
Allouche, Theo, 349
Amari, Youssef, 79
Armengaud, Eric, 129
Arora, Jatin, 45
Auriol, Guillaume, 113
Azil, Azzedine, 401
- Bailleul, Quentin, 67
Barbero, Massimo, 67
Baufreton, Philippe, 135
Becker, Christian, 371
Begueret, Jean-Baptiste, 109
Belcaid, Mohammed, 467
Berne, Alexandre, 173
Berrouyne, Imad, 95
Bitterlich, Thomas, 99
Bois, Guy, 11
Boniol, Frédéric, 183, 327
Bonnafous, Eric, 467
Bonnin, Hugues, 159
Bouabdallah, Salim, 371
Boudiaf, Myrhal, 173
Bouzidi, Halima, 349
Boyer, Marc, 59
Brando, Axel, 305
Brau, Guillaume, 327
Brunel, Julien, 59, 217
Bucaioni, Alessio, 95
Butcher, Paul, 277
- Cappi, Cyril, 477
Caro, Martí, 305
Casanova, Raimon, 197
Chabor, Damien, 135
Chabrol, Damien, 255
Chaine, Pierre-Julien, 59
Chaudron, Jean-Baptiste, 23
- Chenevier, Florent, 231, 243
Ciccozzi, Federico, 95
Cohen, Noémie, 477
Comar, Cyrille, 135
Conejo Laguna, Ramon, 243
Conejo, Ramon, 231
Conejo-Laguna, Ramon, 349
Couderc, Matthieu, 11
Courty, Emmanuel, 327
Creff, Stephen, 231
Crison, Louis, 467
Cuenot, Philippe, 67
- Dalmau, Joseba, 159
Dausend, Daniel, 83
De Albuquerque Silva, Iryna, 315
de Grancey, Florence, 159
De-Grancey, Florence, 79
Delmas, Kevin, 159, 183, 217, 327
Deloye, Lucas, 435
Dimov, Rosen, 129
Dissaux, Pierre, 455
Dubrulle, Paul, 173
Dubé, Sébastien, 445
Ducamp, Mathilde, 423
Ducoffe, Mélanie, 477
Dufour, Jean-Louis, 389
- El Abbazi, Adil, 109
Engelsberger, Maximilian, 99
Esper, Alexandre, 45
Essoussi, Mohamed-Habib, 435
- Farges, Jean-Loup, 231, 243
Faure, Christophe, 467
Ferdinand, Christian, 359
Fernandes Pires, Anthony, 217, 231, 243
Fischer, Lucas, 147
Föhler, Gerhard, 265
Frauenschläger, Tobias, 291

Fruchard, Damien, 67
 Fuentes, Francisco, 197

 Gabreau, Christophe, 147, 477
 Galtier, Florent, 113
 Gardes, Laurent, 477
 Gassino, Jean, 135
 Gatard, Jacques, 23
 Gatti, Marc, 109
 Gauffriau, Adrien, 183, 315, 477
 Gauthier, Jean-Marie, 445
 Gebhard, Gernot, 359
 Gerchinovitz, Sébastien, 159
 Ginestet, Jean-Brice, 477
 Guerard, Hubert, 11
 Guyomarc'H, Jean, 255

 H. Hammadeh, Zain A., 35
 Hamelin, Etienne, 173
 Hammer, Pascal, 207
 Heller, Peter, 103
 Hernandez, Armando, 401
 Hiebert, Tina, 379
 Hlimi, Omar, 11, 349
 Hugues, Jerome, 455
 Hötzer, Johannes, 83

 Id Messaoud, Yassir, 243
 Id-Messaoud, Yassir, 231
 Ito, Yasumi, 129

 Jass, Philipp, 379
 Jenn, Eric, 11, 147, 231, 243, 255, 327, 349, 467
 Jose, Jithin, 337
 Jungmann, Lea, 35

 Kieffer, Hugo, 11
 King, Daniel, 277
 Kliemann, Johannes, 277
 Krause, Veronika, 207
 Kretzschmar, Valentin, 59
 Kumar, Ashish, 125
 Kästner, Daniel, 359

 Latombe, Florent, 231
 Le Bihan, Soazig, 109
 Ledier, Thierry, 67
 Ledinot, Emmanuel, 135
 Lemesle, Augustin, 147
 Lesage, Benjamin, 183
 Lesens, David, 423
 Leucker, Martin, 359
 Louis-Stanislas, Cyril, 349
 Lönn, Henrik, 95
 Lüdtke, Daniel, 35

 Machrouh, Joseph, 135
 Mackay, Adam, 411
 Mader, Ralph, 337
 Maia, Luiz, 265
 Maidl, Julian, 83
 Mamalet, Franck, 159, 477
 Marabotto, Christophe, 349
 Mascarenas-Gonzalez, Alfonso, 183
 Mekki-Mokhtar, Amina, 135
 Menga, Nicolas, 11
 Mihalache, Alin, 401
 Mottok, Juergen, 119
 Mottok, Jürgen, 103, 207
 Muddaiah, Arjun, 125
 Mussot, Vincent, 231, 243, 477
 Mynar, Zbynek, 125

 Nelissen, Geoffrey, 45
 Niar, Smail, 349
 Nicomette, Vincent, 113
 Novak, Thomas, 129

 Ojeda, Mirna, 445
 Olivier, Paul L. R., 113

 Pagetti, Claire, 183, 315, 467, 477
 Patrick, Markl, 83
 Phavorin, Guillaume, 255
 Pientka, Grit, 99
 Pister, Markus, 359
 Plettig, Gustav, 379
 Pletzer, Benedikt, 119
 Posso, Julien, 11
 Potop Butucaru, Dumitru, 147
 Probst, Tobias, 207
 Prof. Dr. Mottok, Jürgen, 291

 Quelin, Jean-Marie, 371
 Quere, Philippe, 135

 Ranoarivony, Philippe, 423
 Rebeiro, Chester, 337
 Renganathan, Karthikeyan, 337
 Roux, Pierre, 59
 Roy, Matthieu, 79

 Sachenbacher, Martin, 359
 Sakaki, Yayoi, 129
 Sans, Mariano, 89
 Schmutzler, Christoph, 83
 Schulz, Albert, 359
 Serratrice, Franck, 135
 Singh, Nikhilesh, 337
 Siron, Fabien, 255
 Sommer, Jan, 35

Sorniotti, Aldo, 129
Sousa Pinto, Davi Henrique, 435

Tarris, Sébastien, 11
Tembo-Mouafo, Serge, 349
Teulieres, Marie-Charlotte, 147
Thiant, Floris, 147, 349
Thierion, Philippe, 349
Thomas, Carsten, 379
Thompson, Sam, 255
Thurieu, François, 255
Todorov, Vassil, 401
Tovar, Eduardo, 45
Traub, Matthias, 83
Tristan, Dubois, 109
Turki, Mariem, 147

Vigouroux, David, 477
Vivot, Paul, 435

Waeselynck, Hélène, 79
Wegener, Simon, 359
Weiss, Alexander, 359

