PHYLOG certification methodology: a sane way to embed multi-core processors

Frédéric Boniol, Youcef Bouchebaba, Julien Brunel, Kevin Delmas, Thomas Loquen, Alfonso Mascarenas Gonzalez, Claire Pagetti, Thomas Polacsek, Nathanaël Sensfelder

PHYLOG certification methodology: a sane way to embed multi-core processors

Frédéric Boniol, Youcef Bouchebaba, Julien Brunel, Kevin Delmas, Thomas Loquen,
Alfonso Mascarenas Gonzalez, Claire Pagetti, Thomas Polacsek, Nathanaël Sensfelder
ONERA-Toulouse, France

***Abstract*—The PHYLOG project aims at offering a model-based software-aided certification framework for aeronautical systems based on multi/many-core architectures. Certifying such platforms will entail fulfilling the high level objectives of the MCP-CRI / CAST-32A position paper. To reach this general objective, we have defined a certification framework based on patterns to express any argumentation; as well as formal and automatic analyses to support the proof of the argumentation. In this paper, we will introduce the certification methodology and apply it on the KEYSTONE platform.**

I. INTRODUCTION

A. Context

Aeronautical safety critical systems are subject to *certification*, meaning that a *certification authority* assesses the compliance of the product with the regulatory requirements. An accepted means of compliance for an *applicant* is to show the compliance with a set of adequate standards. A way to show the compliance with a given standard is to prepare, for each high level objective identified by the standard, an *assurance case* which can be defined as "*an organized argument that a system is acceptable for its intended use with respect to specified concerns (such as safety, security, correctness)*" [RKR15]. Thus, the applicant provides a certain number of elements that go from design activities, organizational description, development process to V&V (Verification and Validation) operations.

For multi-core-based systems, several standards are applicable, such as the ARP 4754 [SAE10] or the DO 178 C [RTC11]. However, a specific position paper, named Multi-Core Certification Review Item (MCP-CRI) [EAS16] (also known as the CAST-32A position paper [Cer16]), has been written by aeronautic industries and certification authorities to provide a set of guidances for software planning and verification on those chips. PHYLOG aims at preparing certification activities related to the MCP-CRI / CAST-32A position paper. In particular, the purpose is to provide a framework that allows any applicant to prepare their certification assurance cases.

B. PHYLOG *objectives*

To reach this general objective, we have defined a certification framework based on:

1. argumentation patterns to express any argumentation. The idea is to organize any argumentation around structured graphical notations diagrams. We have translated most of the CAST-32A objectives to patterns, partly published in [BBB+18b], [BBB+19];

2. formal and automatic analyses to support the proof of the argumentation. A pattern comes with a series of *evidences* that support the reasoning. Among those evidences, two types of analysis are required by the MCP-CRI: *interference analysis* and *safety analysis*. We propose several automated analyses to help the applicant provide evidences.

C. Paper objective

The first objective of the paper is to present the overall PHYLOG framework, something which has never been done, as we have only presented pieces here and there. The second objective is to validate the approach by applying it to a real use case composed of an application executing on the KEYSTONE. The paper is organized as follows: in section II, we introduce the overall framework; we present the use case in section III and we apply the PHYLOG methodology to the use case section IV. For that, we have refined some former PHYLOG results.

II. PHYLOG FRAMEWORK OVERVIEW

The PHYLOG framework is schematized in figure 1. There are three inputs:

1) the MCP-CRI / CAST-32A standard, which is the starting point of the methodology, as the applicant must answer its 9 objectives;
2) the detailed design documentation that contains, in particular, the detailed configuration settings and how the multi-core is programmed (e.g. bare-metal, hypervisor);
3) some other external documents, such as the documentation of the multi-core hardware or the results of some external activities (e.g. FHA derived objectives for the applications hosted by the multi-core).

The framework itself contains two boxes: *argumentation* for the methodology and *model-based formal analyses* for the set of proposed analyses. We introduce both contributions in the next sub-sections.

A. Argumentation-based certification

Several papers have proposed high level strategies to tackle the MCP-CRI / CAST-32A. [AAAC17] has adapt the MCP-CRI certification objectives and has rephrased some of the requirements. [AMP19] proposes some leads to tackle safety issues. However, we are the first to define clear patterns for several objectives.

The applicant must fulfill the 9 MCP-CRI / CAST-32A objectives (P1, P2, RU1, RU2, RU3, RU4, S1, S2, and EH).

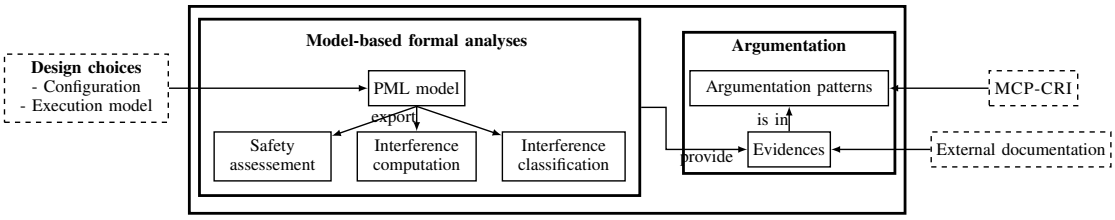

Fig. 1: Overview of the PHYLOG framework

Some simply require descriptive enumeration, such as planning (P1 and P2). Some, on the other hand, require more complex activities. Refining those as sub-objectives (or even deducing justifications) is not a simple break-down that leads to a set of activities. Instead, it is necessary to detail sub-objectives and to explain both why the inference step is correct and why it is sufficient to conclude the objective from these justifications. Thus, more than a decomposition, we must associate justifications to a certification objective and also the reasons that allow the passage from justifications to conclusion.

There is a tremendous literature on how to build an assurance case and on how to make a graphical or textual representation. On the academic side, we can cite the *Goal Structuring Notation* (GSN) [McD94], [KW04], *Claim-Argument-Evidence* (CAE) [EC02], a textual approach from Rushby [RXRW15], and *Justification Diagram* (JD) [Pol16], [DPBF18]. On the standardization organism side, there is the *Structured Assurance Case Meta-model* [OMG13], and the *Generic Methodology for Verification and Validation* [RVS13].

Concerning the usage of argumentation on COTS, the authors of [FS19] provide some material to argue that a COTS possesses a set of high level properties (called overarching properties) ensuring its airworthiness. The presented methodoglogy is not designed to handle such a standard-agnostic reasoning since the certification requirements are already provided by the MCP-CRI. Nevertheless the argumentations and analyses presented in the remainder of this paper can be used as building blocks for the high level argumentation presented in [FS19].

All of them rely on the model defined by Stephen Toulmin [Tou03], which focuses on three concepts: *claim*, *strategy* and *evidence* (or sub-claim). The claim is the conclusion, the objective to demonstrate. Evidence are the facts on which the claim is based and rely on *given* elements provided by external knowledge or specific V&V activities. The strategy adds information about the reasoning and justifies the passage from evidence to claim. A strategy may come with a *backing* that is an explanation of why the strategy is indeed applicable.

In the rest of the paper, to represent our argumentations, we have chosen an agnostic notation approach based on a simple graphical notation (which is kind of an abstract syntax) compliant with all existing notations (which are kind of a concrete syntax). We will not present all patterns but will instead focus on the Resource Usage 3 (RU3) objective and the Error Handling (EH) objective. Let us first provide a reminder of these objectives.

Objective RU3 *The applicant has identified the interference channels that could permit interference to affect the software applications hosted on the multi-core processor cores, and has verified the applicant's chosen means of mitigation of the interference.*

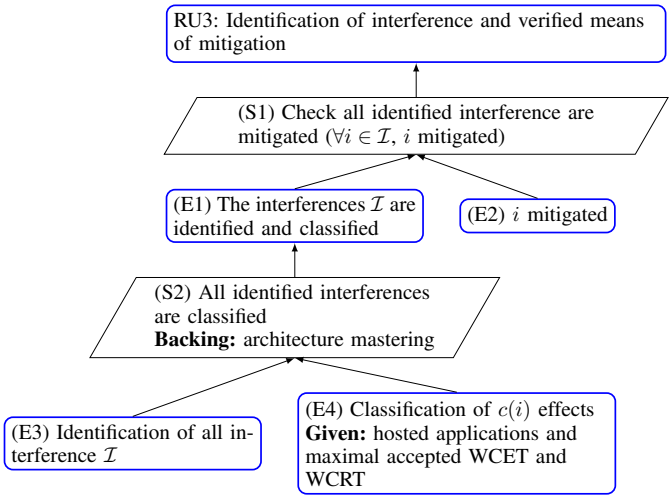

Fig. 2: RU3 pattern

Figure 2 shows its transcription as a pattern. Evidence (E1) states that the existing interferences have been identified and classified. This has been achieved because an expert, who masters the architecture, has reviewed and double-checked two activities (strategy (S2)). Evidence (E3) points to a report that summarizes which interferences have been identified, how they have been identified, and why the identification is sound and complete. Evidence (E4) points to a safety report that details the effects of each interference on the hosted applications. Those effects can be expressed in different units (e.g. delay, bandwidth). If expressed in delays, the questions are *what will be the impact of interference on the WCET (Worst Case Execution Time) and/or on the WCRT (Worst Case Response Time) of each application?* and *are these slow downs acceptable?* From this information, the applicant has defined adequate means of mitigation to prevent, for instance, unacceptable effects. Evidence (E2) collects all those means of mitigation, how they mitigate each unacceptable interference and how they were verified. The applicant can argue the compliance with RU3 because an expert, who masters the architecture, has reviewed and double-checked that each interference has been correctly mitigated (S1).

The applicant shall also identify the possible hardware failures, their impact, and the means of mitigation. This requirement is called *error handling – EH* in the MCP-CRI.

Objective EH *The applicant has identified the effects of*

failures that may occur within the MCP and has planned, designed, implemented and verified means (which may include a 'safety net' external to the MCP) commensurate with the safety objectives, by which to detect and handle those failures in a fail-safe manner that contains the effects of any failures within the equipment in which the MCP is installed.

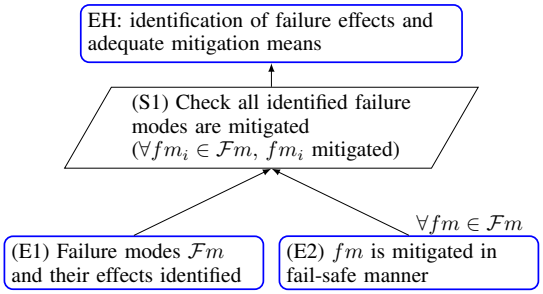

Fig. 3: Justification pattern for EH

As depicted by Figure 3, the argumentation structure for EH is quite similar to the one of RU3. The top level claim is EH with a shorten textual description. The applicant can argue the compliance with EH because a traceability matrix (for instance) shows the coverage between the identified failure modes and the associated mitigation means (S1). Evidence (E1) is a report that collects all the identified failure modes and their effects [Vil92]; whereas evidence (E2) describes the associated mitigation means. The latter should enforce that the effects of the failure modes are handled in a *fail-safe* manner (*i.e* contained within the equipment on which the multi-core processor is installed).

B. Interference and safety analyses

As illustrated before, a pattern comes with a series of *evidences* that support the reasoning. Among these evidences, two types of analyses are required by the MCP-CRI: *interference analysis* and *safety analysis*. However, these two kinds of analysis are made quite independently from each other and often rely on different abstractions and tools (e.g. WCET static analysis [AEF+14] for interference and MBSA - model based safety assessment - [BVÅ+03] for safety). An idea to overcome this divergence is to unify both analyses and to base them on a common abstract representation of multi-core processors. For that purpose, we have developed a modelling language, PML (PHYLOG Meta-Model) [BBB+18a], representing a multi-core-based system.

1) Interference analysis: Due to resource sharing, coupling exists at the platform level, which can cause interferences between the applications. Certification requires interference situations to be fully controlled and mastered in any configuration for safety critical applications, as explained for *resource usage 3 – RU3*. Moreover, the applicant shall also identify the shared resources and shall describe a usage domain for each of them (how the resources are shared and how to prevent resource capabilities from being exceeded). This requirement is called *resource usage 4 – RU4* in the MCP-CRI.

We have defined two approaches to compute *topological interference* in [BBB+18a], [BPS19] and *cache coherence interference* [SBP19]. With these solutions, the argumentation pattern can directly be filled. Notice that some complementary analyses are proposed in the literature. The first widespread approach is intensive benchmarking to observe interference and part of their effects [NP12], [RGG+12], [GJlR+15], [CCB17] As an example, [PSJ+19] studies the impact of Time Division Multipla Access (TDMA) and Acquisition-Execution-Restitution (AER) execution models on interference on the NXP T2080 platform. The second approach is more recent and proposed a profiling of applications [GlRS18] to quantifiy the transactions. With these characterization methods, the argumentation pattern for RU3 must be adapted to link benchmark and envelope with exact interference and effects.

2) Safety analysis: Due to the high integration density, the complexity of internal components and the mix of hardware / software inside multi-core chips, their behaviors under random failures are hard to define and to characterize. Thus, two specific objectives are defined for dependability and safety issues in the MCP-CRI / CAST-32A. First, the applicant shall argue that the critical configuration settings are static and are protected against inadvertent changes at run-time. This requirement is called *resource usage 2 – RU2* in the MCP-CRI. Second, the applicant shall also identify the possible hardware failures, their impact and the means of mitigation. This requirement is called *error handling – EH* in the MCP-CRI.

As identified by [Pro14], performing safety analyses with classical techniques like FMEAs, fault-trees or Markov chains raises two main challenges. First, since multi-core are highly hierarchical and complex systems, their modelling with classic formalisms will be cumbersome and error-prone. Second, classic formalisms require an in-depth knowledge of the multi-core architecture and internal components failure modes. Such knowledge is seldom available, since the chip makers will not commit themselves to provide detailed information about multi-core architectures.

To tackle these challenges, we have developed a pragmatic model-based safety assessment in [BBB+18b], [BBB+19], which can be performed with the first functional level of the multi-core (like the one depicted in Figure 4) and abstract failures modes based on the services provided by components.

III. USE CASE: ROSACE EXECUTING ON THE KEYSTONE

Before going into further details, we introduce the use case on which we apply the PHYLOG methodology: an application executing on a multi-core. The application is the ROSACE [PSG+14] longitudinal controller, which has been extended to include neural network representations of aerodynamic coefficients in the flight dynamic model. The application runs on bare metal on the KEYSTONE TCI6630K2L [Tex13] from Texas Instruments, which is depicted in Figure 4.

A. KEYSTONE

This platform is composed of:

1) an eight C66 DSP pack, in which each core comes with dedicated L1 and L2 caches, and a memory extension and protection unit (MPAX);
2) a four ARM pack, in which each core comes with dedicated L1 caches, and a memory management unit (MMU);

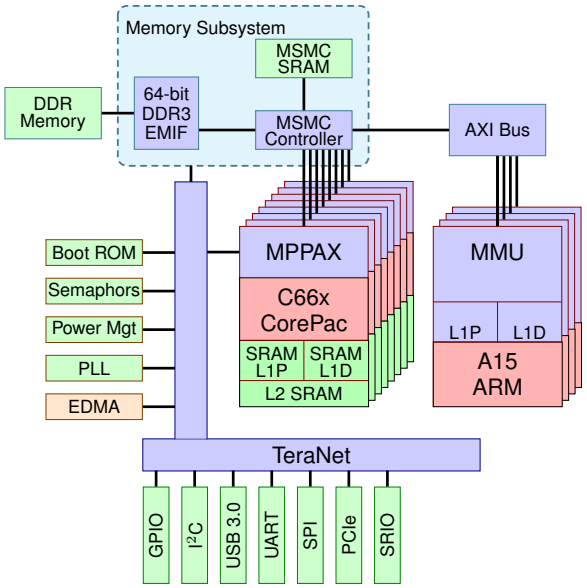

Fig. 4: Keystone – simplified view

3) a central memory system that gives access to the platform's SRAM (MSMC SRAM), and an external DDR. The memory access management is performed by the Multicore Shared Memory Controller (MSMC);
4) a set of IO peripherals (e.g. GPIO, UART), and utility peripherals (e.g. Boot, Semaphores);
5) a memory transfer peripheral (EDMA);
6) an ultra speed bus (TeraNet) connecting the peripherals, the memories, and the cores.

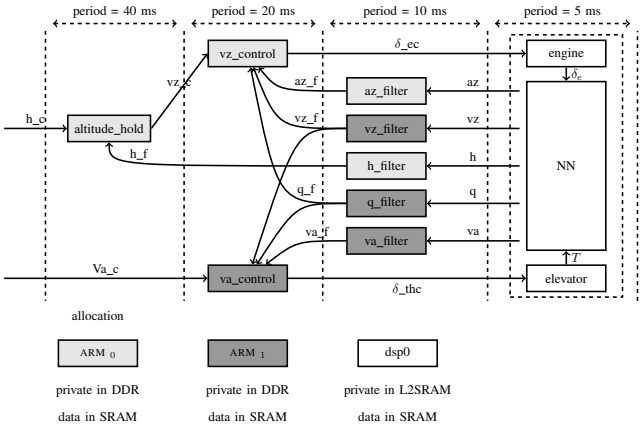

Fig. 5: ROSACE– overview

B. ROSACE

The ROSACE is a use case developed in MATLAB/SIMULINK [PSG+14]. It includes a model of the dynamic of an aircraft with actuators (engines and elevators) in closed-loop with sensors and a switching controller allowing tracking of altitude and airspeed requirements. The closed-loop simulation model involves four sample times for the system to be controlled, as shown in Figure 5. Actually, the aircraft model should be time-continuous, but for simulation and code generation purposes, it has been discretized according to a fast enough sample time.

Due to the complexity of flight control laws (including guidance, mode selection, protection, and control), and a high safety level, some flight parameters are critical to ensure the good behaviour of the aircraft. Thus, the availability of all or part of system states has become essential, and erroneous values can reduce performance or stability. A classic approach consists in estimating these parameters thanks to signal processing (Extended Kalman Filter [SHEP13]) or model-based techniques which propose to embed all or part of the aircraft flight dynamic model.

This model includes some mechanical parameters like mass, inertia, and aerodynamic coefficients which represent forces and moments applied on the system in motion. Such aerodynamic coefficients are often issued from CFD (computational fluid dynamics) computations, wind tunnel or flight testing. Usually, these data are only available in the form of look-up tables, which are not very convenient for on-board implementation, and that is why analytical and differentiable approximations are preferred, especially because of their lower memory requirements. Neural networks are good candidates for representing such complex nonlinear parameters [Har98].

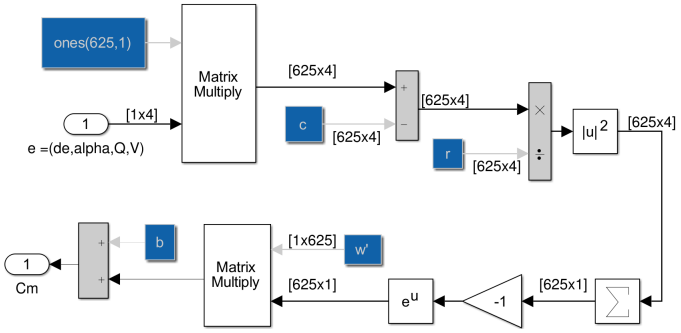

Fig. 6: SIMULINK implementation of the 625 cells RBFN

In this use case, we consider the pitching moment M_a appearing in the longitudinal flight nonlinear equations of motion [SEH15]:

$$M_a = SP_d l C_m \tag{1}$$

where S, P_d, l refer respectively to reference area, dynamic pressure, and mean aerodynamic chord. The aerodynamic coefficient C_m can be written as

$$C_m = C_{m0} + C_{m\delta}\delta_e + C_{m\alpha}\alpha + C_{mq} * f(q, V) \tag{2}$$

where δ_e, α, q, V are, respectively, elevator deflection, angle of attack, airspeed, and pitch rate. Coefficients C_{m0}, $C_{m\alpha}$, C_{mq} are non-linear effects mainly depending of M the Mach number.

We use a Radial Basis Function Network (RBFN) to approximate the MISO (multi input - single output) equation 2 as

$$C_m^{NN} = \sum_{k=1}^{n_{cel}} w_k exp\left(-\sum_{j=1}^{4}\left(\frac{e_j - c_{k,j}}{r_{k,j}}\right)^2\right) + b \tag{3}$$

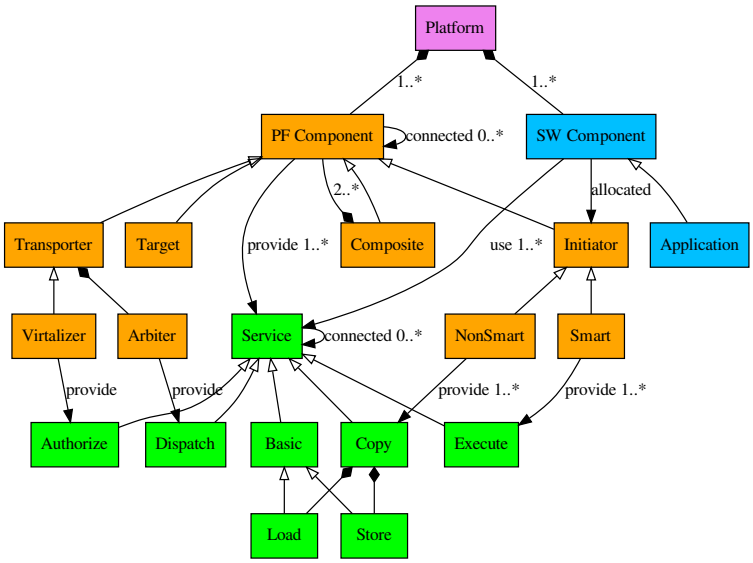

Fig. 7: PML model

where nel is the RBFN number of cells, $e_j, \rfloor = 1 \ldots 4$ are the inputs δ_e, α, q, V. In (3), RBFN parameters are c: center of a RBF cell, r: radii of a RBF cell, w: weighting factor and b: bias.

The learning of parameters, noted $\Theta = (r, b, w, c)$, is an optimization process aimed at minimizing the quadratic error between the value of the coefficient C_m at different points (obtained by simulations), and the corresponding neural output $C_m^{NN}(\Theta)$.

Finally, by considering 5 cells for each of the 4 inputs, we obtain an RBFN with 625 cells with optimized centers and radii. For validation and code generation purposes, the RBFN is implemented in MATLAB/Simulink as presented in Figure 6, and the efficiency of the RBFN is illustrated in Figure 8.

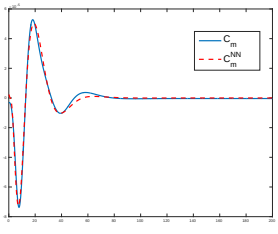

Fig. 8: Temporal simulation of a climb manoeuvre - blue: C_m - red: C_m^{NN}

IV. Application of Phylog methodology on the Keystone

In this section, we apply the Phylog methodology for the system defined previously: Rosace executing on the Keystone.

A. PML overview

Both EH and RU3 heavily rely on a mastering of the multi-core processor architecture. Moreover, a modelling of the hardware and software components of the platform must be achieved prior to any safety or interference analysis. We propose to perform this modelling using the Phylog Modelling Language (PML), whose model is depicted by the Figure 7. The model contains 3 types of components: hardware components in yellow, service components in green and software components in blue.

1) Hardware components: The classification of hardware components proposed in PML is directly inspired from the initiator-target model introduced in [BR14], [JMB16], [MJB16], [MJB$^+$17].

Definition 4.1 (Initiator-target model): A multi-core is composed of four types of components:

- **Initiator**, which is subdivided in two types. **Smart initiator** (e.g. core), *i.e* components which can initiate single transactions through the architecture to *target* components. **Non smart initiator** (e.g. DMA), *i.e* components which can only initiate dual transactions (*i.e* with two targets at the same time);
- **Target** (e.g. DDR or PCIe), *i.e* end-components targeted by initiators;

- **Transporter** (e.g. bus), *i.e* any intermediate component between initiators and targets. A Transporter can be a **Virtualizer** (e.g. MMU), *i.e* any intermediate component between initiators and targets providing a virtual resource segregation. A Transporter can contain an **Arbiter** (e.g. Serializer), *i.e* a component handling the concurrent accesses to a given resource.
- **Composite** (e.g. CorePac), *i.e* a composition of hardware components.

The components of Figure 4 are **colored** according to their type, the color code being: red for smart initiators, blue for transporters, green for targets, and orange for non-smart initiators.

2) Software components: The software components that will be executed by the platform are user applications. At this step, we consider only bare metal applications and partitioned non preemptive schedules computed off-line. Thus, each piece of software is allocated to a **unique** initiator. The allocation of ROSACE is defined in Figure 5: functions *altitude_hold*, *vz_control*, *az_control*, and *h_filter* are allocated on the ARM_0. Any software may request some data that are potentially distributed over several components. To retrieve these data, the initiator executing the software initiates communications within a multi-core called *transactions*.

Definition 4.2 (Transaction): A transaction starts from an initiator and follows pre-defined *path(s)* to its final target, a transaction may include the response(s) of the target.

3) Service components: The transactions are issued thanks to a platform service call. The idea of PML is to abstract the platform as a set of services. Compared to what we have presented in [BBB+18a], we have developed a second version based on the notion of service.

Definition 4.3 (Platform service): A platform offers a set of services that can be called upon by the initiators and that generate transactions. We have identified the following services:

- EXECUTE: execution of a piece of software on some core;
- LOAD: retrieval of some data from a given target t by an initiator i.
- STORE: writing of some data to a given target t by an initiator i.
- COPY: copy of some data from one memory area t_1 to another t_2 by a non-smart initiator i.
- AUTHORIZE: forbidding of transactions outside of an authorized address; the set of authorized addresses is defined at configuration by a table memorized by the service.
- DISPATCH: handling of simultaneous accesses to subsequent services.

B. PML representation of the KEYSTONE

1) General model: The *instantiation* of the PML model for the KEYSTONE provides

1) The physical components of the platform depicted by the Figure 4 categorised according to the Initiator-Transporter-Target model,
2) The software components, here ROSACE has been split into three pieces,
3) The physical links between physical components (used or not),
4) The services provided by each platform component. For the KEYSTONE, we consider that all components provides both the STORE and LOAD services, an Initiator additionally provides a unique EXECUTE service, a Transporter (resp. Virtualizer) provides a unique DISPATCH (resp. AUTHORIZE) service.

At this step, the platform is said to be "unconfigured", that is, the relations *allocated* and *connected* (for services) are unknown. Without any further information, any connection between services is possible (if a physical connection exists between the components providing these services) and any software allocation is possible.

Due to the lack of space, we do not provide the full instantiation of the unconfigured KEYSTONE platform[1]. These instantiations can be tedious to perform for commercial platforms (like the KEYSTONE). That is why a simple Domain Specific Language (DSL) has been defined and integrated in a translation tool.

2) Configuration of the platform: The platform can then be *configured* by populating the *allocation* and *connection* relations. In our case, the allocation, mapping, and schedule over the platform are fully static:

- the ROSACE application is split in three pieces, namely ROSACE_0 allocated to ARM_0, ROSACE_1 allocated to ARM_1, and the Neural Network (NN) allocated to the DSP_0;
- ARM_0 and ARM_1 store their code and data in the DDR memory. DSP_0 stores its code and data in its local SRAM.
- All global variables exchanged between the ARMs and DSP are stored in the global SRAM controlled by the MSMC.

These information make it possible to connect the services of the platform used to perform the EXECUTE, LOAD, and STORE requests of the applications. An excerpt of the KEYSTONE service connection[1] is provided in Figure 9. In this excerpt, we do not consider the private memories and caches, nor the TeraNet connecting the MSMC to the cores' memory management units (MPAX for DSPs, MMU for ARM). Nevertheless, this simple excerpt contains the main types of components, allowing it to illustrate the different kinds of transactions transiting through the platform.

The software components uses the EXECUTE, LOAD, and STORE services provided by the cores on which it is allocated. The initiator (core) does not request any additional service to provide the execution service. Concerning LOAD and STORE services, the ROSACE application will request some accesses to the DDR and the MSMC SRAM. Consequently, the LOAD and STORE services of the initiators are connected to the LOAD and

[1]the interested reader can find the diagrams at https://www.onera.fr/sites/default/files/274/ERTS_material.zip

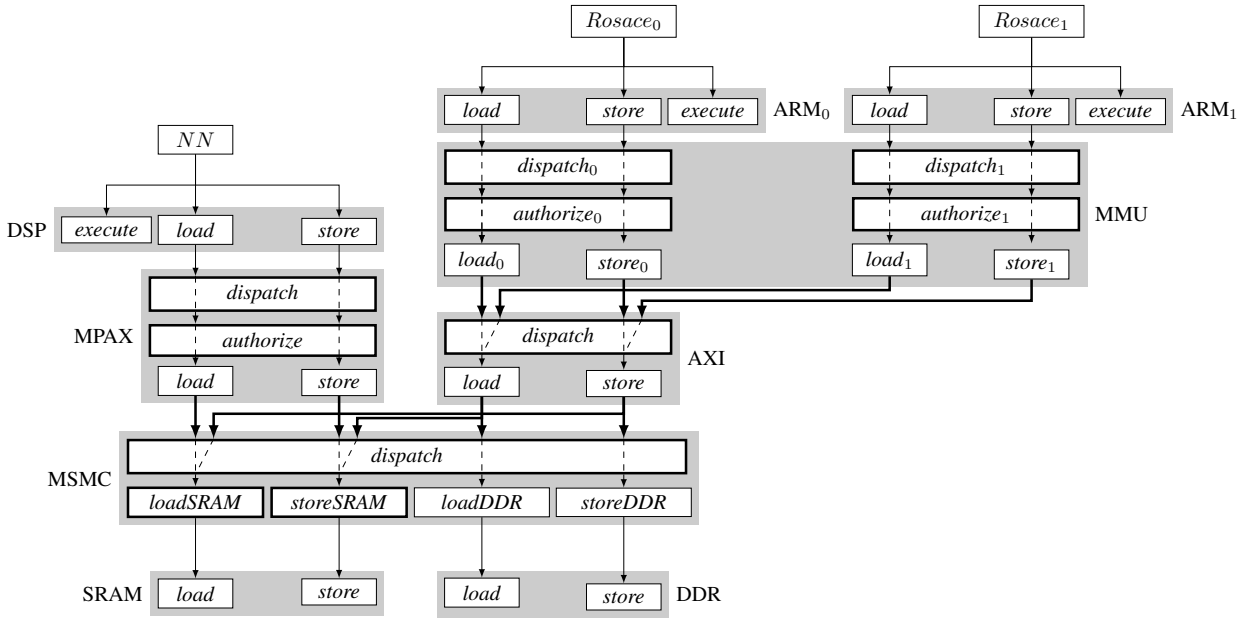

Fig. 9: Excerpt of the KEYSTONE service-oriented architecture

STORE services of their memory management units (MPAX for DSPs and MMU for ARM) through their DISPATCH service.

Since MPAX and MMU are virtualizers, they provide an AUTHORIZE service enforcing the memory allocation considered in the configuration. For instance, the DSP is only allowed to access the MSMC SRAM, so any transaction involving a communication with the DDR will be rejected by the AUTHORIZE service of the MPAX.

Eventually, the transporter components, like the MSMC, that may be involved in several simultaneous transactions provide a DISPATCH service handling this concurrency. For instance, in Figure 9, the DISPATCH service of the MSMC is able to handle concurrent loads and stores if the target of these requests are different. The potential conflicts are depicted in Figure 9 by the dotted lines using the same service.

C. Interference analysis on the KEYSTONE

Back to RU3 pattern of Figure 2, strategy (S2) needs to identify all interferences (E3) and to classify them (E4).

1) Interference identification (E3): Existing interference analyses consists in enumerating all combination of transactions (more or less deeply). An interference analysis applied to a service-based modelling identifies the simultaneous service calls that may cause a degradation of QoS. A transaction is a sequence of service calls in PML and looking at Figure 9, we can observe the LOAD from ROSACE_0 on ARM_0 to the LOAD of the Target DDR (sequence of arrows and sevices). Thus, if there exists a service s reached by two arrows from two services s_1 and s_2, it means that s can be used by two transactions from s_1 and s_2. If the transactions arrive at the same time on s, they will provoke a contention on s, and one of them will have to wait until s becomes free again. From that point of view, an "interference channel" is a set of services targeted by simultaneous transactions produced by different initiators.

In the system under study, shown in Figure 9, 12 services can be simultaneously reached from different initiators:

- the three services of the AXI component (reached for instance when the two ROSACE_i initiators run two LOAD or STORE transactions);
- the five services of the MSMC component (reached for instance when NN and one of the ROSACE_i initiators try to access at the same time the SRAM or the DDR memory);
- and the four services of the SRAM and DDR components.

Thus, these services are part of the interference channels that can occur in the system.

To answer objective RU3, one needs to identify all the interference channels and all the interferences generated by simultaneous transactions involving these channels. For that purpose the KEYSTONE case-study PML model is transformed into an interference oriented view. This view is modeled in IDP[2], a knowledge-based system allowing formal modeling and analysis in a first-order logic language. Based on this formal modeling, an exhaustive analysis identifies:

- 32 binary interference transactions (i.e., involving 2 initiators), and 32 ternary interference transactions (i.e., involving the 3 initiators);
- 10 interference channels, i.e., 10 sets of shared services that can be simultaneously reached by at least one of these 64 interference transactions. These interference channels are:
  1) {*msmc-dispatch*}, involved in 12 binary transaction and 30 ternary transactions,
  2) {*msmc-dispatch, msmc-store-sram, srm-store*}, involved in 3 transactions

[2]https://dtai.cs.kuleuven.be/software/idp

3) {*axi-dispatch, msmc-dispatch*}, involved in 8 transactions
4) {*axi-dispatch, axi-store, msmc-dispatch*}, involved in 2 transactions
5) {*axi-dispatch, axi-load, msmc-dispatch*}, involved in 2 transactions
6) {*axi-dispatch*, *axi-load*, *msmc-dispatch*, *msmc-load-sram*, *sram-load*}, involved in 1 transaction
7) {*msmc-dispatch*, *msmc-load-sram*, *sram-load*}, involved in 3 transactions
8) {*axi-dispatch*, *axi-load*, *msmc-dispatch*, *msmc-load-ddr*, *ddr-load*}, involved in 1 transaction
9) {*axi-dispatch*, *axi-store*, *msmc-dispatch*, *msmc-store-ddr, ddr-store*}, involved in 1 transaction
10) {*axi-dispatch, axi-store*, *msmc-dispatch*, *msmc-store-sram*, *sram-store*}, involved in 1 transaction.

Classifying the extra-cost due to these interference channels would require testing each of these 64 interference transactions. However, it is possible to take advantage of the symmetries of the architecture to reduce the number of transactions to run. For instance, if we notice (and can argue) that the two ARM blocks are symmetric and that the two ROSACE$_i$ initiators are able to produce similar LOAD and STORE requests, then, for each of the 10 previous interference channels, it is possible to regroup the transactions by symmetric classes, i.e., groups of transactions that produce the same effect when crossing the shared ressources. For instance, let us again consider the first interference channel {*msmc-dispatch*}. Among the 12 binary transactions crossing it, we have:

- t_0 = ROSACE$_0$ $\rightsquigarrow$ *ddr-store* $\|$ *NN* $\rightsquigarrow$ *sram-load*
- t_1 = ROSACE$_1$ $\rightsquigarrow$ *ddr-store* $\|$ *NN* $\rightsquigarrow$ *sram-load*

Thanks to the symmetry between ARM$_0$ and ARM$_1$ and between ROSACE$_0$ and ROSACE$_1$, t_0 and t_1 will produce the same worst-case interference on *msmc-dispatch*. To classify the interference on *msmc-dispatch*, it is sufficient to only focus on t_0 instead of the two of them. Using the IDP formal modeling, we can show that it is sufficient to consider only 6 transactions over the 12 double transactions involved in the {*msmc-dispatch*} interference channel to classify it.

2) Interference classification: Once the interferences have been identified, the second step is to classify them. For that purpose, one benchmark is associated with each interference transaction. For the sake of conciseness, let us consider the interference transaction:

t = ROSACE$_0$ $\rightsquigarrow$ *sram-store* $\|$ ROSACE$_1$ $\rightsquigarrow$ *sram-store*

The interference channel involved in t is {*axi-dispatch*, *axi-store*, *msmc-dispatch*, *msms-sram-store*, *sram-store*, }. To classify the cost of t, we define a benchmark composed of two processes: p_0 (resp. p_1) hosted by ARM$_0$ (resp. ARM$_1$) simulating STORE requests from ROSACE$_0$ (resp. ROSACE$_1$). p_0 and p_1 execute the same infinite loop:

```
while(1) {
   time1 = current cycle number
   assembly(str r9 [r8])
   time2 = current cycle number
   d = time2 - time1
}
```

d = *time2* − *time1* denotes the duration to store the content of register $r9$ at the address contained in register $r8$. Only the d parameter produced by p_0 is measured. Figure 10 shows the variation of d. In the first part of the trace, p_0 runs alone. d approximatively varies around 158 clock cycles). At time 460, p_1 runs on ARM$_1$, causing an interference. Then d increases to 180 cycles. At time 900, p_1 stops. The value of d returns to 158. Following this benchmark, running two STORE transactions in parallel on ARM$_0$ and ARM$_1$ can lead to an extra cost of 14%.

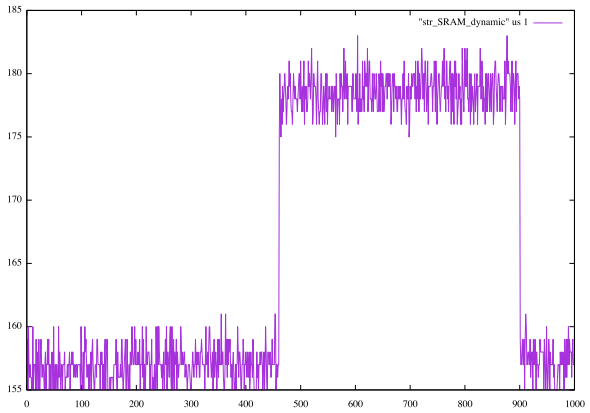

Fig. 10: Interference ROSACE$_0$ $\rightsquigarrow$ *sram-store* $\|$ ROSACE$_1$ $\rightsquigarrow$ *sram-store*

D. Safety assessment of the KEYSTONE

The argumentation diagram of the EH objective (see Figure 3) indicates that a safety assessment must be performed to identify the possible failure modes of the platform and their safety impact for the considered applications. Such an assessment can be performed using Model-Based Safety Assessment (MBSA) using the service oriented representation of the platform provided in the Figure 9.

1) Modelling of the platform: The modelling of the platform is based on a library of reusable components formalized with the *mode automata* [Rau02]. Such a formalization enables us to:

1) be generic and formal,
2) implement the following definitions as a library of components coded with ALTARICA,
3) use pre-existing tools to perform automatic safety assessments.

Up to now, there have been very few FMEA identifying the failure modes for a given platform. This can be explained by the in-depth architecture knowledge required to perform an FMEA, and by the reluctance of chip makers to (legally) commit themselves to provide a detailed FMEA. Therefore, the failure modes are derived from pragmatic reasoning. In the remainder of the paper we consider:

1) *err*: the component does not provide a proper service,
2) *lost*: the component does not provide anything.

These abstract failure modes, and the link with concrete failure modes, are summarized in Table I.

Component family	Abstract FM	Comments	Example of concrete FM
Core	*err*	mis-execution and corruption of LOAD/STORE transactions	Register corruption by SEU
	lost	no software execution	OPCODE corruption
Memory	*err*	LOAD/STORE a corrupted data	Memory area corruption
	lost	no LOAD/STORE service	DDR controller stalled
Interconnect	*err*	corruption of the transactions	Internal queue corruption
	lost	transactions are not dispatched	Queue overflow

TABLE I: Atomic component failure modes

The applications are affected by the LOAD/STORE transactions failures, which result from the considered atomic component failures. For instance, in Figure 9, if ARM_0 asks to LOAD a data from DDR, and if the MMU is *lost*, the LOAD service to DDR is no more provided to ARM_0. Similarly, if the DSP asks to STORE a value in the SRAM, and if the MSMC is *err*, then an erroneous value is stored in the SRAM, corrupting its content.

The impact of the atomic components' (initiator, target and transporter) failure mode on the incoming transactions are modelled using the mode automaton formalism. Once these local safety effects are modelled, the analyst can instantiate and connect these component to provide the model of the complete platform.

The platform model[3] built for the KEYSTONE is depicted by Figure 11. Since the components of the library model the behavior of initiators, transporters, and targets, the obtained safety model can be easily deduced from the service-oriented architecture depicted in Figure 9.

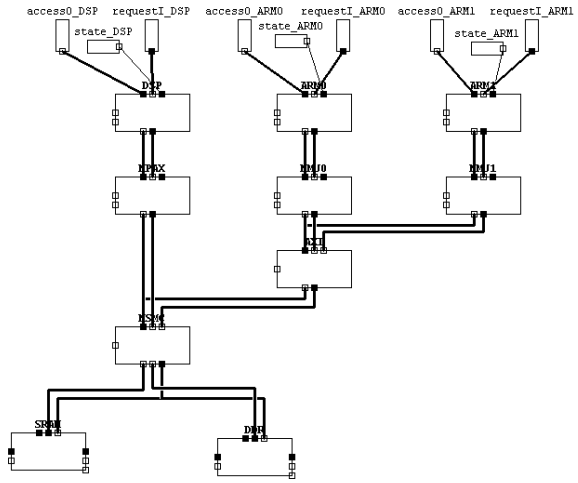

Fig. 11: Safety model of the KEYSTONE platform

2) Safety objective assessment: Let us consider that the loss of any application (ROSACE_0, ROSACE_1 or NN) is a feared event, called FC. Thanks to CECILIA-OCAS, we are able to automatically compute the smallest combinations of physical failures (so-called minimal cutsets) leading to FC. The analyzer provides a set of single failures triggering FC, the most obvious ones being the direct loss of the resource upon which the application directly rely on, that is: 1) the loss or erroneous behavior of the executing unit, *i.e*DSP for the NN and the ARM for ROSACE, 2) the loss or corruption of the target, *i.e* DDR for ROSACE and the SRAM for both ROSACE and the NN, 3) the loss of a transporter on the transaction path to the final target, *i.e* the MPAX and the MSMC for the NN and the MMU, AXI Bus and the MSMC for ROSACE.

In addition to these obvious failures, the assessment is able to detect some failure propagation through successive data exchanges between the NN and ROSACE_i. For instance, an erroneous DDR leads to an erroneous providing to the LOAD service for ROSACE_i applications. Because of this erroneous LOAD, the applications then perform an erroneous STORE in the SRAM, eventually resulting in the erroneous behaviour of the NN itself. The identification of such failure propagation is paramount to the safety assessment of platforms implementing robust partitioning to enforce safety.

V. CONCLUSION

We have presented the PHYLOG framework and applied it to a real use case. We believe the approach to be valuable and adapted to certification. Our patterns and analyses have been presented to EASA, who are open to the idea of applicants relying on these kinds of techniques.

PHYLOG will end in December 2020. By then, we will put our argumentation patterns online. We will describe the PML formalism more in details. We will refine the interference analysis to take into account symmetries. We will develop automated transformations from PML to IDP and ALTARICA.

REFERENCES

[AAAC17] Irune Agirre, Jaume Abella, Mikel Azkarate, and Francisco Cazorla. On the Tailoring of CAST-32A Certification Guidance to Real COTS Multicore Architectures. In *12th IEEE International Symposium on Industrial Embedded Systems (SIES'17)*, 2017.

[AEF+14] Philip Axer, Rolf Ernst, Heiko Falk, Alain Girault, Daniel Grund, Nan Guan, Bengt Jonsson, Peter Marwedel, Jan Reineke, Christine Rochange, Maurice Sebastian, Reinhard von Hanxleden, Reinhard Wilhelm, and Wang Yi. Building timing predictable embedded systems. *ACM Trans. Embedded Comput. Syst.*, 13(4):82:1–82:37, 2014.

[AMP19] Jyotika Athavale, Riccardo Mariani, and Michael Paulitsch. Flight safety certification implications for complex multi-core processor based avionics systems. In *25th IEEE International Symposium on On-Line Testing and Robust System Design, IOLTS 2019, Rhodes, Greece, July 1-3, 2019*, pages 38–39, 2019.

[BBB+18a] Pierre Bieber, Frédéric Boniol, Youcef Bouchebaba, Julien Brunel, Claire Pagetti, Olivier Poitou, Thomas Polacsek, Luca Santinelli, and Nathanaël Sensfelder. A model-based certification approach for multi/many-core embedded systems. In *9th European Congress on Embedded Real Time Software and Systems (ERTS 2018)*, 2018.

[BBB+18b] Frédéric Boniol, Youcef Bouchebaba, Julien Brunel, Kevin Delmas, Claire Pagetti, Thomas Polacsek, and Nathanaël Sensfelder. PHYLOG: a model-based certification framework. In *37th AIAA/IEEE Digital Avionics Systems Conference (DASC 2018)*, 2018.

[3]the complete ALTARICA model is available at https://www.onera.fr/sites/default/files/274/ERTS_material.zip

[BBB+19] Frédéric Boniol, Youcef Bouchebaba, Julien Brunel, Kevin Delmas, Claire Pagetti, Thomas Polacsek, and Nathanaël Sensfelder. A service-based modelling approach to ease the certification of multi-core COTS processors. In *SAE 2019 AeroTech Europe*, 2019. under submission.

[BPS19] Frédéric Boniol, Claire Pagetti, and Nathanaël Sensfelder. Identification of multi-core interference. In *Proceedings of the 19th IEEE High Assurance Systems Engineering Symposium (HASE'19)*, 2019.

[BR14] Vincent Brindejonc and Anthony Roger. Avoidance of dysfunctional behaviour of complex cots used in an aeronautical context. In *19eme Congrès de Maîtrise des Risques et Sûreté de Fonctionnement*, 2014.

[BVÅ+03] Marco Bozzano, Adolfo Villafiorita, Ove Åkerlund, Pierre Bieber, Christian Bougnol, Eckard Böde, Matthias Bretschneider, Antonella Cavallo, C Castel, M Cifaldi, et al. Esacs: an integrated methodology for design and safety analysis of complex systems. In *Proc. ESREL*, pages 237–245, 2003.

[CCB17] Roberto Cavicchioli, Nicola Capodieci, and Marko Bertogna. Memory interference characterization between CPU cores and integrated gpus in mixed-criticality platforms. In *22nd IEEE International Conference on Emerging Technologies and Factory Automation, ETFA*, pages 1–10, 2017.

[Cer16] Certification Authorities Software Team. Multi-core Processors - Position Paper. Technical Report CAST 32-A, November 2016.

[DPBF18] Clément Duffau, Thomas Polacsek, and Mireille Blay-Fornarino. Support of justification elicitation: Two industrial reports. In *Advanced Information Systems Engineering - 30th International Conference, CAiSE 2018, Tallinn, Estonia, 2018, Proceedings*, Lecture Notes in Computer Science. Springer, 2018.

[EAS16] EASA (European Aviation Safety Agency). The Use of Multi-Core Processors in Safety-Critical Applications - CRI, 2016.

[EC02] Luke Emmet and George Cleland. Graphical notations, narratives and persuasion: a pliant systems approach to hypertext tool design. In James Blustein, Robert B. Allen, Kenneth M. Anderson, and Stuart Moulthrop, editors, *HYPERTEXT 2002, Proceedings of the 13th ACM Conference on Hypertext and Hypermedia, June 11-15, 2002, University of Maryland, College Park, MD, USA*, pages 55–64. ACM, 2002.

[FS19] Hakan Forsberg and Andreas Schwierz. Emerging cots-based computing platforms in avionics need a new assurance concept. In *the 38th Digital Avionics Systems Conference (DASC'19)*. IEEE Press, 2019.

[GJlR+15] Sylvain Girbal, Xavier Jean, Jimmy le Rhun, Daniel Gracia Pérez, and Marc Gatti. Deterministic Platform Software for Hard Real-Time systems using multi-core COTS. In *34th Digital Avionics Systems Conference (DASC'15)*, 2015.

[GlRS18] Sylvain Girbal, Jimmy le Rhun, and Hadi Saoud. METrICS: a measurement environment for multi-core time critical systems. In *9th European Congress on Embedded Real Time Software and Systems (ERTS'18)*, 2018.

[Har98] Georges Hardier. Recurrent rbf networks for suspension system modeling and wear diagnosis of a damper. In *1998 IEEE International Joint Conference on Neural Networks Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98CH36227)*, volume 3, pages 2441–2446, 1998.

[JMB16] Xavier Jean, Laurence Mutuel, and Vincent Brindejonc. Assurance methods for cots multi-cores in avionics. In *35th Digital Avionics Systems Conference (DASC'16)*, 2016.

[KW04] Tim Kelly and Rob Weaver. The goal structuring notation /- a safety argument notation. In *Proceedings of Dependable Systems and Networks 2004 Workshop on Assurance Cases*, 2004.

[McD94] John A McDermid. Support for safety cases and safety arguments using sam. *Reliability Engineering & System Safety*, 43(2):111–127, 1994.

[MJB16] Laurence Mutuel, Xavier Jean, and Vincent Brindejonc. Investigation of error types associated with failures in multicore processors. In *20eme Congrès de Maîtrise des Risques et Sûreté de Fonctionnement*, 2016.

[MJB+17] Laurence Mutuel, Xavier Jean, Vincent Brindejonc, Anthony Roger, Thomas Megel, and E. Alepins. Assurance of Multicore Processors in Airborne Systems, 2017.

[NP12] Jan Nowotsch and Michael Paulitsch. Leveraging multi-core computing architectures in avionics. In *Proceedings of the 2012 Ninth European Dependable Computing Conference*, EDCC '12, pages 132–143, Washington, DC, USA, 2012. IEEE Computer Society.

[OMG13] OMG. Structured assurance case meta-model (sacm). Technical report, Object Management Group, 2013.

[Pol16] Thomas Polacsek. Validation, accreditation or certification: a new kind of diagram to provide confidence. In *IEEE Tenth International Conference on Research Challenges in Information Science (RCIS'16)*, 2016.

[Pro14] Tatiana Prosvirnova. *AltaRica 3.0: a Model-Based approach for Safety Analyses*. PhD thesis, Ecole Polytechnique, 2014.

[PSG+14] Claire Pagetti, David Saussié, Romain Gratia, Eric Noulard, and Pierre Siron. The rosace case study: From simulink specification to multi/many-core execution. In *20th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'14)*, April 2014.

[PSJ+19] Sihyeong Park, Daeyoung Song, Hyeoksoo Jang, Mi-Young Kwon, Sang-Hun Lee, Hoon-Hyu Kim, and Hyungshin Kim. Interference analysis of multicore shared resources with a commercial avionics RTOS. In *the 38th Digital Avionics Systems Conference (DASC'19)*. IEEE Press, 2019.

[Rau02] Antoine Rauzy. Mode automata and their compilation into fault trees. *Rel. Eng. & Sys. Safety*, 78(1):1–12, 2002.

[RGG+12] Petar Radojkovic, Sylvain Girbal, Arnaud Grasset, Eduardo Quiñones, Sami Yehia, and Francisco Cazorla. On the evaluation of the impact of shared resources in multithreaded cots processors in time-critical environments. 8:34, 01 2012.

[RKR15] David J Rinehart, John C Knight, and Jonathan Rowanhill. Current practices in constructing and evaluating assurance cases with applications to aviation. Technical report, NASA, 2015.

[RTC11] RTCA, Inc. DO-178 ED-12C - Software Considerations in Airborne Systems and Equipment Certification, 2011.

[RVS13] Manfred Roza, Jeroen Voogd, and Derek Sebalj. The generic methodology for verification and validation to support acceptance of models, simulations and data. *The Journal of Defense Modeling and Simulation*, 10(4):347–365, 2013.

[RXRW15] John Rushby, Xidong Xu, Murali Rangarajan, and Thomas L Weaver. Understanding and evaluating assurance cases. Technical Report NASA/CR-2015-218802, NASA Langley Research Center, 2015.

[SAE10] SAE. Aerospace Recommended Practices ARP4754a - development of civil aircraft and systems, 2010. SAE.

[SBP19] Nathanaël Sensfelder, Julien Brunel, and Claire Pagetti. Modeling Cache Coherence to Expose Interference. In *Proceedings of the 31st Conference on Real-Time Systems (ECRTS'19)*, 2019.

[SEH15] Cédric Seren, Pierre Ezerzere, and Georges Hardier. Model–based techniques for virtual sensing of longitudinal flight parameters. *International Journal of Applied Mathematics and Computer Science*, 25, 03 2015.

[SHEP13] Cédric Seren, Georges Hardier, Pierre Ezerzere, and Guilhem Puyou. Adaptive extended kalman filtering for virtual sensing of longitudinal flight parameters. pages 25–30, 10 2013.

[Tex13] Texas Instruments. TCI6630K2L Multicore DSP+ARM KeyStone II System-on-Chip. Technical Report SPRS893E, Texas Instruments Incorporated, 2013.

[Tou03] Stephen E. Toulmin. *The Uses of Argument*. Cambridge University Press, Cambridge, UK, 2003. Updated Edition, first published in 1958.

[Vil92] Alain Villemeur. *Reliability, availability, maintainability and safety assessment*. John Wiley & Sons, 1992.