PHYLOG: a model-based certification framework

Frédéric Boniol, Youcef Bouchebaba, Julien Brunel, Kevin Delmas, Claire Pagetti, Thomas Polacsek, Nathanaël Sensfelder

PHYLOG: a model-based certification framework

Frédéric Boniol, Youcef Bouchebaba, Julien Brunel, Kevin Delmas, Claire Pagetti,
Thomas Polacsek and Nathanaël Sensfelder
1 ONERA-Toulouse, France

***Abstract*—This paper describes PHYLOG, a framework intended to help certify the use of a multi-core in an aeronautical context. Specific guidelines for such systems have been published in a document, the MCP-CRI / CAST-32A, which provides a series of objectives to be fulfilled. To justify that an objective is indeed achieved, PHYLOG relies on structured graphical notations, recursively refining each objective by solving it using a strategy, itself having its own sub-objectives, until all that remains are evidences found either through the use of formal methods, as part of the design choices, or in external documentation. The PHYLOG framework includes such formal methods, providing the means to model the multi-core and to acquire further evidences through automatic analysis.**

I. INTRODUCTION

A. Context

An aircraft is allowed to enter in operation if the manufacturer has obtained a type certificate from the certification authorities. For that, the aircraft manufacturer must demonstrate the compliance of its product with the regulatory requirements [EAS17]. An accepted mean of compliance with the requirements is to rely on mature standards, such as the ARP 4754 [SAE10], for the system's development process. When using this way to prove that a product is trustworthy, the certification activities consist in providing a detailed documentation, and justifications, that argue how the development process is indeed compliant with the standard.

The last decade has seen the emergence of multi-core processors and many-core architectures, i.e. chips integrating several cores interconnected by either a shared bus or a network on chip. Although these architectures may provide huge gains in terms of performance, they also face important challenges to their integration in safety critical environment. Unfortunately, existing standards, such as the ARP 4754 [SAE10] (recommendation on the development process), the DO 254 [RTC05] (recommendation on the hardware) and the DO 178 B/C [RTC08], [RTC11a] (recommendation on software development), do not fully cover the case of multi-core. This is the reason why aeronautic certification authorities, associated with industrial manufacturers, have published the Multi-Core Certification Review Item (MCP-CRI) [EAS16] (also published as the CAST32-A position paper [Cer16]), in order to provide a set of guidance for software planning and verification on multi-core chips. As a consequence, certifying those new architectures is a novel and challenging task.

B. Objectives of the project

PHYLOG[1] is a four years DGAC (French Civil Aviation) project (2016-2020) that aims at offering a model-based and software-aided certification framework for aeronautics systems based on multi/many-core architectures. More precisely, our goal is twofold: to reduce as much as possible the amount of textual documentation, by replacing it with model(s) wherever possible; to promote automatic analysis and replace part of the testing with formal methods, as recommended by the DO333 [RTC11b].

Such an approach offers several advantages: simplify the certification activities; cope with inflation of documentation; reuse design models; improve the coverage of requirements. Thus, our methodology aims at helping both the applicant answer the requirements and the certification authority to assess the arguments provided by the applicant. In previous works [BBD+16], [Pol16a], we proposed the basics of a new framework to simplify the certification of IMA systems. In PHYLOG, we reuse and improve those ideas to specifically address multi/many-core issues.

C. Methodology and focus of the paper

With our purpose being to cover the MCP-CRI / CAST-32A requirements, our first task has been to carefully analyze it. We briefly review those requirements in section II, and proceed to focus on two of them (namely RU2 and RU3) as a way to support the description of our methodology.

We believe that, in some situations, model-based approaches are more convenient than textual documentation. Because of this, we have identified which elements can be modeled and automatically analyzed.

The overall PHYLOG framework is detailed in section III. Our first step is to organize the justification around structured graphical notations diagrams, which we have used to refine the requirements. This argumentation methodology is presented in section IV, which features the application of a justification pattern on RU2.

As part of that framework, we proposed PML (PHYLOG meta-model) in [BBB+18]. It lets its users create a model describing the platform architecture (including its hardware and executive layers), which is intended to be used both as a mean to perform automatic analysis, and as way to obtain evidences to be used in the argumentation.

We then apply part of the methodology on RU2 in sections V and VI on two different use cases. We take the role of an applicant and show how to use the framework for these requirements.

[1] http://w3.onera.fr/phylog/

II. Certification Objectives

Multi-core platforms offer huge parallelisation capabilities and resource sharing that were unavailable in the mono-core platforms. This technological breakthrough motivates the need to adapt and create new certification objectives as formalized in the CAST32-A. In this section, we provide an overview of two particular objectives.

A. Position paper overview

The MCP-CRI / CAST-32A is not applicable in every situation: for instance, only DAL A, B or C systems are concerned, hyper-threading is not covered, lock-step systems do not need these additional requirements, etc.

Example 1. *Let us consider the simple, yet representative, multi-core COTS architecture depicted in the figure 1. If embedded in an aircraft, this architecture would have to be compliant with the* MCP-CRI / CAST-32A.

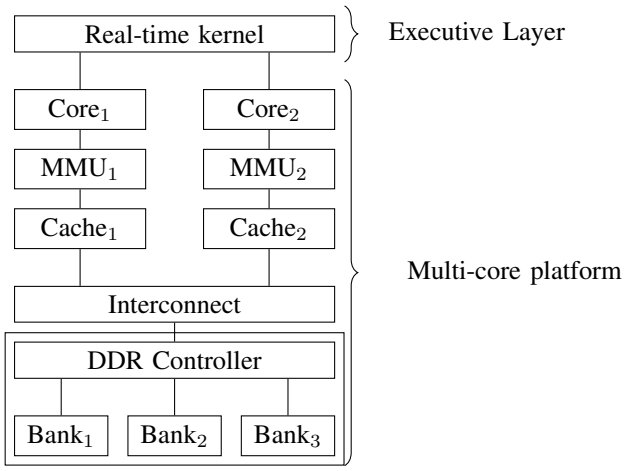

Fig. 1. Running example

The processor is composed of two cores, each of which has its own private L1 and L2 caches, as well as a Memory Management Unit (MMU) that offers memory protection. The shared memory, a DDR composed of three banks, can be accessed via a shared interconnect and a memory controller. Not represented here are the peripherals also accessible via the interconnect. The executive layer is composed of a real-time kernel that provides partitioning and follows the Arinc 653 [Aer97] paradigm. This means that any application will be statically mapped on a given core and wrapped in a partition the schedule of which is computed off-line.

The certification objectives of the MCP-CRI / CAST-32A [Cer16] are classified into four categories:

Planning: the 2 associated objectives consist in describing the architecture family (AMP,...), its usage and the plans to meet the objectives;

Resource Usage: the 3 associated objectives concern the description of the configuration settings, the identification of all interference channels and their mitigation, the description of the shared resources and how they will be used;

Software: these 2 objectives concern the correct execution of the software even in the context of multi-core;

Error Handling: the last objective concerns the possible hardware failures, the impact of interference generated by these failures and the means of mitigation.

We will focus in the sequel on two particular objectives, namely RU2 and RU3.

B. Resource Usage 2 (RU2): Alteration handling on configuration settings

The second resource usage objective is focused on the alteration of configuration settings. The configuration settings are registers and pin settings which enable, disable or restrict the usage of some multi-core processor resources such as: activation or deactivation of resources (cores, peripherals like DMA); modification of core frequency; management of dynamic features such as energy saving management; etc. Thus, the alteration of such configuration settings may change the behavior of the multi-core processor in a way that prevents the applicant from ensuring that functional, timing and safety requirements are still fulfilled.

Objective 2. *The applicant has planned, developed, documented, and verified a means that ensures that, in the event of any of the Critical Configuration Settings of the multi-core processor being inadvertently altered, an appropriate means of mitigation is specified.*

Example 2. *Let us again consider the example 1 and let us assume that the core frequency can be modified at run-time by changing the value of a register. In such cases, the applicant must consider the hazard* core frequency modified by a corruption*. They must first assess whether the configuration setting is critical and, if the answer is positive, they must add some means of mitigation on the platform.*

- *Identification of the effect: If the frequency is changed, then the WCET of any application mapped on the core will be modified. If it increases, then there may be a deadline miss and, if the application is DAL A, the consequences may be not acceptable. Therefore, the setting can be categorized as critical;*
- *Mitigation solution: one possible mitigation is to store a copy of the core frequency in a trusted resource, i.e., a memory with a low failure rate, and regularly compare the values of the register with the trusted copy. When the comparison fails, the mitigation can modify the register to restore the nominal situation;*
- *Validation of the means of mitigation: for the chosen solution, the applicant must argue that it is appropriate, and that the platform will fulfill the high level safety requirements.*

In section V, we show how to construct an argumentation for that alteration and the proposed mitigation solution.

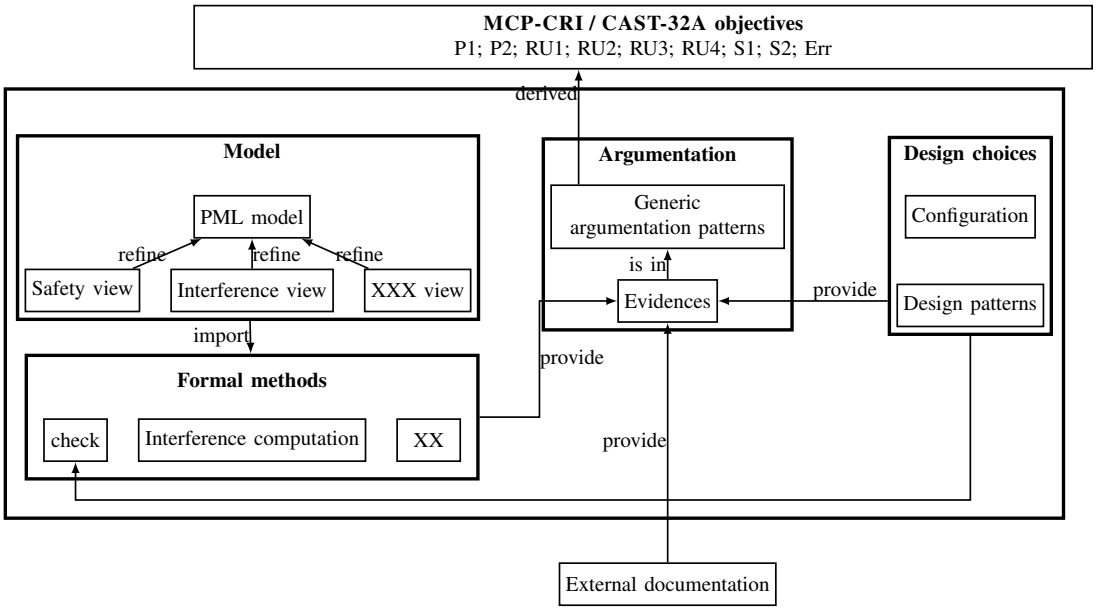

Fig. 2. Overview of the PHYLOG framework

C. Resource Usage 3 (RU3): Interference analysis and mitigation

On a multi-core processor, software components can interfere, because of concurrent shared resource accesses. The CAST32-A identifies several types of interference:

Definition 1 (Interference)**.** *An interference is any perturbation of the execution of functionally independent applications hosted on the same multi-core processor such as:*

- *Time interference : causing a delay in software execution;*
- *Functional interference : causing data corruption between independent applications;*
- *Others: the aforementioned list is not supposed to be exhaustive, so the applicant must consider any type of interference not mentioned in the position paper but observable on the platform.*

Definition 2 (Interference channel)**.** *An interference channel is a platform property that may cause interference between independent applications.*

Objective 3. *The applicant has identified the interference channels that could permit interference to affect the software applications hosted on the MCP cores, and has verified the chosen means of mitigation for those interference.*

Example 3. *Let us again consider the architecture of example 1 and let assume that two applications run on top of the real-time kernel, one mapped on $core_1$ and the other on $core_2$. If both applications simultaneously access the same bank, there may be a contention at the interconnect level, at the memory controller level or at the bank level. Thus, the applicant must assess whether the interference is acceptable or not. In the second situation, they must define a means of mitigation.*

- *Identification of the effect: After a series of benchmarks, the applicant has observed that the interference on the interconnect and the memory controller are acceptable, whereas the sharing of a common bank is not acceptable;*
- *Mitigation solution: Applications executing on different cores are allocated to separate banks;*
- *Validation of the means of mitigation: for the chosen solution, the applicant must argue that it is appropriate.*

III. PHYLOG FRAMEWORK

The main objective of PHYLOG is to define a new certification framework in order to introduce model-based documentation and automatic verification for multi-core-based systems that need to satisfy the CAST32-A requirements. That proposed overall approach is shown in figure 2.

There are two inputs: the MCP-CRI / CAST-32A position paper and some external documents. The latter could be the documentation on the multi-core or the results of some external activities (e.g. an safety analysis made on the multi-core by safety experts).

A. Design choices input

Another input is the documentation on the design choices. This should include the detailed description of the configuration settings and the chosen means of mitigation. Those information must be specifically traced as they are the basis of the evidences required on the argumentation process.

B. Argumentation pattern-based approach

Our first contribution is to rely on *justification patterns*, which are templates allowing their users to express a structured reasoning with a set of evidences and a conclusion. We have translated each CAST32-A objective as *generic argumentation patterns*. An example of generic argumentation pattern is given for the RU2 objective in section IV-B.

C. Model-based documentation approach

We have defined a very generic *model*, in the sense that it is not dedicated to a specific platform. The PML (PHYLOG meta-model) has been introduced in [BBB+18]. The idea is to represent the architecture, including its hardware and the executive layers. More specifically, the applicant must provide the configuration settings, indicate which shared resources are used and how, and specify which interference channels exist. This model has been applied on parts of two real platforms, the T4240 [Fre14] and a Kalray MPPA cluster [Cor12].

D. Software-aided framework

We then need to select formal methods to automatically analyze a PML model and show its compliance with some requirements (refined as a justification pattern). As an example, we can currently automatically compute the topological interference channels with a solver in an efficient way [BBB+18]. The topological analysis consists in computing all potential topological paths, and our enumeration is an equivalence class compared to the automatic enumeration of [MJB+17]. With the different models defined in the *argumentation*, *model* and *design choices* worlds, we can make some analyses:

- the first of these analyses is *checking* that the different models are all coherent. As an example, if, in the configuration, it is stated that 3 cores are used, the model description of the architecture must contain at least 3 cores;
- some requirements could be analyzed by an automatic tool. This is the case for part of RU3, which requires the enumeration of all interference channels;
- formal methods could also help prove the coverage of a requirement by a set of evidences.

E. Validation of the approach

Another important aspect is the validation. We must offer tools that will: first, prove that a model is compliant with the real system; second, ensure the coverage of the CAST32-A requirements. This is a future work of the project.

IV. ARGUMENTATION-BASED METHODOLOGY

A. A need of justification

A system is said to be *critical* if it has the potential to endanger a person's life. Those systems are subject to certification, not only in aviation, but also in several other domains. Examples of certification authorities are: the European Aviation Safety Agency (EASA), and the Federal Aviation Administration (FAA), for civil aviation; the Department of Defense (DoD) for military matters; the European Medicines Agency (EMA), and the Food and Drug Administration (FDA), for drugs and medical devices. In any case, the applicant must convince the relevant authority that their system is operable. For this, they must provide elements of the design and Verification and Validation (V&V) operations that have been carried out. To structure, organize and share all these V&V items between stakeholders, more and more organizations are using *assurance cases*.

An *assurance case* is defined by [RKR15] as: "*an organized argument that a system is acceptable for its intended use with respect to specified concerns (such as safety, security, correctness)*". Thus, assurance cases cover a broad range of issues, including security [AHK11] and dependability [WGH04]. As assurance cases could be complex, lots of work in the literature [McD94], [KW04], [EC02], or carried out by standardization organizations [OMG13], [ISO11], propose ways to standardize these practices to make them intelligible.

In its study on Certifiably Dependable Software Systems [Cou07], the National Research Council points out that: "*concrete evidence must be present that substantiates the dependability claim. This evidence will take the form of a "dependability case," arguing that the required properties follow from the combination of the properties of the system itself (that is, the implementation) and the environmental assumptions*". The whole problem here is, for the designers, how to argue well, and, for the certification authority, how to evaluate an argumentation. Indeed, the assurance case is an argumentative process in which one has a *conclusion* that one seeks to establish on the basis of evidences. Obviously, the conclusion is not formally provable as it results from the evidences and a plausible link from the evidences to the conclusion. Such a process belongs to the context of *argumentation*. In the sequel we will use equally *argumentation* or *justification*.

Based on the Toulmin schema [Tou03], we have introduced the concept of *Justification Diagrams* [Pol16b] to organize in diagram form the various elements, formal and informal, that contribute to the justification of a result. For the authority, a justification diagram gives a rational view of the documentation and, for the development teams, it gives – as a sort of a recipe – the list of necessary evidences. Experiments on the use of justification diagrams have already been conducted in various domains and contexts:

- in an European project, TOICA, to structure all justifications that would be needed to convince an authority that a thermal simulation process, and the associated results, upheld a particular conclusion [PSCT18];
- in a project with the French aerospace technical center, to build a model-based frame of reference for the certification of embedded modular architectures [BBD+16];
- in the context of medical experimental studies and medical software certification [DCBF17], [DPB18].

What has emerged from these experiments were the following key points: most of the time, justification diagrams are used to define patterns, in the sense that a common structure could be re-used in several cases. Then, in practical cases, the justification diagrams are an *instantiation* of a pattern that has been defined for a generic solution. In that case, it is better suited to speak of *Justification Patterns* (JPs) rather than justification diagrams. A justification pattern can be seen as a template dedicated to a specific conclusion, a specific property. Thus, for a given conclusion, generic patterns can list the necessary evidences as well as the means which make it possible to justify this conclusion. These patterns are the result

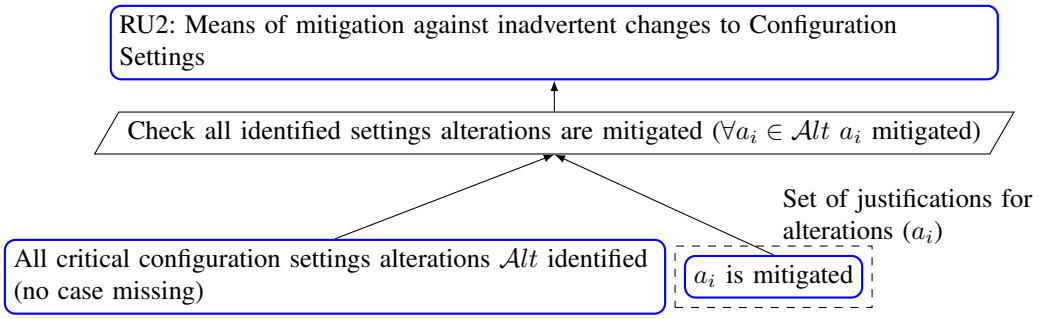

Fig. 3. JP-RU2: Means of mitigation against inadvertent changes to Critical Configuration Settings.

of good practices and experts consultations. In the context of certification, they can be provided by the authority or by the system designers. In addition, the patterns can be a part of the definition of means of compliance defined by the authority and the stakeholders. We will have a deeper look at the notion of *justification pattern* and *justification diagram* by applying it to the RU2 objective.

B. RU2 justification pattern

Let us structure the RU2 objective in the form of *justification pattern*. Making such a pattern is like eliciting justification requirements that fall within the scope of CAST32-A. In the case of RU2, the applicant must ensure that any critical settings that could be altered must be mitigated appropriately. The pattern of Figure 3 focuses on establishing that the means of mitigation are trustable. The conclusion is on the top of the figure. It relies on two evidences:

- on the left hand side of the figure, a safety analysis has correctly identified all conditions that can alter the configuration settings, has classified them according to high level safety requirements to extract the critical ones (set $\mathcal{A}lt$) and has been itself validated (there are no other condition that may alter the configuration settings to something which is not in $\mathcal{A}lt$). This leaf is an evidence, no further details in the argumentation tree is provided;
- on the right hand side, a set of justifications that critical configuration settings are appropriately mitigated. Each configuration settings alteration mitigation has its own justification. This second leaf could be a document or a new justification step. In that case, the leaf is a sub-conclusion that will become the conclusion of a new justification pattern (see next section).

The passage from the evidence to the conclusion is based on the argument that a verification step has been done. For RU2, the check consists in verifying that, for each possible alteration, a justification of its mitigation is provided. The graphical symbol for this step is the *strategy*: "*Check all identified settings alterations are mitigated ($\forall a_i \in \mathcal{A}lt$ a_i, mitigated)*". The strategy here refers to a document that explains how this verification was done.

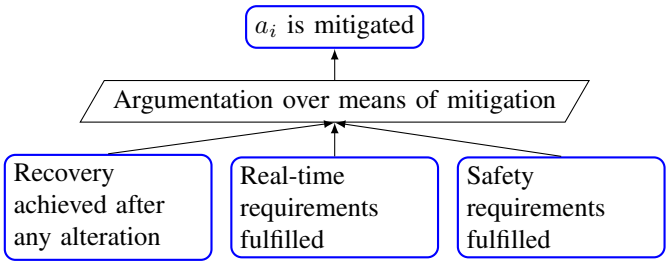

Fig. 4. JP-RU2-SC: a_i is mitigated.

C. RU2 sub-conclusion

The right hand side of Figure 3 is a sub-conclusion that needs some further refinement, which is given as the pattern of Figure 4. The pattern applies to all potential alteration and has to be instantiated for every possible condition leading to an alteration. The justification is organized as follows: first, the applicant must argue how the means of mitigation performs the recovery correctly; second they must justify how the means of mitigation fulfilled its Real-time and Safety requirements. The evidence for *Recovery achieved after any alteration* could be for instance a safety analysis result demonstrating that, if the resources used by the means of mitigation are available, then the effect of the alteration is corrected. The two other evidences are again refined as justification patterns.

D. JP-RU2-SC sub-conclusions

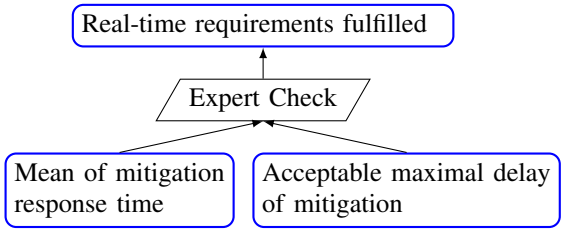

Fig. 5. JP-RU2-SC-RT: focus on the real-time requirements.

The pattern Figure 5 focuses on the real-time requirements of the sub-conclusions from pattern 4. The justification is composed of two evidences: the *acceptable maximal delay* before mitigating an alteration and a justification of why

this delay is acceptable. The aim here is to justify that the implementation respects the real-time requirement. For this, the strategy is based on expert verification that the means of mitigation's response time is below the maximal delay time. The evidence relating to the mean of mitigation's response time may link, for example, to a worst-case response time (WCRT) analysis which takes WCETs and a scheduling policy as inputs. The evidence relating to the maximal delay time links to the real-time requirements document.

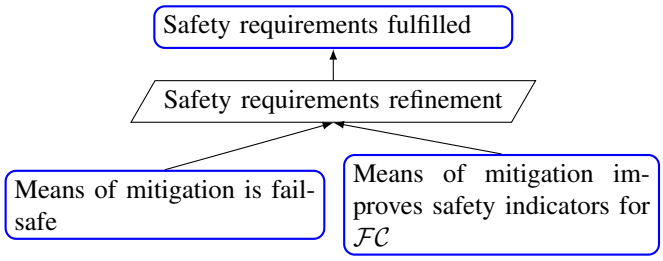

Fig. 6. JP-msr: Means of mitigation fulfilled the Safety requirements.

The pattern Figure 6 focuses on the safety requirements and relies on two evidences:

- on the left hand side, the justification that, in case of one or several resource losses, the means of mitigation will be fail-safe *i.e.* will not cause additional losses of resources (for instance by corrupting external resources). These justifications can be provided by safety analysis results;
- on the right hand side, a set of justifications that if the means of mitigation's resources are available then the safety indicators (probability and cutsets size) will be better, for each failure condition $fc \in \mathcal{FC}$. Those failure conditions come from the initial system (*i.e.* without the means of mitigation). Note that failure conditions are inherited from a preliminary system safety assessment and should be identified in the Error Handling objective. The justifications here can be provided by safety analysis results.

V. Application of the methodology for RU2 on use case 1

Let us apply the methodology on RU2 objective on the 1 example. First, according to the justification pattern RU2 (see figure 3), the applicant must identify all the critical configuration settings and how each of them could be altered. As explained in example 2, since the core frequency could be modified at run-time by an adversarial event, such as an SEU, and since the effect on the applications could be a serious slow-down, the frequency register configuration settings is classified as *critical*. In the sequel, we focus in this critical settings and we argue that it is mitigated following pattern of figure 4.

A. Design choice: means of mitigation – trusted value pattern

For the means of mitigation, we elaborate on the solution previously introduced in example 2. This mechanism, called the *trusted value pattern*, works as follows: the means of mitigation M reads the core register value and the trusted copy. If they disagree, M reads several times the trusted copy until reaching a consensus on the value or a maximal number of attempts, in which case the alteration reparation fails. If a consensus has been reached on a value then there are two cases. Either the value is not equal to the register, in which case M will write the computed value in the register and the alteration is corrected, or the value is equal to the one in the register, in which case no alteration has occurred.

M executes on core_1 every 10 ms on the real-time kernel which is supposed to be Arinc 653 [Aer97] compliant. Thus, M executes as a partition with a period of 10ms and a slot length of 1ms.

B. Recovery achieved after any alteration

This evidence is a document summarizing the different verifications and testings that have been done on the implementation. We will not go further on this part, as it is a usual activity as required by the DO 178.

C. Safety requirements argumentation

1) Fault model: To perform the safety analysis, one must define the failure modes of each component and the propagation of these local failures within the system, which, in our case, is the multi-core. Let us consider that any component of the architecture of figure 1 can fail, *i.e.* does not provide the expected service to the application. The identified failure modes are:

- the cores can be erroneous (incorrect execution of the software request) or lost (no processing of the request);
- the frequency registers can be corrupted;
- the MMU can be lost (no transaction between the core and the rest of the platform);
- the cache can be erroneous (corruption of the cache content);
- the DDR controller can be lost (all memory accesses are discarded);
- the banks can be erroneous (corruption of the content);
- corruption occurring during the traversal of the read request in the network and the DDR controller will never produce the same erroneous value. So any corruption of the request is detected by the mitigation algorithm

An application transaction is properly processed by the hardware if every involved component is in its nominal mode. For instance, the transaction realized by *M* when reading the trusted value involves 1) $Core_1$ that sends an access request to the memory/cache through the MMU; 2) the MMU_1 that controls and redirects the request accordingly; 3) $Cache_1$ and $Memory_1$ as we don't know if the data is in the cache or should be retrieved from the memory; 4) the network and the DDR controller that dispatch the request.

The two evidences asked by the safety pattern Figure 6 are that the *means of mitigation is fail-safe* and that the *means of mitigation improves safety indicators*. In order to assess the validity of both evidences, we consider the cases when that

they are not fulfilled and represent them as the two failure conditions:

FC1: not fail-safe : the trusted-value pattern M corrupts the core register;

FC2: no improvement : the trusted-value pattern M does not correct the frequency registered when altered.

With the safety analysis, we then want to show that these two failure conditions are improbable if not impossible.

2) Evidences in the form of safety analysis: The dysfunctional model has been done using ALTARICA [APGR99] and its associated tool CECILIA OCAS [Avi15]. All the sequences of failure events leading to the considered failure conditions have been computed, when considering that a sequence is composed of up to three failures.

Analysis 1 (FC1: not fail-safe)**.** *The means of mitigation can only corrupt the register when: 1)* $Core_1$ *executes incorrectly* M*; 2)* $Bank_3$ *that contains the trusted copy is corrupted. Thus the cut sets are:*

$$\{\{Core_1.err\}, \{Bank_3.err\}\}$$

Argumentation 1 (FC1 is acceptable)**.** *If the first cut set occurs, that is* Core_1 *is erroneous, then the frequency register is also highly probably altered. If the failure* Core_1 *erroneous is permanent, the core must not be used anymore, which has to be ensured by a means of mitigation at the Error Handling requirement level and in that case the frequency problem is the least of the issues. If the failure is temporary, then the next execution of* M *will be valid and the register will be corrected.*

If the second case occurs, an hypothesis of the design choice was that the Bank *failure is highly improbable.*

Analysis 2 (FC2: no improvement)**.** *The mitigation of a register corruption fails if and only if one of the resources involved in the execution of* M *is unavailable. That is 1) the core on which* M *executes is unavailable; 2) the MMU or the DDR controller systematically discard the read request.*

$$\left\{ \begin{array}{l} \{Core_1.lost\}, \\ \{Core_1.Reg.fail, \quad MMU_1.lost\}, \\ \{Core_1.Reg.fail, \quad DDRController.lost\} \end{array} \right\}$$

Argumentation 2 (FC2 is acceptable)**.** *If* Core_1 *is lost, the frequency issue is of least importance. The Error Handling must mitigate the loss.*

For the other cases, M *improves the safety indicators as there are 2 cut sets that are of length 2, instead of a single cut set of length 1. Moreover, if* MMU_1.lost *or* DDRController.lost *are permanent, the problem will, once again, be managed at the Error Handling level.*

D. Real-time requirements argumentation

The last missing evidence concerns the real-time requirements fulfillment, see pattern of figure 5. The *acceptable delay of mitigation* has been computed and validated in a document written by safety and real-time experts. The value is 50ms. A real-time analysis provides the maximal time needed by M to correct an alteration which is 30ms. Indeed, if an alteration occurs just at the moment where M starts, it will take 2 periods to correct the register. Moreover, the experts take an additional margin of 1 period in case M suffers from temporary failures when executing. The worst-case situation is shown in figure 7. The strategy of JP-RU2-SC-RT checks that $30 < 50$.

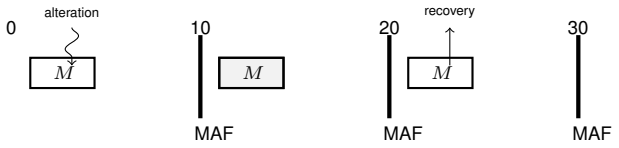

Fig. 7. M worst-case scenario schedule

VI. APPLICATION OF THE METHODOLOGY FOR RU2 ON USE CASE 2

Let consider another architecture inspired from a cluster of the Kalray MPPA [Cor12]. We only focus on 4 cores that are directly connected via hardware link. The executive layer is bare-metal.

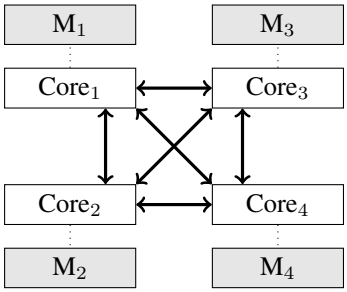

Fig. 8. Cluster architecture

A. Design choice: means of mitigation – cross-register check

Let us introduce another means of mitigation, called the *cross-register check*. For this architecture, on each $Core_i$ a means of mitigation M_i (4 identical instances of the same pattern) executes. M_i reads the frequency values of the three other cores plus its own register value, and writes the consensus value to its own register if it disagrees with the consensus.

As the executive layer is bare-metal, on off-line scheduling [PNP15] is computing ensuring by construction that M_i executes alternatively, with no temporal overlapping. Each M_i executes with a period of 10 ms and their WCET is below 1ms.

We summarize in the justification diagram of figure 9 the organization of the documentation and analyses. This diagram is an instantiation of the justification pattern of figure 4 for a particular architecture and a particular critical configuration settings.

B. Recovery achieved after any alteration

This evidence is again a documentation on the implementation and the V & V activities.

Document 1. *Reference to development document summarizing traceability, V & V activities, code review ...*

C. Safety requirements argumentation

1) Fault model: We consider that an erroneous behavior of the $core_i$ leads to the halting of M_i.

2) Evidences in the form of safety analysis: We, once again, use ALTARICA to assess the safety requirements on this architecture. The requirements are similar to the 2 failures conditions FC1 and FC2 of the previous use case.

Analysis 3 (FC1: not fail-safe)**.** *The safety assessment demonstrates that there are no failures of hardware component that could lead to the corruption of a core register by the cross-check register pattern. This comes from the fact that the loss of the mitigation resources (the register access and the core) is either impossible (safe register access) or leads to an harmless loss of the mitigation software (core loss).*

Argumentation 3 (FC1 is acceptable)**.** *This mitigation pattern is, unlike the previous use case, completely fail-safe.*

Analysis 4 (FC2: no improvement)**.** *The sequences leading to a mitigation failure are listed below. These scenarios highlight the only weakness of the pattern, i.e. a triple corruption of the registers.*

$$\left\{\begin{array}{lll} Core_1.Reg.fail, & Core_2.Reg.fail, & Core_3.Reg.fail \\ Core_1.Reg.fail, & Core_3.Reg.fail, & Core_4.Reg.fail \\ Core_1.Reg.fail, & Core_3.Reg.fail, & Core_4.Reg.fail \\ Core_2.Reg.fail, & Core_3.Reg.fail, & Core_4.Reg.fail \end{array}\right\}$$

Argumentation 4 (FC2: no improvement)**.** *The means of mitigation clearly improves the probability of system failure conditions. Indeed, this pattern abides by the fail-safe requirement and replaces the event $Core_i.Reg.fail$ by four cutsets of size three. As a result, the cross-check register pattern enhances the safety indicators on all $fc \in \mathcal{FC}$ involving $Core_i.Reg.fail$.*

D. Real-time requirements argumentation

As with the previous use case, the *acceptable delay of mitigation* has been computed and validated in a document written by safety and real-time experts. The value is 50ms.

Document 2. *Reference to the design document that explicitly states the value to be 50 ms.*

Similarly to the previous use case, a real-time analysis provides the maximal time needed by the union of M_i to correct an alteration which is 30ms.

Analysis 5 (Maximal time for recovery)**.** *Result of the real-time analysis which considers the case where the alteration occurs right after the beginning of M_i and which is corrected at the next valid execution of M_i assuming that one occurrence may fail.*

VII. CONCLUSION

We have presented an overview of the PHYLOG framework and detailed an argumentation-based methodology rooted in graphical and structured notation techniques. Furthermore, we have illustrated its capabilities by applying it to the RU2 objective, through two detailed case studies.

In the future, we plan to provide similar guidelines for all of the MCP-CRI / CAST-32A objectives. We will also provide efficient exploration techniques for the interference analysis, not only based on topological description, but also taking into account the execution model. We will develop a prototype to support the use of PHYLOG that will include automatic checking and formal verification.

Finally, we will define means of validation to verify that a model is compliant to the real architecture. This means will be based on generic benchmark that could either validate some assumption but also stress the platform according to some evidences used in the justification pattern (e.g. when an interference has been categorized as acceptable).

REFERENCES

[Aer97] Aeronautical Radio Inc. *Avionics Application Software Standard Interface*, 1997.

[AHK11] Rob Alexander, Richard David Hawkins, and Tim Kelly. *Security Assurance Cases: Motivation and the State of the Art*. Department of Computer Science, University of York, 2011.

[APGR99] André Arnold, Gérald Point, Alain Griffault, and Antoine Rauzy. The altarica formalism for describing concurrent systems. *Fundamanta Informaticae*, 40(2-3):109–124, 1999.

[Avi15] Dassault Aviation. *Cecila OCAS*, 2015.

[BBB+18] 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.

[BBD+16] Pierre Bieber, Frdric Boniol, Guy Durrieu, Olivier Poitou, Thomas Polacsek, Virginie Wiels, and Ghilaine Martinez. MIMOSA: Towards a model driven certification process. In *Proc. 8th Int. Congress on Embedded Real Time Software and Systems (ERTS'16)*, 2016.

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

[Cor12] Kalray Corporation. *The MPPA hardware architecture*, 2012.

[Cou07] National Research Council. *Software for Dependable Systems: Sufficient Evidence?* The National Academies Press, Washington, DC, 2007.

[DCBF17] Clément Duffau, Cécile Camillieri, and Mireille Blay-Fornarino. Improving confidence in experimental systems through automated construction of argumentation diagrams. In *ICEIS 2017*, 2017.

[DPB18] 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, June 11-15, 2018, Proceedings*, volume 10816 of *Lecture Notes in Computer Science*, pages 71–86. Springer, 2018.

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

[EAS17] EASA. Certification Specifications and Acceptable Means of Compliance for Large Aeroplanes CS-25 - AMC 1309. Technical report, 2017.

[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.

[Fre14] Freescale. T4240 QorIQ: Integrated multicore communications processor family reference manual, 2014.

[ISO11] Systems and software engineering – Systems and software assurance – Part 2: Assurance case. Standard, International Organization for Standardization, 2011.

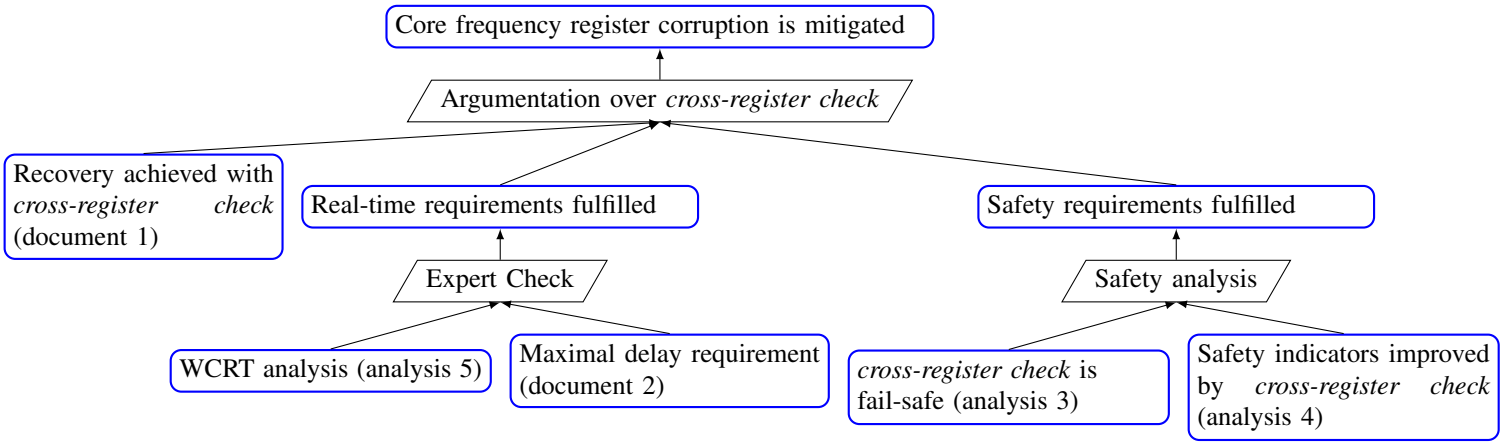

Fig. 9. Argumentation diagram of register cross-check

[KW04] Tim Kelly and Rob Weaver. The goal structuring notation /- a safety argument notation. In *Proc. 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.

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

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

[PNP15] Wolfgang Puffitsch, Eric Noulard, and Claire Pagetti. Off-line mapping of multi-rate dependent task sets to many-core platforms. *Real-Time Systems*, 51(5):526–565, 2015.

[Pol16a] 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.

[Pol16b] Thomas Polacsek. Validation, accreditation or certification: a new kind of diagram to provide confidence. In *10th IEEE International Conference on Research Challenges in Information Science, RCIS,*, pages 59–466, 2016.

[PSCT18] Thomas Polacsek, Sanjiv Sharma, Claude Cuiller, and Vincent Tuloup. The need of diagrams based on toulmin schema application: an aeronautical case study. EURO Journal on Decision Processes, to appear, 2018.

[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.

[RTC05] RTCA, Inc. DO-254 - Design Assurance Guidance For Airborne Electronic Hardware, 2005.

[RTC08] RTCA, Inc. DO-178 ED-12B - Software Considerations in Airborne Systems and Equipment Certification, 2008.

[RTC11a] RTCA, Inc. DO-178 ED-12C - Software Considerations in Airborne Systems and Equipment Certification, 2011.

[RTC11b] RTCA, Inc. DO-333 - Formal Methods Supplement to DO-178C and DO-278A, 2011.

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

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

[WGH04] Charles B Weinstock, John B Goodenough, and John J Hudak. Dependability cases. Technical report, Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst, 2004.