A service-based modelling approach to ease the certification of multi-core COTS processors

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

A service-based modelling approach to ease the certification of multi-core COTS processors

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

January 15, 2020

1 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. Among those, two types of analysis are required: interference and safety analyses. Because of the large size of the platforms and their complexity, those analyses can lead to combinatorial explosion and to some misinterpretation. To tackle these issues, we explore a service-based modelling approach that leads to a simplification of the analyses and to the highlighting of salient properties, making the adaptation of the certification argumentation efficient.

2 Introduction

2.1 Context

An aircraft is allowed to become operational only if the manufacturer has been issued a type certificate from the certification authorities. For that to happen, the aircraft manufacturer has to demonstrate the compliance of their product with the regulatory requirements [EAS17]. An example of accepted mean of compliance with the requirements is to rely on mature standards. For multi-core-based systems, the Multi-Core Certification Review Item (MCP-CRI) [EAS16] (also known as the CAST32-A position paper [Cer16]) provides a set of guidance for software planning and verification on multi-core chips.

PHYLOG[1] is a four years long DGAC (French Civil Aviation) project (2016-2020) that aims at offering a model-based software-aided certification framework for aeronautical systems based on multi/many-core architectures. More precisely, its goal is threefold: to structure the certification arguments; to reduce the amount of textual documentation as much as possible, by replacing it with model(s) wherever possible; to promote automatic analysis and to replace parts of the testing with formal methods, as recommended by the DO333 [RTC11].

2.2 PHYLOG approach

When making a case before a certification authority, an *applicant* has to provide all the elements related to the design of the system and the Verification and Validation (V&V) operations that have been carried out. All of these elements are generally referred to as an *assurance case*. To reach this objective, we have defined a certification framework based 1. on justification patterns to express any argumentation; 2. and on formal and automatic analyses to support the proof of the argumentation.

The idea is to organize any argumentation around structured graphical notations diagrams. More precisely, *justification patterns* are templates specifically dedicated to the expression of a structured reasoning based around a set of evidences. We have translated most of CAST32-A objectives as *generic justification patterns*. The basics of this approach have been defined for the certification of IMA systems in [BBD+16, Pol16]. The use of generic justification patterns for the CAST32-A has been partially presented in [BBB+18b].

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

A justification 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*. Because of the large size of multi-core platforms and their complexity, those analyses can lead to combinatorial explosion and some misinterpretation. We have presented a *topological interference* analysis in [BBB+18a, BPS19] and a preliminary safety assessment analysis in [DPC19].

2.3 Contributions

The purpose of this paper is to unify both analyses and to provide a more accurate representation of multi-core processors. The main idea is to abstract a platform as a set of services (e.g. *execute*, *load*) and a set of intrinsic properties (e.g. partitioning). This leads to the consideration of 1. a common model for any analysis, 2. specific analyses according to the set of properties associated to a platform, 3. dedicated justification patterns, and 4. a classification of platforms (for re-use of existing justifications).

The outline of the paper is as follows: the next section introduces the MCP-CRI / CAST-32A expectations and the way Phylog answers these objectives. The section **Service-based modeling approach** describes the modeling of any hardware as a set of abstract services and how to define an interference and a safety analysis.

3 Required justifications

An applicant must master its architecture and control, in any worst case scenario, both the safety effects and the timing behaviours of the multi-core-based system.

3.1 Terminology

A first step towards profound understanding of a given multi-core platform is the definition of a clear and formal terminology. For that purpose, we reuse and extend the initiator-target model introduced in [BR14, JMB16, MJB16, MJB+17]. Thus, a multi-core is composed of four types of components:

- *Smart initiator* components, *i.e.* components which can initiate single transactions through the architecture to *target* components.
- *Non smart initiator* components, *i.e.* initiator components which can only initiate dual transactions (*i.e.* with two targets at the same time).
- *Target* components, *i.e.* end-components targeted by initiators.
- *Transporter* components, *i.e.* any intermediate components between initiators and targets.

Example 1 (Component types) *Let us consider the simplified representation of the Keystone TCI6630K2L [Tex13a] from Texas Instruments depicted Figure 1. This platform is composed of:*

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

The components of Figure 1 are ***coloured*** *according to their type, the color code being: 1. red for smart initiators i.e. the cores; 2. blue for transporters i.e. the TeraNet, the memory access controllers (MMU, MPPAX, MSMC, EMIF), and the memories used as caches here ARM L1 since, in this approach, caches are considered as proxies of the central memory; 3. green for targets i.e. non-cached memories (e.g. SRAM, C66 L1 and L2, DDR, IO); 4. and orange for non-smart initiators i.e. DMA (EDMA). For the sake of simplicity, we will consider that peripherals and the EDMA are deactivated, and are therefore omitted in the remainder of the paper.*

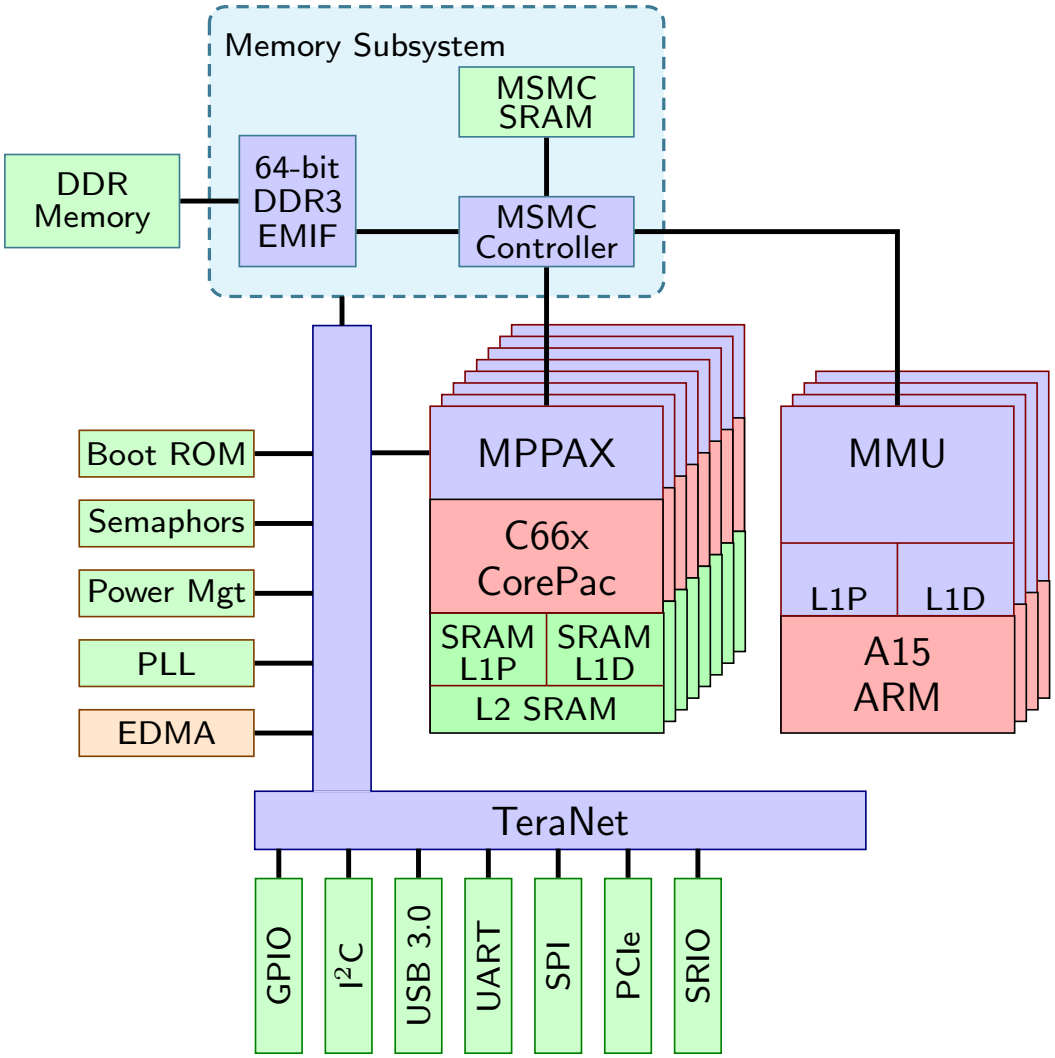

Figure 1: Keystone – simplified view

3.2 Interference analysis

Due to resource sharing, coupling exists at the platform level, which can cause interferences between the applications. For instance, let us consider again the Keystone TCI6630K2L and let us suppose that one ARM core and one C66 DSP try to access the MSMC SRAM simultaneously (as shown in Figure 2). As the MSMC controller can only serve a single transaction at a time, such simultaneous accesses provoke a contention in the controller. The result of such contentions is an *interference* between the two initiators (the ARM and the DSP). Such behaviour is non-deterministic, as at least one transaction can (possibly) suffer from an (unexpected) delay. Certification requires interference situations to be fully controlled and mastered in any configuration for safety critical applications. This requirement is called *resource usage 3 – RU3* in the MCP-CRI. 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.

The PHYLOG approach consists in eliciting justification requirements from certification standards and guidelines. For that, we use a generic notation that allows modularity and hierarchy to help describe elementary argumentation steps and combinations of steps. An argumentation is then represented as a *justification pattern*, whose conclusion is a given objective. The justification patterns are in the tradition of the design patterns [GHJV95]. If a design pattern gives a generic solution for a type of problem, a justification pattern organizes the justifications underlying a claim [RXRW15, Hol15, Pol16, DPBF18]. The benefits of justification patterns are that they give an overview of all justification requirements, they elicit links between justifications, and they list necessary evidence related to the certification. Here, we

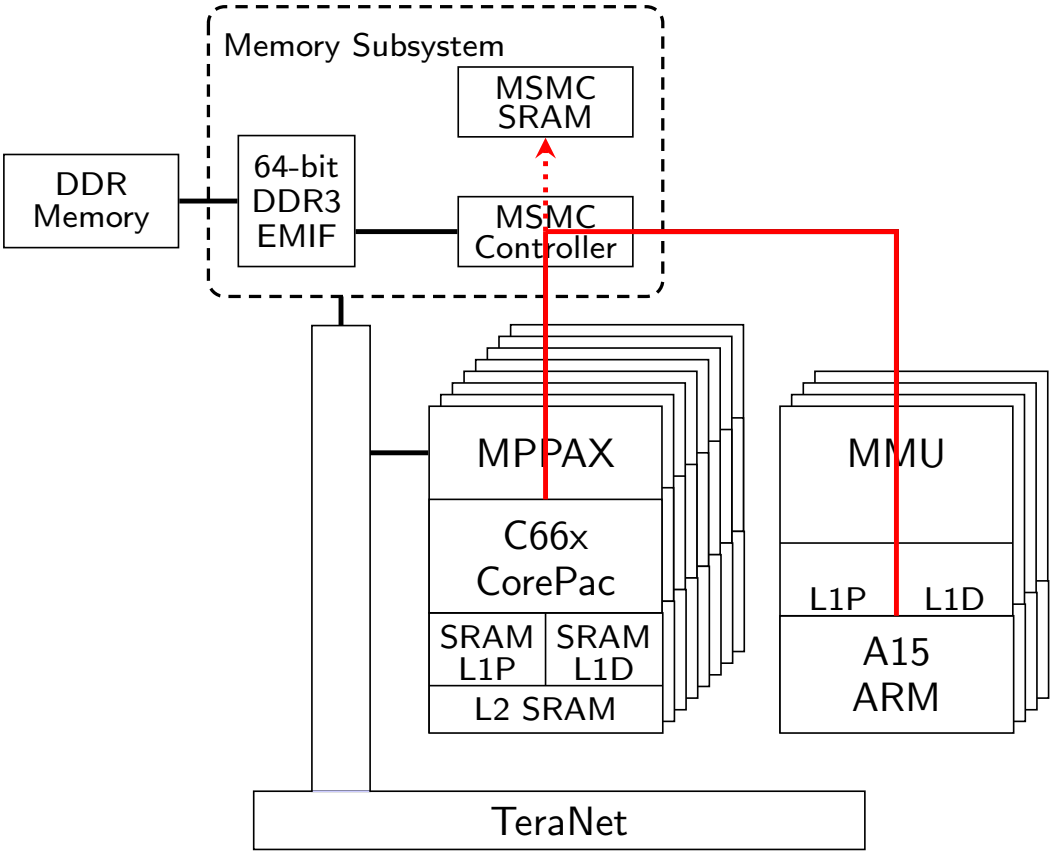

Figure 2: Example of interference

use a simplified version with three concepts: *Goal*, the property that must be demonstrated; *Evidence*, any relevant document or piece of information (for instance calculation results, test reports or expert judgments); and *Strategy*, which corresponds to the explanation of how it is possible to conclude the Goal from the Evidence. Note that, Evidences can be of two types, either final evidences (leading to a documentation or the results of an analysis) or sub-goals leading to new patterns.

We have defined a justification pattern for several objectives of the MCP-CRI / CAST-32A and illustrate the RU3 pattern in Figure 3. The top level Goal "*Identification of interference and verified means of mitigation*" is precisely the RU3 objective (the text of which has be reduced for readability of the pattern). It is possible to conclude this Goal because "*all interferences were correctly identified*" (evidence on the left) and because adequate mitigation means were designed for all identified interferences (evidence on the right). The strategy used to conclude here is a check that all identified interferences are mitigated. This verification can be done manually or automatically, regardless of whether the means of verification can be certified.

The evidence on the left is a sub-goal that leads to a sub-justification pattern. The sub-goal is considered to be reached because the interferences were identified with a certain level of confidence (evidence on the left) and those have been "*classified*" with respect to their effects on the software hosted on the platform (evidence on the right). The Strategy is a check that all identified interferences are classified.

In terms of activities, an applicant has to define a process composed of at least the four following steps: First, the applicant has to identify all *interference channels*. In the CAST-32A terminology, an interference channel is *a platform property that may cause interference between independent applications*. Second, the applicant has to classify each interference as either *acceptable*, *tolerable*, or *unacceptable*. Third, for each *unacceptable* interference, they have to provide a mean of mitigation which prevents the system from having catastrophic behaviors. In this context, mitigation signifies that some mechanisms have been proposed to forbid *unacceptable* interference or to lower their effect down to *acceptable* or *tolerable* levels. Fourth and final, the applicant has to argue why the means of mitigation are adequate and why *unacceptable* interferences will never occur during aircraft operations.

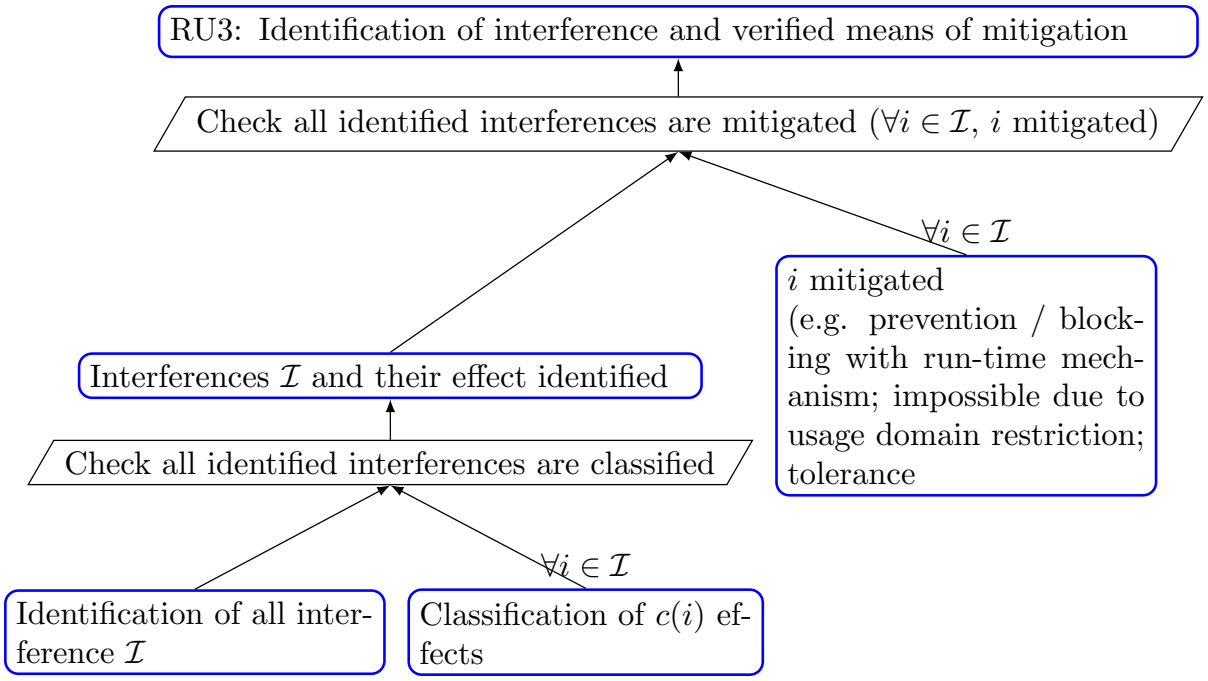

Figure 3: Justification pattern for RU3

Example 2 (RU3 satisfaction) *Let us consider again the architecture of Figure 2 and let us suppose that a safety-critical application a_1 is running on the first ARM core which only accesses the MSMC SRAM memory through the MSMC controller.*

Identification of interferences *The applicant has to first identify all the interferences and, in particular, those in which the application is involved. Few works have proposed solutions for how to compute them. Researchers from Thales have proposed an enumerative technique for the* Initiator-Target Model *[BR14, JMB16, MJB16, MJB+17] to help identify the interference channels on multi-core chips. Their model is very simple, but suffers from a combinatorial explosion. In* PHYLOG*, we have also proposed a more efficient yet equivalent method [BBB+18a, BPS19] to compute the interference channels. Instead of enumerating all possible interferences, our method generates only representative elements of the equivalence classes of an equivalence relation.*

Classification of interferences *For each identified interference, the applicant must derive temporal properties (e.g. delays). One possibility is to conduct intensive benchmarks [WKRP05, GJlR+15, GlRS18] to estimate the temporal delays induced by contention. Let us suppose that such an hardware analysis has shown that 1. the two transactions of Figure 2 generate an unacceptable delay for a_1; 2. the delay generated by two transactions (one from an* ARM *core to the SRAM and the second from the C66x* DSP *to the DDR) is negligible.*

Mitigation means *The applicant has to design and implement mitigation means in order to prevent unacceptable interferences (such as the one of Figure 2) or to make them acceptable. A mitigation mean (see Figure 4) could be 1. to prevent the C66x* DSP *from accessing the MSMC SRAM by configuring the MPPAX protection unit as a barrier between the* DSP *and the SRAM, and 2. to design a time driven execution model on the four* ARM *core pack in order to prevent multiple accesses to the controller.*

3.3 Safety analysis

Due to the high integration density, the complexity of internal components and the mix of hardware / software inside multi-core chips, their behaviours 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. 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.

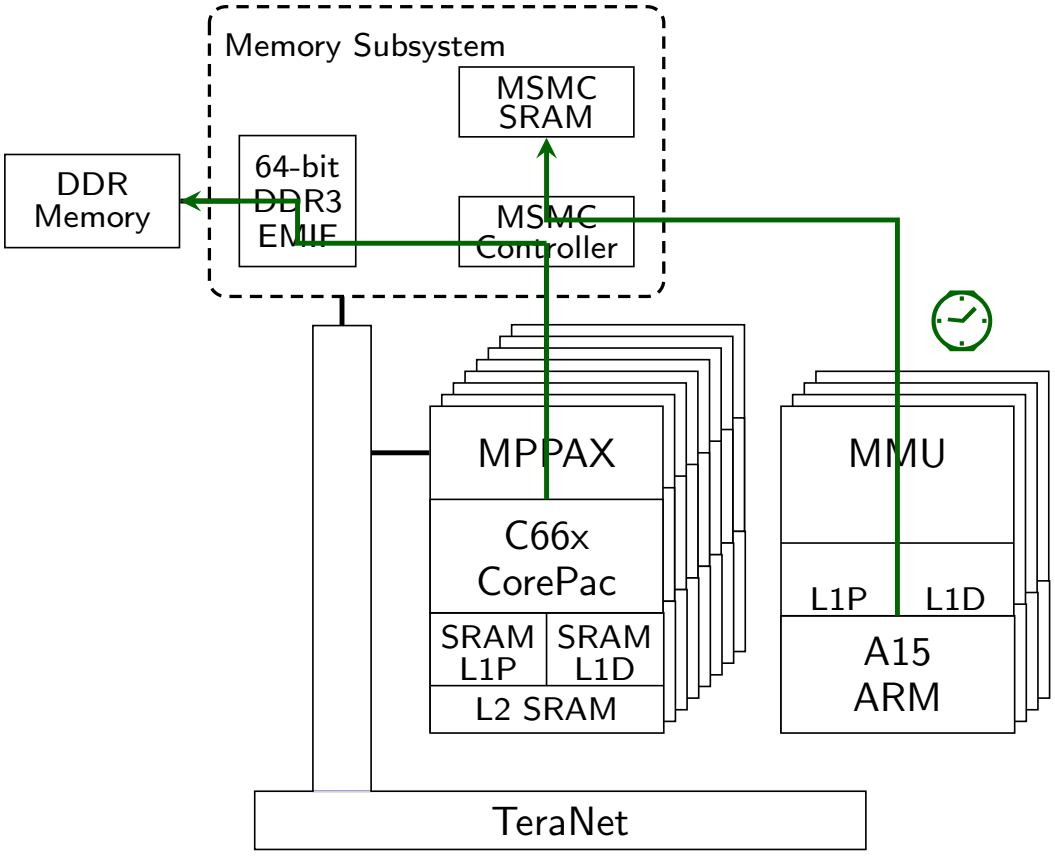

Figure 4: Example of acceptable interferences

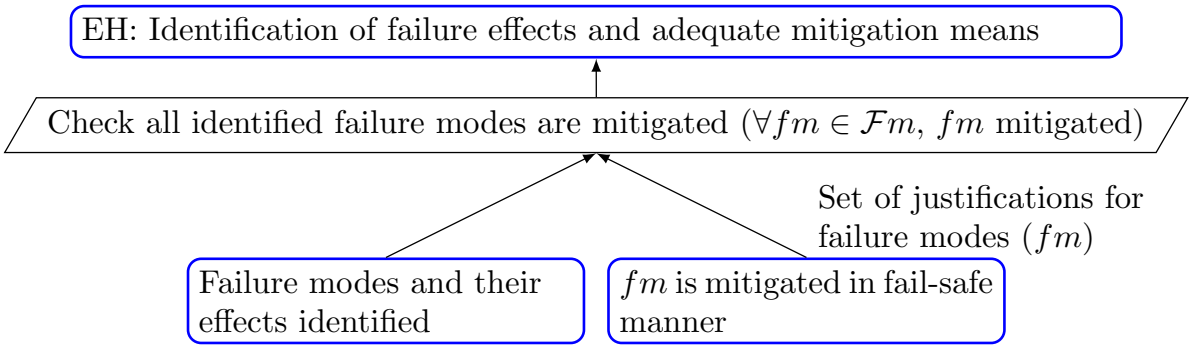

Figure 5: Justification pattern for EH

As depicted by Figure 5, the argumentation structure for EH is quite similar to the one of RU3. The top level Goal is EH with a shorten textual description "*Identification of failure effects and adequate mitigation means*". It is possible to conclude this Goal because "*all failure modes and their effects were correctly identified*" (evidence on the left) and because adequate mitigation means were designed for all identified failure modes (evidence on the right). Here again, the strategy used to conclude is a check that all identified failure modes are mitigated, independently of the quality and certifiability of the means of verification.

The evidence on the left can be achieved by a safety analysis identifying the failure modes of internal components and the safety effects of these failure modes without any mitigation means (e.g. FMECA [Vil92] – Failure Modes Effects and Criticality Analysis). Implicitly again, the classification of the impact of each failure mode on the software must be assessed. We could have seen the identification as a sub-goal and split in two sub-branches the justification as in RU3. The evidence on the right can also be achieved through safety analysis (e.g. MBSA [BVÅ+03] – Model-Based Safety Assessment). In this case, the

mitigation means are considered in the analysis: they should ensure the safety objectives with respect to the failure conditions and they should enforce that the effects of the identified failure modes are handled in a *fail-safe* manner (*i.e.* contained within the equipment on which the multi-core processor is installed). As identified by [Pro14], performing these safety analyses with classic techniques like FMEAs, Fault-Trees or Markov chains raises the following challenges:

- since multi-core are highly hierarchical and complex systems, their modelling with classic formalisms will be cumbersome and error-prone.
- 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, pragmatic model-based safety assessment as proposed in [BBB+18b, DPC19] can be performed with the first functional level of the multi-core (like the one depicted in Figure 1) and abstract failures modes based on the services provided by components.

Example 3 (Safety analysis) *Let us consider Example , where a safety-critical application a_1 is running on the first ARM core, which only accesses the MSMC SRAM memory through the MSMC controller. In addition, a second safety critical application a_2 executes on a C66* DSP *which only accesses the DDR memory through the MSMC controller. This configuration is compliant with Figure 4 and acceptable with respect to RU3 (see Example 3).*

A safety analysis then consists in modelling the possible hardware failures of the internal components, and in deriving their effects over applications. For instance, if the MPPAX is erroneous, then a_2 cannot access its data. Even worse, transactions may erroneously be redirected to the MSMC SRAM (see Figure 2), compromise the SRAM integrity, and violate the interference requirements. Such a situation is very jeopardizing and appropriate means of mitigation must be developed for this scenario so that it is handled in a fail-safe manner.

4 Service-based modelling approach

PHYLOG proposes concrete solutions for an applicant. However, interference and safety analyses are made quite independently from each other. So far, there is no possible reuse or factorization of knowledge between two platforms, nor is there a specific approach depending on the multi-core architecture, despite the fact that they may share common features (as the KEYSTONE and the TMS [Tex13b] do).

We claim that an architecture can be abstracted as the services it offers, and that such an approach will lead to a better understanding and handling of the MCP-CRI / CAST-32A requirements. The notion of service, as presented in [TMD09, FL13], is widely used in the domain of software architectures. It has been particularly developed in the context of web services, the main idea being to abstract what a software component can provide and request (as a service).

4.1 What is a service?

We propose to adapt the service-based approach to model the interactions between software and complex hardware components. The main difference from the previous software approach is that hardware service bindings are statically enforced by physical connections. Note that, in this new modelling, CPUs (as ARM cores and C66x DSP) are considered as platform components that provide services to the software applications hosted on them. From this new point of view, applications are considered as *initiator* whereas CPUs are transporter components.

Example 4 *For instance, let us consider again the* KEYSTONE *with the configuration described in Example 3. Application a_2 may need to read a data stored in the DDR memory. This will be expressed in terms of services as a* load *service call. When a_2 makes a* load *service call, the transaction is propagated through the internal components to reach the DDR and each of these locally provide some services. Figure 6 shows an extract of the service-oriented* KEYSTONE *architecture. The* load *from the* DSP *communicates with the* load *service provided by MPPAX.*

We first have to define the set of services offered by a platform. We have identified six basic services (extended from those of [MJB+17]):

- *execute*: execution of a piece of software on some core;

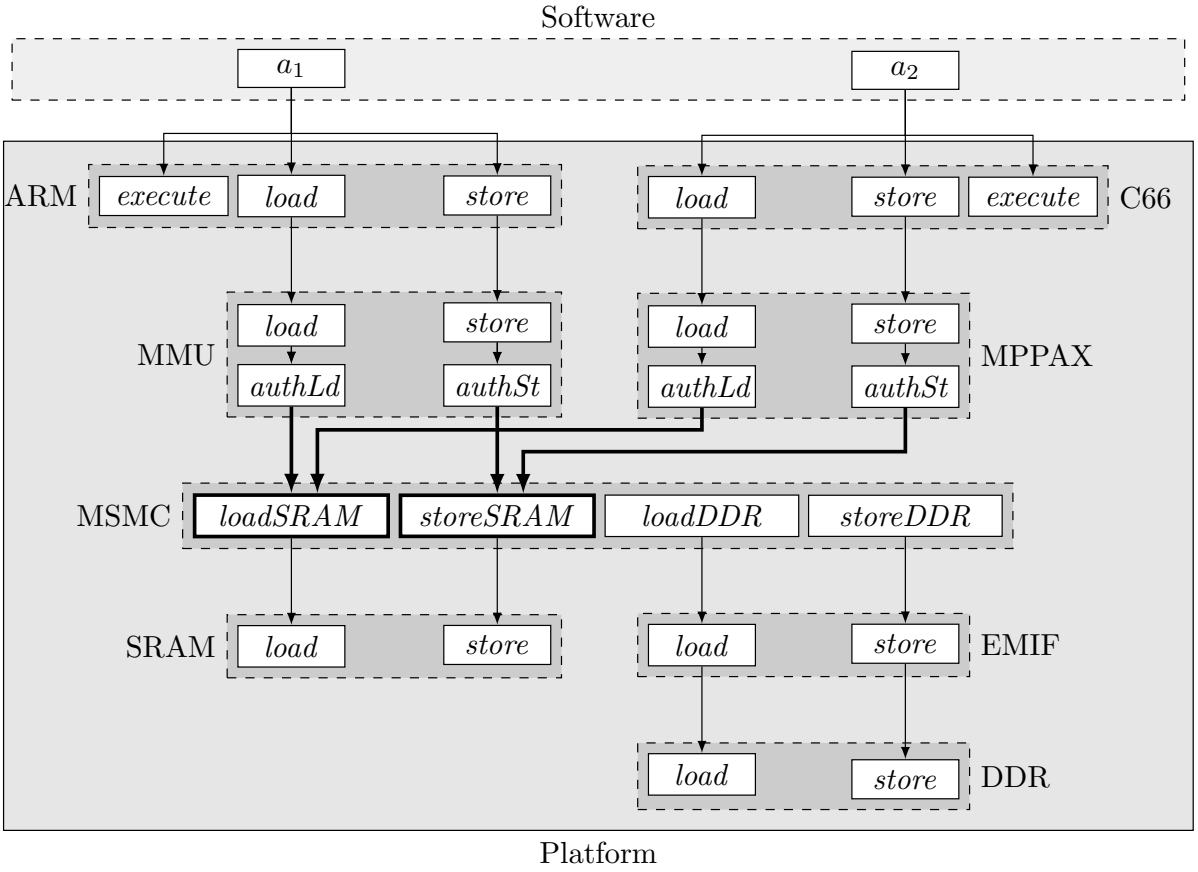

Figure 6: KEYSTONE service-oriented architecture

- *load*: retrieval of some data from a given target t by an initiator i. In particular, a *load* generates a transaction such that there exists a path from i to t.
- *store*: writing of some data to a given target t by an initiator i. In particular, a *write* generates a transaction such that there exists a path from i to t.
- *copy*: copy of some data from one memory area t_1 to another t_2 by a non-smart initiator i. In particular, a *copy* generates two transactions somehow in parallel such that there exists two paths from i to t_1 and from i to t_2.
- *time*: providing of a real-time reference, i.e. a clock which can be reached and read. If n applications $a_1, \ldots, a_n$ are hosted on the same component which provides a *time* service, then $a_1, \ldots, a_n$ have the same time reference.
- *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.

Example 5 (Basic services) *In the figure 6, a_1 executes on an* ARM. *When a_1 reads a data from the SRAM, it calls the* load *service of ARM, itself using the* load *service of MMU, which calls the* loadSRAM *of the MSMC, which calls the* load *of the SRAM.*

4.2 Why a service-based approach?

Modelling a complex hardware platform as a set of components offering and requesting services helps the applicant to identify the dependencies between the services provided by internal components. Such an approach eases and clarifies the two types of analysis requested by the MCP-CRI / CAST-32A. The idea is to shift the interference and safety analyses to service analyses, allowing the safety & interference analyses to be performed using the common service-oriented model.

4.3 Service-based interference analysis

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. Let us consider Figure 6 as a baseline for a more detailed explanation: the arrows between services denote *propagations* of transactions from one service to another. 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 service targeted by multiple arrows in the service-oriented platform model.

Example 6 *Back to Figure 6. There are two services targeted by two arrows:* loadSRAM *and* storeSRAM. *Not represented in the figure is the arbiter inside the MSMC, which routes the requests either for the SRAM or the DDR. Since the hardware analysis has demonstrated (see example 3) that arbiter's temporal effect to be negligible, the conflicts are managed at the SRAM and EMIF levels. Thus,* loadSRAM *and* storeSRAM *services are interference channels between applications* a_1 *and* a_2. *The solution proposed in Example 3 consisted in changing the memory allocation used by* a_2 *(running on the C66x DSP), and in configuring the authorization service hosted by MPPAX to prevent any transaction from* a_2 *to the SRAM memory. This solution is depicted Figure 7. Although the MSMC component is crossed by multiple transactions, there is no more services targeted by more than one arrow. Thus, this solution does not contain any unacceptable interference channels.*

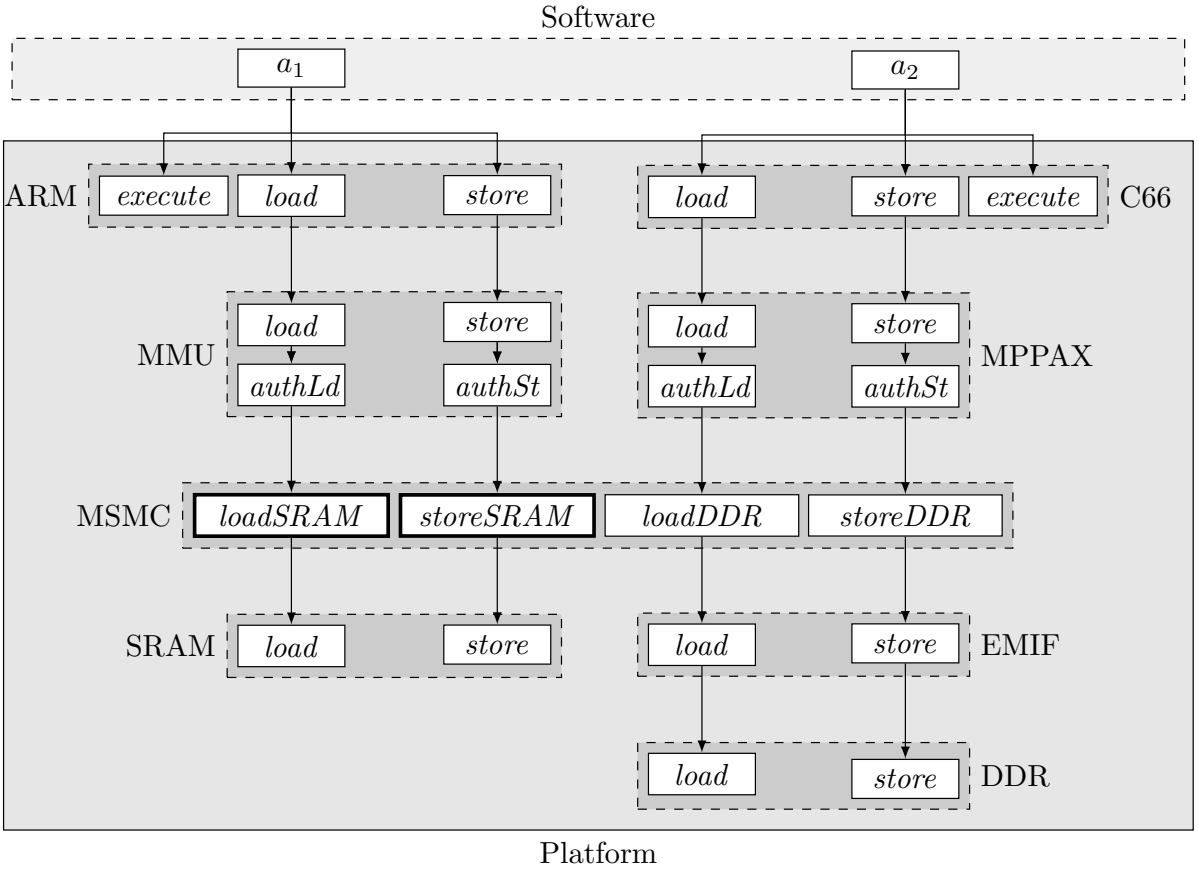

Figure 7: Figure 4 service-oriented modeling

4.4 Service-based safety analysis

Example 3 presented the safety analysis in a way that is agnostic of the service approach. A safety analysis applied to a service-based modelling identifies the physical failures leading to a service failure

affecting critical software (*i.e.* whose failure has an unacceptable safety effect). With the new approach, the arrows between services are considered as *propagations* or *dependencies* (*i.e.* service s_1 requests service s_2) to be provided properly. In addition, the relationship between the components failure modes and their effect on services must be specified.

Example 7 *Let us consider the same failure mode as in Example 3, where the MPPAX fail-erroneous mode leads to an erroneous* load *service used by the C66 and, ultimately, by the application a_2. Hence, if a_2 is a safety-critical application and requests the* load *service, an analysis of service dependencies discloses a single point of failure, therefore the applicant has to design appropriate mechanisms to mitigate the effects of this failure.*

4.5 Service-based platform definition

We can now formally define the concept of a service-oriented platform model:

Definition 1 (Platform model) *A platform is defined by $P = (\mathcal{C}, \rightarrow_{HW}, privilege, \rightarrow_{BS}, provides, request)$ where*

- $\mathcal{C} = I \cup T \cup O$ *with I the set of initiators, T the set of targets, and O the set of transporter components. All those sets are disjoint.*
- $\rightarrow_{HW} \subseteq \mathcal{C}^2$ *are the hardware connections between components.*
- $\text{privilege} : T \mapsto \mathbb{N}$ *is a function which associates a privilege level to each target. A target t can only be accessed by requests issued from an initiator with a privilege level higher or equal to* privilege(t).
- $\text{provides} : \mathcal{C} \mapsto 2^{\mathcal{BS}}$ *is the function which associates each component with the set of services it can provide. $\mathcal{BS}$ is the set of basic services provided by the components. Each element of $\mathcal{BS}$ is an instance of one of the six basic services defined previously.*
- $\rightarrow_S \subseteq \mathcal{BS}^2$ *are the connections between services provides by the components.*
- $request : I \mapsto 2^{2^{\mathcal{BS}} \times \mathbb{N}}$ *is the function which associates each initiator component with the set of transactions it can issue; a transaction being a path in the service-based architecture, at a privilege level n.*

Example 8 (Platform model) *For the sake of readability, let us introduce the following notations: ld stands for load, st stands for store, ex stands for execute, and at stands for authorize service. The service-oriented platform of the figure 7 can be modelled as:*

- $I = \{a_1, a_2\}$, $T = \{\text{SRAM}, \text{DDR}\}$
- $O = \{\text{ARM}, \text{C66}, \text{MPPAX}, \text{MMU}, \text{MSMC}, \text{EMIF}\}$
- $\rightarrow_{HW} = \left\{ \begin{array}{ll} (\text{ARM}, \text{MMU}), & (\text{MMU}, \text{MSMC}), \\ (\text{MPPAX}, \text{MSMC}), & (\text{MSMC}, \text{EMIF}), \\ (\text{MSMC}, \text{SRAM}), & (\text{EMIF}, \text{DDR}), \\ (\text{C66}, \text{MPPAX}) & \end{array} \right\}$
- $\forall c \in \mathcal{C}, privilege(c) = 1$
- $\mathcal{BS} = \left\{ \begin{array}{ll} ld_{\text{SRAM}}, & st_{\text{SRAM}}, \\ ld_{\text{DDR}}, & st_{\text{DDR}}, \\ ld_{\text{EMIF}}, & st_{\text{EMIF}}, \\ ld_{\text{MMU}}, & st_{\text{MMU}}, \\ ld_{\text{MPPAX}}, & st_{\text{MPPAX}}, \\ ld_{\text{C66}}, & st_{\text{C66}}, \\ ld_{\text{ARM}}, & st_{\text{ARM}}, \\ ex_{\text{ARM}}, & ex_{\text{C66}}, \\ ld\text{SRAM}_{\text{MSMC}}, & st\text{SRAM}_{\text{MSMC}}, \\ ld\text{DDR}_{\text{MSMC}}, & st\text{DDR}_{\text{MSMC}}, \\ atLd_{\text{MPPAX}}, & atSt_{\text{MPPAX}}, \\ atLd_{\text{MMU}}, & atSt_{\text{MMU}}, \end{array} \right\}$
- $provides(c) = \{s_{c'} \in \mathcal{BS} | c = c'\}$

- $\rightarrow_{BS}= \left\{ \begin{array}{l} (ld_{\text{ARM}}, ld_{\text{MMU}}), \\ (ld_{\text{MMU}}, atLd_{\text{MMU}}), \\ (atLd_{\text{MMU}}, ld\text{SRAM}_{\text{MSMC}}), \\ (ld\text{SRAM}_{\text{MSMC}}, ld_{\text{SRAM}}), \\ (st_{\text{ARM}}, st_{\text{MMU}}), \\ (st_{\text{MMU}}, atSt_{\text{MMU}}), \\ (atSt_{\text{MMU}}, st\text{SRAM}_{\text{MSMC}}), \\ (st\text{SRAM}_{\text{MSMC}}, st_{\text{SRAM}}), \\ (ld_{\text{C66}}, ld_{\text{MPPAX}}), \\ (ld_{\text{MPPAX}}, atLd_{\text{MPPAX}}), \\ (atLd_{\text{MPPAX}}, ld\text{DDR}_{\text{MSMC}}), \\ (ld\text{DDR}_{\text{MSMC}}, ld_{\text{EMIF}}), \\ (ld_{\text{EMIF}}, ld_{\text{DDR}}), \\ (st_{\text{C66}}, st_{\text{MPPAX}}), \\ (st_{\text{MPPAX}}, atSt_{\text{MPPAX}}), \\ (atSt_{\text{MPPAX}}, st\text{DDR}_{\text{MSMC}}), \\ (st\text{DDR}_{\text{MSMC}}, st_{\text{EMIF}}), \\ (st_{\text{EMIF}}, st_{\text{DDR}}) \end{array} \right\}$
- $request(i) = \left\{ \begin{array}{ll} \{(\{ld_{\text{ARM}}, ld_{\text{MMU}}, & \textit{if } i = a_1 \\ atLd_{\text{MMU}}, ld\text{SRAM}_{\text{MSMC}}, & \\ ld_{\text{SRAM}}\}, 1)\} & \\ \{(\{ld_{\text{C66}}, ld_{\text{MPPAX}}, & \textit{otherwise} \\ atLd_{\text{MPPAX}}, ld\text{DDR}_{\text{MSMC}}, & \\ ld_{\text{EMIF}}, ld_{\text{DDR}}\}, 1)\} & \end{array} \right.$

5 Platform properties

Platforms come with a series of features and properties that can be related to the service modeling point of view. The expected benefit of exhibiting properties is to take advantage of such a knowledge to simplify the safety and interference analyses. In the following, we discuss four platform properties: (a) full synchronism, (b) weak synchronism, (c) hardware segregation, (d) partitioning.

5.1 Full Synchronism

Definition 2 *Let us consider a platform P and a set of components $C = \{c_1, \ldots, c_n\}$ of P such that each c_i provides a time reference service, noted $time_{c_i}$. The set C is* fully synchronized *if and only if $\forall i, j \in \{1, \ldots n\}^2, time_{c_i} = time_{c_j}$, i.e., all components in C have exactly the same time reference (same clock with same offset).*

Platforms 1 *Examples of platforms offering group of full synchronized components include the Keystone TCI6630K2L [Tex13a] from Texas Instruments depicted Figure 1. The local clock of each Cortex-A15 core is sourced from a unique* ARM *PLL Controller shared by the four cores. The signal clocks of the four cores are then strictly identical, meaning that the four* ARM *cores form a fully synchronized group.*

Benefits 1 (Interference analysis) *Full synchronism helps implement TDMA (time division multiple access) execution models, enforcing a unique access to any shared resource at any time, such as the execution models of [PBB+11, BCNP12, GJlR+15]. This can be used to implement time partitioning mechanisms, as proposed in Example 3 to prevent simultaneous access from the* ARM *cores to the shared SRAM. Such reasoning can be generalized with a new pattern. In this new pattern, described in Figure 8, the goal "i* is avoided*" is the RU3 sub-goal: "i* is mitigated*". The sub-goal can be concluded because there is mathematical proof that S ensures that i is avoided under the full synchronism hypothesis and the fact that the system is fully synchronous.*

Benefits 2 (Safety analysis) *From a safety point of view, the failure propagation model assuming synchronization service availability should be simpler than a non-synchronized system, since some simultaneous access like failures will be avoided. As a result, the full synchronism property can be used to split the safety analysis as follows: 1. identify the hardware failures leading to a loss of the property; 2. assess the*

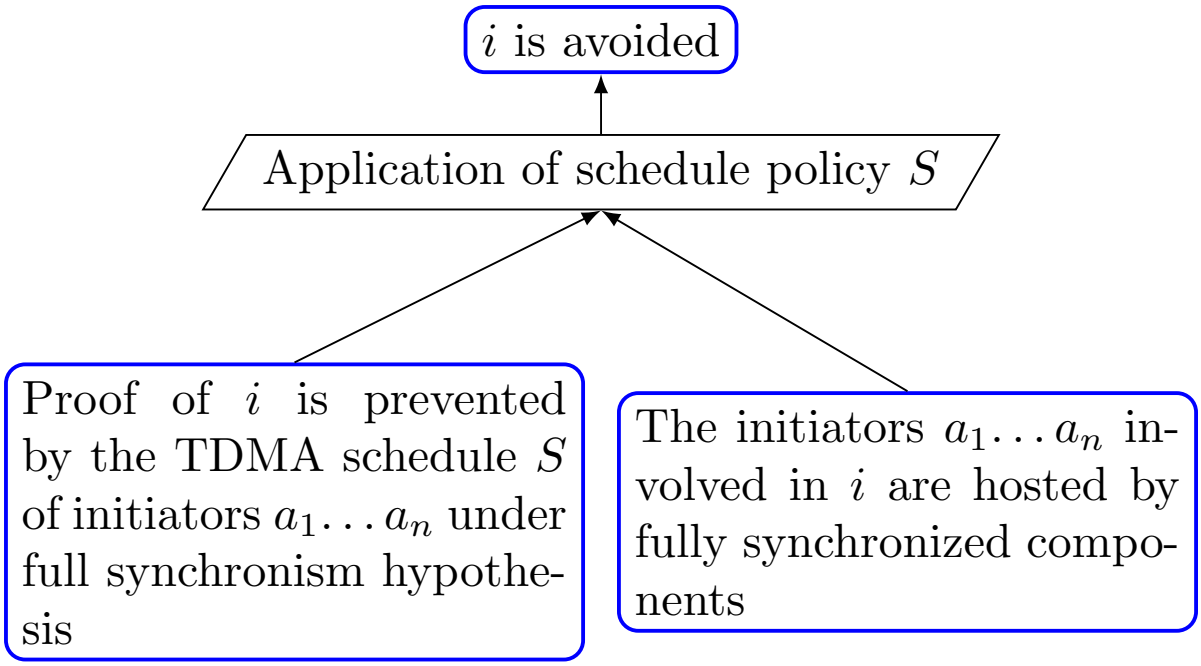

Figure 8: RU3 mitigation sub-goal with full synchronism

safety issues caused by a loss of the property; 3. perform the safety assessment assuming that the property is met.

5.2 Weak Synchronism

In some architectures, only a weaker form of synchronism can be achieved.

Definition 3 *Let us consider a platform P and a set of components $C = \{c_1, \dots, c_n\}$ of P, such that each c_i provides a time reference service, noted $time_{c_i}$. The set C is* weakly synchronized *if and only if* $\forall i, j \in \{1, \dots n\}^2, \frac{d\ time_{c_i}}{dt} = \frac{d\ time_{c_j}}{dt}$*, i.e., the rates of the local clocks of each component c_i are identical whereas their offsets may be different.*

Platforms 2 *Again, examples of platforms offering groups of weakly synchronized components include the Keystone TCI6630K2L. The clock of each C66x core is locally sourced from its own PLL controller. All the PLL controllers of the C66x CorePac can be configured in such a way that all the clock signals have the same rate. However, the offsets between the clocks cannot be set. They are unknown and only non deterministically fixed at boot time.*

Kalray MPPA [Cor12] is another example of weak synchronous platform. An MPPA Processor is composed of 16 clusters of 16 cores. Every core of each cluster has access to a local 64-bit Time Stamp Counter (TSC) driven by a single hardware signal. However, because of the propagation delays of this hardware signal through the 256 cores, all the TSC are mesosynchronous, i.e., they share the same rate but might have constant unknown offsets depending on the length of the path followed by the signal to reach each core.

Benefits 3 (Interference analysis) *Even in this case, it remains possible to implement a TDMA execution model avoiding interferences, provided that the TDMA schedule is tolerant to all possible offset values. As previously, this reasoning can be generalized by a new pattern depicted in Figure 9.*

5.3 Hardware segregation

A third interesting property is hardware segregation. It occurs when a platform supports the decomposition and isolation of several parts at hardware level.

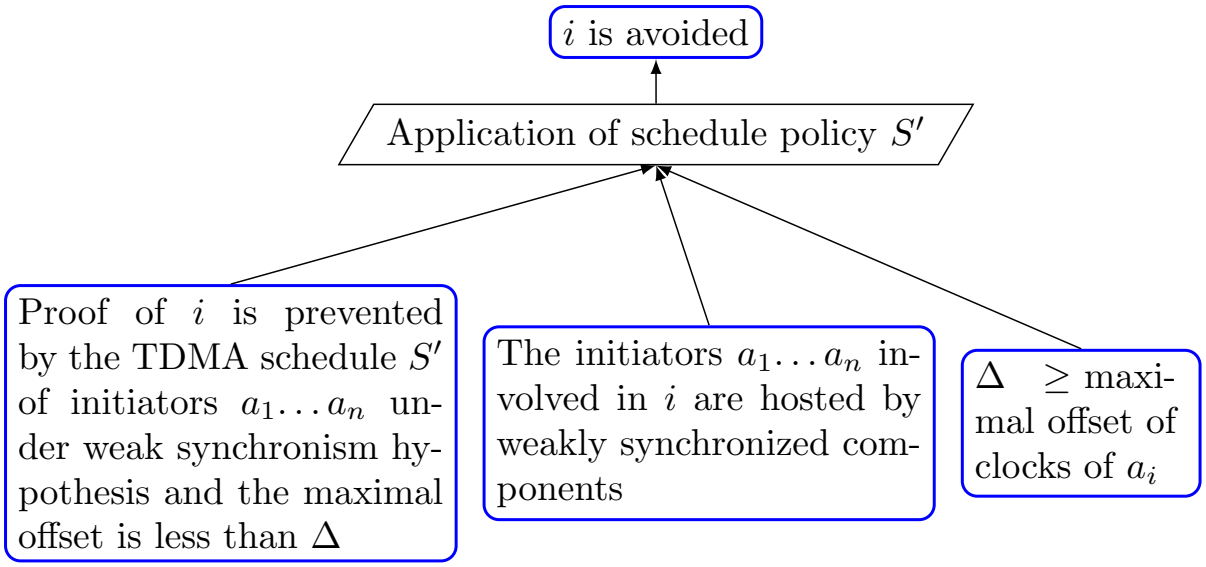

Figure 9: RU3 mitigation sub-goal with weak synchronism

Definition 4 *Let us consider a platform* $P = (\mathcal{C}, \rightarrow_{HW}, privilege, \mathcal{BS}, \rightarrow_{BS}, provides, request)$. *$P$ is segregated if and only if there exist non empty sub-sets $C_1 \dots C_n$ of $\mathcal{C}$ such that $C_1 \dots C_n$ is a partition of $\mathcal{C}$ and for any i, any initiator from C_i cannot reach (with* load *or* store *service) any component in another part C_j $(j \neq i)$. Formally, P is segregated into $C_1 \dots C_n$ iff:*
(1) $C_1 \dots C_n$ is a partition of $\mathcal{C}$,
(2) $\forall i = 1 \dots n, \forall a : initiator \in C_i, \forall t \notin C_i, a \not\rightarrow^*_{HW} t$.

Platforms 3 *An example of platform which can be segregated at hardware level is the Kalray MPPA Bostan [Cor12]. This processor is composed of clusters communicating through a central network on chip (NoC). Each router of the NoC can be configured at boot time in such a way that a specified set of input-output links (of the router) are disabled. It makes it possible to isolate non-interfering parts inside the processor.*

Benefits 4 (Interference analysis) *The expected benefit for interference analysis is straightforward. When a platform meets the hardware segregation property, if two initiators a_1 and a_2 belong to two isolated parts, then a_1 and a_2 cannot interference. Firstly, this allows to reduce the number of interferences to analyze. Secondly, this makes the separation of analyses possible, as shown in figure 10.*

Benefits 5 (Safety analysis) *If the segregation mechanisms is trusted (no probable failures considered), then the segregated resource can be modeled and assessed as two independent systems. This segregation eases both the modeling and the assessment.*

5.4 Partitioning

When hardware segregation is not possible, some platforms support the decomposition and isolation of several parts by configuration and/or software.

Definition 5 *Let us consider a platform* $P = (\mathcal{C}, \rightarrow_{HW}, privilege, \mathcal{BS}, \rightarrow_{BS}, provides, request)$. *$P$ is partitioned if and only if there exist non empty sub-sets of services $S_1 \dots S_n$ of $\mathcal{BS}$ such that $S_1 \dots S_n$ is a partition of $\mathcal{BS}$ and for any i, any initiator able to request a service in S_i cannot request any service in another part S_j $(j \neq i)$. Formally, P is partitioned into $S_1 \dots S_n$ iff:*
(1) $S_1 \dots S_n$ is a partition of $\mathcal{BS}$,
(2) $\forall a : initiator, request(a) = \cup\{A_i, p_i\}_i$ *with* $A_i \subseteq \mathcal{BS}$ *then* $\exists j,\ A_i \subseteq S_j$.

Platforms 4 *Many platforms offer partitioning capabilities. For instance,* KEYSTONE *allows partitioning of cache in SRAM areas. At software level, hypervisor systems such as* XTRATUM *[MRC+09] offers time and space partitioning.*

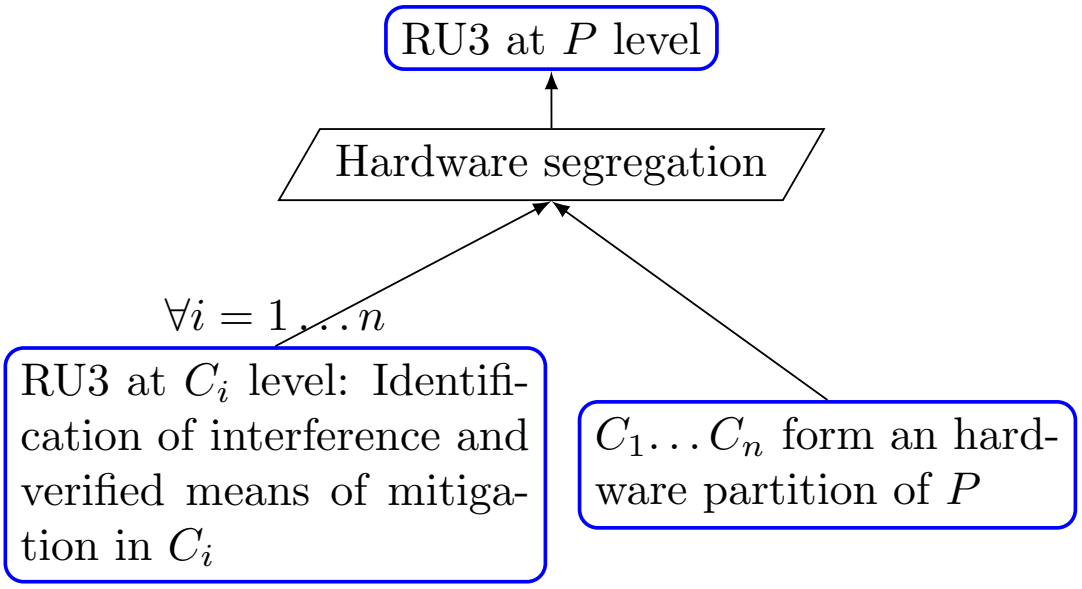

Figure 10: RU3 top level with HW segregation

Benefits 6 (Interference analysis) *The expected benefit of the partitioning property is similar to the one of the segregation property. Let us consider a platform partitioned into two set of services S_1 and S_2. No connections are allowed between services of S_1 and S_2. Hence, by definition, two initiators, respectively a_1 and a_2, that can only request services, respectively from S_1 (for a_1) and from S_2 (for a_2), cannot interfere. Similarly to the segregation property, the benefit is double: firstly, partitioning leads to a reduction in the number of interferences to analyze. Secondly, it makes it possible to do separated analyses (one for each partition).*

Benefits 7 (Safety analysis) *The partitioning is often paramount for failure containment argumentation. Hence, the partitioning service capabilities, limitations and underlying resource must be clearly identified. The analyst can benefit from the partitioning service to split the safety assessment (similarly to the synchronization service) as follows: 1. identify the hardware failures leading to a failure of the partitioning service; 2. assess the safety issues of the failures of the partitioning service; 3. perform the safety assessment assuming the partitioning service is available.*

6 Conclusion and next work

In this paper, we have presented an improved approach to automate interference and safety analyses. The main idea is to abstract any platform as the services it offers. We have thus adapted the service-based approach to model the interactions between software and complex hardware components. The benefits are numerous and we have illustrated two of them: it provides a common model to support both safety and interference analyses; it allows the definition of architecture properties in order to simply the analyses and the justification patterns.

In the future, we will improve the current approach in several ways. First, we will extend the set of basic services to cover usual features offered by multi-core (e.g. interrupt). Second, we will study any other platform properties (e.g. symmetry or modularity). Third, we will show how to reuse the analyses and patterns for platforms with common features.

We will also define formal semantics for the platform model and provide transformations from the core platform model to the interference model and to the safety model, in order to allow analyses on dedicated models.

Acknowledgements

The authors would like to thank Ghilaine Martinez for her ideas and suggestions to improve platform modelling analysis.

References

[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.

[BBD+16] Pierre Bieber, Frédéric 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.

[BCNP12] Frédéric Boniol, Hugues Cassé, Eric Noulard, and Claire Pagetti. Deterministic execution model on cots hardware. In *Proceedings of the 25th International Conference on Architecture of Computing Systems (ARCS'12)*, pages 98–110, 2012.

[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.

[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.

[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.

[DPC19] Kevin Delmas, Claire Pagetti, and Philippe Cuenot. Multi-core processors: Stepping inside the box. In *under submission*, 2019.

[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.

[FL13] José Luiz Fiadeiro and Antónia Lopes. An interface theory for service-oriented design. *Theoretical Computer Science*, 503:1–30, 2013.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[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.

[Hol15] C. Michael Holloway. Explicate'78: Uncovering the implicit assurance case in do-178c. In *23rd Safety-Critical Systems Club (SCSC) Annual Symposium*, February 2015.

[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.

[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.

[MRC+09] Miguel Masmano, Ismael Ripoll, Alfons Crespo, Jean-Jacques Metge, and Paul Arberet. Xtratum: An open source hypervisor for TSP embedded systems in aerospace. In *DASIA 2009. DAta Systems In Aerospace.*, May. Istanbul 2009.

[PBB+11] Rodolfo Pellizzoni, Emiliano Betti, Stanley Bak, Gang Yao, John Criswell, Marco Caccamo, and Russell Kegley. A predictable execution model for cots-based embedded systems. In *17th IEEE Real-Time and Embedded Technology and Applications Symposium RTAS 2011*, pages 269–279, 2011.

[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.

[RTC11] RTCA, Inc. DO-333 - Formal Methods Supplement to DO-178C and DO-278A, 2011.

[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.

[Tex13a] Texas Instruments. TCI6630K2L Multicore DSP+ARM KeyStone II System-on-Chip. Technical Report SPRS893E, Texas Instruments Incorporated, 2013.

[Tex13b] Texas Instruments. TMS320c6678 Multicore fixed and floating-point digital signal processor. Technical Report SPRS691D, Texas Instruments Incorporated, 2013.

[TMD09] Richard N. Taylor, Nenad Medvidovic, and Eric Dashofy. *Software Architecture: Foundations, Theory, and Practice.* Wiley Publishing, 2009.

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

[WKRP05] Ingomar Wenzel, Raimund Kirner, Bernhard Rieder, and Peter Puschner. Measurement-based worst-case execution time analysis. In *3th Workshop on Software Technologies for Future Embedded and Ubiquitous Systems (SEUS'05)*, 2005.