MIMOSA: Towards a model driven certification process

Pierre Bieber, Frédéric Boniol, Guy Durrieu, Olivier Poitou, Thomas Polacsek, Virginie Wiels, Ghilaine Martinez

MIMOSA: Towards a model driven certification process

Pierre Bieber, Frédéric Boniol, Guy Durrieu,
Olivier Poitou, Thomas Polacsek, Virginie Wiels
ONERA, Département Traitement de l'Information et Modélisation
2, avenue Edouard Belin BP74025, 31055 TOULOUSE Cedex 4
{firstname.lastname}@onera.fr
Ghilaine Martinez, DGA TA, firstname.lastname@intradef.gouv.fr

27-29 JANUARY 2016

Abstract

A certification process usually consists in analyzing, in a restricted amount of time a, potentially very large, set of documents that are intended to convince the auditor that the documented system fulfills all its requirements. The MIMOSA Project presented in this paper introduces a model driven certification process based on the key concepts of argumentation step, patterns and composition. The aim is: at first, to structure the documentation provided as evidences of the good properties of the system, and then to check this structure against identified argumentation patterns that will help identifying lacks or misuse of elements. Argumentation step and composition principles as well as a set of patterns for arguing about real-time properties are given along with their expression in a prototype tool, that offers to describe the architecture, requirements and argumentation in a common language and then offers to compute some basic checks on the argumentation structure.

Keywords : Certification, Safety, Real-Time, Model-based System Engineering, Argumentation

1 Introduction

The MIMOSA[1] project aims at building a frame of reference for the certification of military embedded architectures. The goal of this frame is not to design and develop architectures, but rather to formalize requirements for this kind of architectures and place in front of them acceptable means of compliance, building a coherent argumentation. This frame of reference may be used by DGA to assess architectures proposed by industrial companies with respect to certification standards [5].

The frame of reference includes models of the fundamental concepts of a modular architecture, models of the requirements attached to these architectures and a model of the argumentation of the compliance of the architecture to the requirements. The frame includes different levels of description from functional to hardware; different facets such as Architecture, Safety and Real-time and different concerns Architecture documentation, requirements and argumentation.

Section 2 presents the modeling approach, language and tool. Section 3 describes the argumentation modeling principles, section 4 gives some examples and section 5 concludes.

2 Language Overview

The MIMOSA framework is organised as a three dimensional grid having as dimensions:

- the layer – or level of detail –(function, software, topology, hardware),

[1] MIMOSA stands for Means of engIneering for MOdelling and analysis of modular embedded aeronautic Systems and Architectures

- the concern (architecture, requirement, argumentation) and
- the facet (general, safety, real-time).

Facets are a way to focus on domain specific considerations: properties and associated elements of description. For example, the realtime facet extends the general one by adding:

- specific concepts or attributes to describe an architecture from the realtime aspect such as best and worst-case execution times of each application of the software level,
- specific realtime requirements such as "Worst Case Latency of function f should be bounded by time t" at the functional level and
- specific argumentation pattern for example to efficiently convince that an application worst case response-time is bounded.

Two specific facets are currently handled by the Mimosa framework: safety and real time.

2.1 MIMOSA language internals

The language in which all of those descriptions are expressed should make available to its users concepts such as a function, an application, a binary code and so on. We call this high level language the user language. To be able to fully support the user we need some reasoning capabilities on this user language. Obtaining such capabilities directly on a very rich language as the one we were about to define a very complicated task and leads to hard to maintain result. To obtain such capabilities we then started by defining a simpler language, a formal specification language, called WEIRD, that offers reasoning capabilities by being translatable to propositional logic. Then, the user language is described using WEIRD then inheriting its reasoning capabilities.

Elements of the WEIRD language are *entity*, *concept* and *relation* as well as constraint expressions (with quantifiers). WEIRD offers a typing system that offers to express that an *entity* e is an instance of a *concept* c, or that a *concept* sc is a sub concept of a *concept* c. Another central notion of WEIRD is the notion of *World* that allows modular modeling by constraining visibility and extent to which properties must apply. A World can contain Concepts, Entities, Relations and Constraints/properties to be satisfied. A World can derive from one or several other Worlds; in that case, it has access to all the elements and shall satisfy all the constraints of the World from which it derives.

The notion of World is used to model facets, layers and concerns. In particular one world is associated to each layer (function, software, topology, hardware) as well as to some combination of layers. Those "combination world" mainly contain allocation relations and constraints that take place between elements of different worlds. For example a constraint like "every partition (topological concept) is allocated on a unique CPIOM (hardware concept)" will be expressed in the topological hardware mapping world.

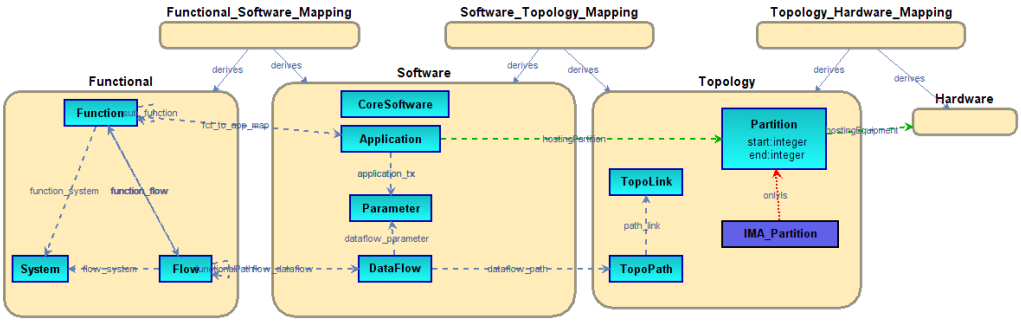

Figure 1: The architecture layers and mapping worlds[2]

This way the final "user language" – the language from which the final user will take elements to describe the system under analyses – is defined. Having this formal WEIRD language offers both the ability to make some reasoning and to ease evolution of the user language.

[2]The content of hardware world have been hidden since it would not have fit in the page width.

In our prototype, WEIRD is translated into propositional logic by applying rewriting rules (translation is rather direct from constraints expressions, user defined relations are kept as relations, all typing information becomes relations, world are used to restrict the domains on which quantifiers are expanded). This way, properties to check are valued by confronting them to knowledge expressed by the user[3].

2.2 Description of the system to analyze

When describing an instance to analyze, newly introduced worlds will derive from the layer, facet and concern worlds for which they are relevant. For example a world describing the functional safety requirements of the examined solution will derive at least from the three corresponding worlds Functional, Requirement, and Safety. This way:

- it will gain access to all the concepts, relations and entities they introduce and
- it will have to respect constraints introduced by all of them.

This derivation may be indirect in some cases: *instance functional description world* will derive from *Functional*, then the above described world may derive from the *instance functional description world* to gain access, at the same time, both to general rules from Functional (indirect derivation) and instance elements and rules from its instantiated version (direct derivation).

These derivations offer to reuse defined concepts and entities in the new worlds as in the example in Figure 2 of a Fire Control function (partial) description. The global CdT function, as well as its enabling applications are directly introduced without the need of redefining anything. In the same way 4 partitions are introduced in the TopoRef world by reusing the concept of IMA_Partition from the generic Topology one.

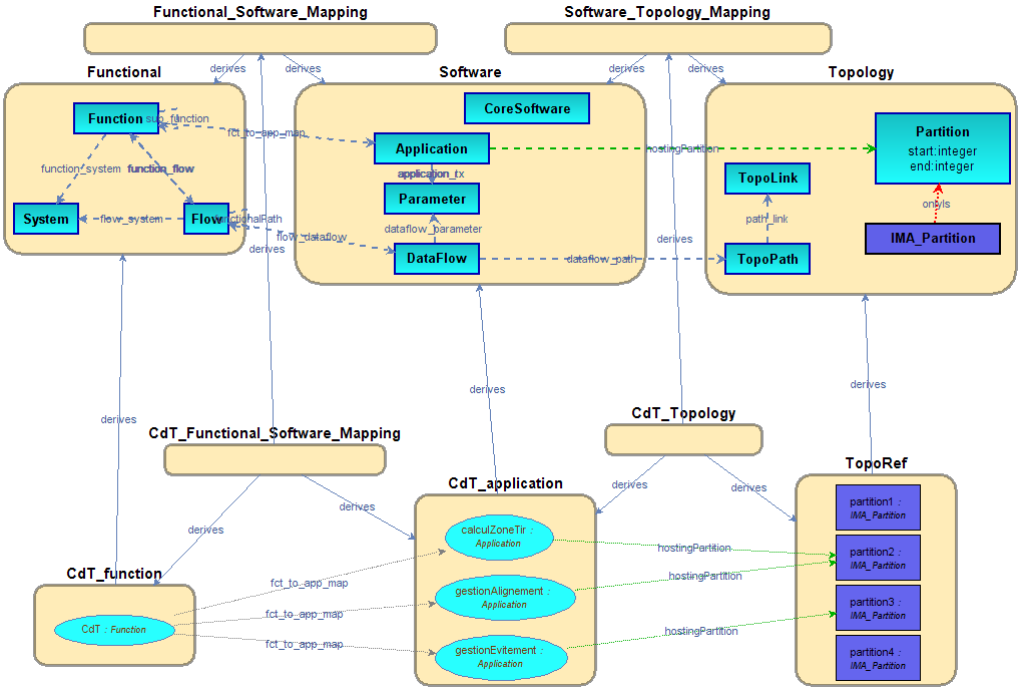

Figure 2: Worlds describing a product derives from generic ones

The same applies to constraints, the following requirement that every application is hosted by a partition is expressed in an intermediate SoftTopoRequirements world as this:

[3]Though this is not used in the context described in this article, the low level language is then compatible with SAT solvers tools that can be used to answer different questions.

```
world SoftTopoRequirements
      derives   Requirements,Software_Topology_Mapping   {
assert all_apps_allocated =
      forall entity a | a::Application =>
        exists entity p | p::Partition and hostingPartition[a]===p
}
```

This world is then derived by *CdTRequirements* that will gather all requirements applicable to CdT. There it is evaluated to *satisfied* by the prototype tool (this is why it is represented in green in the screenshot of Figure 3

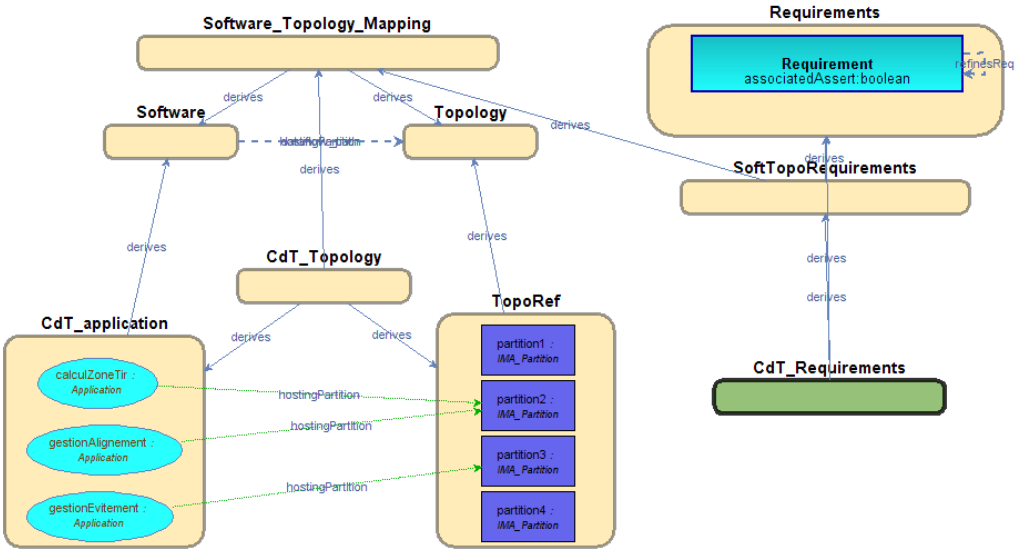

Figure 3: CdT requirements are gathered in the CdTRequirements world

3 Argumentation principles

When inquiring the certification of a system, the inquirer must provide a certification file. According to the Ministry of Defence, this should be "A reasoned, auditable argument created to support the contention that a defined system will satisfy the R&M requirements"[3]. The exact nature of the elements is not detailed but graphical representation or models are more and more part of this certification file. For the safety aspects, the Ministry of Defence even explicitly requires safety cases[4] from which our work is inspired. Anyway the provided elements tend to grow while the structure, in particular the precise intent of an element or another, is sometimes missing. At the same time, IMA (Integrated Modular Avionics) is becoming the standard while its certification offers some additional issues [6].

The goal of the MIMOSA Argumentation facet is to represent graphically the different means of compliance used to justify the satisfaction of the requirements. The Argumentation facet organizes the various elements (formal and informal) that contribute to the justification of requirements.

When coming to represent graphically argumentation, GSN [2] is a reference and MIMOSA argumentation strongly inspires from it. It also inspires from Toulmin works for underlying principles [7] and from existing work on assurance cases in the aeronautical domain [1]. In MIMOSA a generic argumentation step relies on the following concepts:

Claim the property to be justified (will often link to a requirement),

Evidence the facts that will be used to justify the claim (analysis results, test results, expert knowledge, bibliographical reference...),

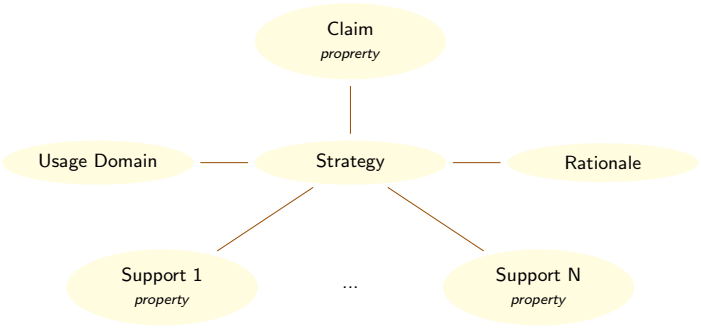

Figure 4: The generic argumentation step

Strategy combination of different evidences in order to justify a claim, is the model counterpart of "Mean of Compliance".

Usage Domain domain on which the method is usable

Rationale justifies the use of the method in this particular context

Argumentation patterns are then proposed as a partial instance of this generic step, specifying subconcepts, known entities, and their necessity status depending on the strategy or strategy family.

Argumentation patterns can be of different natures: generic (as "Using a software" that will make mandatory to explicit the usage domain and add a corresponding support to show its respect) or domain specific (like "Showing that the Worst Case Response Time of an application is bounded by a value").

When building an argumentation two mechanisms are then used:

- argumentation step chaining – one claim of a level becomes a support for the next one ;
- argumentation pattern composition – a given step inherits from more than one argumentation patterns.

In this last case, it has to exhibit the union of supports, usage domain and rationale of its "parents" (see Figure 5). To make sense strategy and claim of the parents should be compatible, for example "using a software to compute a WCRT bound" would inherit from the patterns "using a software" and "calculating a WCRT bound". See the example just below.

Note that the particular usage that is envisaged here protects us from what could have been an important issue: the supports consistency. In a general case, both mechanisms of combination of argument steps proposed just above would have, as an additional task, to prove that the resulting support set is consistent (in fact, this should also be explored for each elementary step). Each support is here considered valid (and the real situation consistent), that means that:

- an elementary argumentation steps having inconsistent support requirements is unusable (all its supports can not be fulfilled at the same time)
- when combining two steps for which supports have been provided, no inconsistency may appear.

The first item is one reason – the formal one – why argumentation step merging may not make sense [4]. For example if two argumentation patterns have been written to deal with two different situations, merging them is useless as no real case would match both situations and then it would be impossible to fulfill all the supports of the resulting pattern.

Eventually, a global constraint should be that every Requirement written –the system must have property P– has a corresponding argumentation claim –The system has the property P–

A certification argumentation is complete when every requirement has got a corresponding assert with proper method, supports and some times rationale and usage domain. This can

[4]The other reason why a argumentation step merging may not make sense is that it does not make sense from a business point of view.

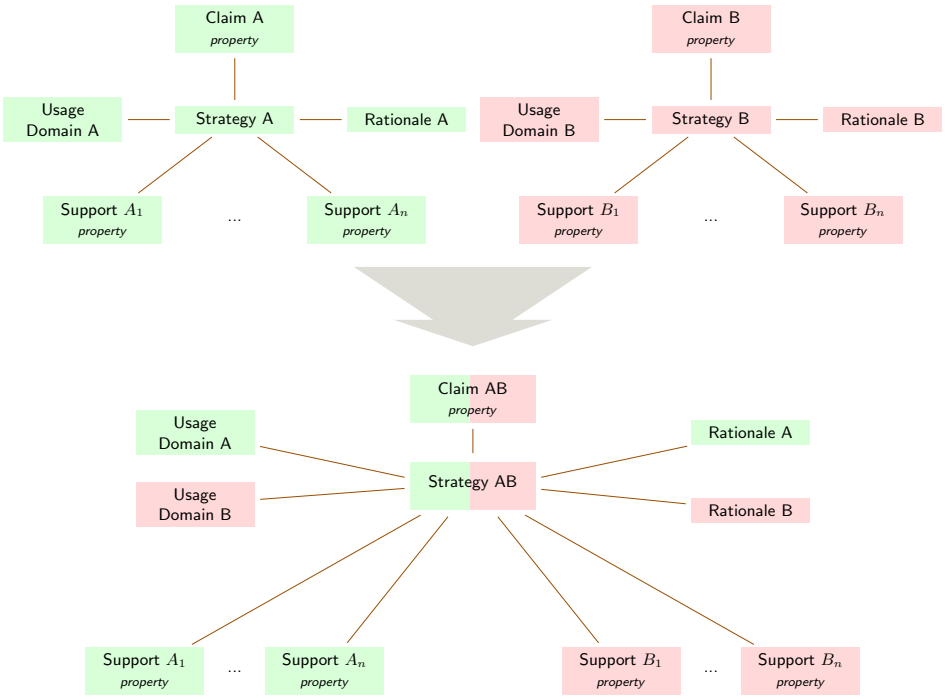

Figure 5: The argumentation step composition

be automatically checked by a tool then guiding the certification authority to missing asserts or argumentation steps.

4 Argumentation examples

"Using a software" is a generic pattern that mainly adds the constraint of exhibiting the *Usage Domain* (that is optional by default). This constraint activates another one that comes from the generic argumentation step telling that if a *Usage Domain* is attached to the *Strategy* then at least one support must show conformance to this usage domain.

The second argumentation pattern introduced is domain specific: "assessing an application maximum Worst Case Response Time". Such a claim needs that this WCRT is calculated, and relies necessarily on the computation of the Worst Case Execution Time of each involved applications in the partition hosting the evaluated one, and that the preemption policy and the period are known (links to architecture and realtime facet products will help precisely determine the relevant set of applications, as well as, if filled, automatically check preemption policy and period). A graphical representation of this pattern can be found in Figure 7. A constraint is added to ensure that, at least, WCET, preemption and period supports are provided if this strategy is used.

The third argumentation pattern is the composition of the two previous ones: "Using a software to assess an application maximum Worst Case Response Time" (see Figure 8). It then inherits from all the supports from the two previous ones, as well as the *Usage Domain* mandatory constraint coming from "Using a software".

5 Implementation and usage

Today, qualification/certification of new systems becomes more problematic due to several causes, the more invoked being the system complexity increase that is observed and a more practical one being the size of certification teams.

A less trivial reason is the evolution of the way system are developed: the development of a system is no more made from scratch with every subsystem developments made in consis-

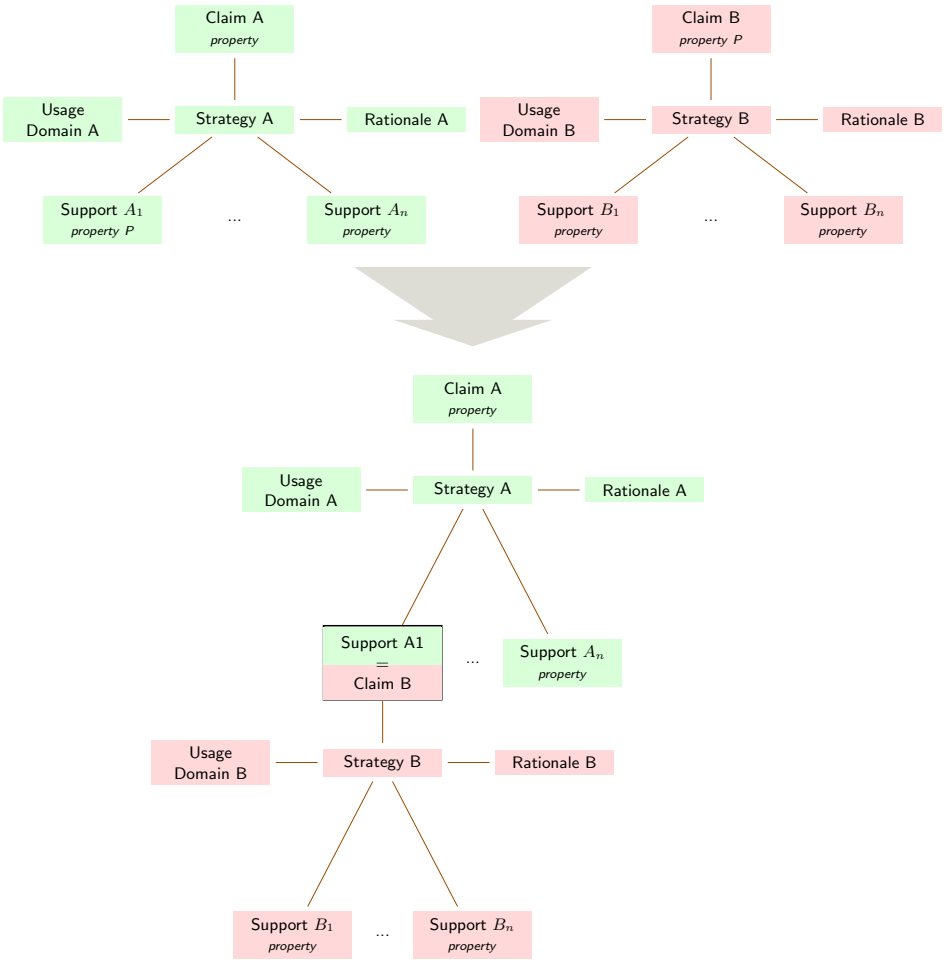

Figure 6: The argumentation step chaining

tency with the unique goal of integrating the system under development. On the opposite, a component based approach is more and more intensively used with a lot of reuse and "generic" components integration. This is strongly involved in the loss of structure of the certification file because part of the argumentation is then made either in another context (reuse case) or still at a general level (generic components case). Documentation provided to certification team is then not so organized, the same information may appear in many documents (strongly redundant documents may be provided to support the same kind of property but for different parts of the system due to the reuse of certification supports in another context and/or the pre-qualification steps). The certification file volume then grows up and, on the opposite of very redundant information, some precise information may become hard to find.

Formalizing the "structure" of the argumentation is the answer we promote in this work, by relying on the introduced argumentation pattern and combination mechanisms.

Apart from the formal principles enunciated in the previous section, the way one may use these argumentation patterns typically varies with the level of abstraction. Technical, low abstraction level patterns may benefit from formal inputs that a tool is able to manage, and provers or solvers can make automatic or semi-automatic reasoning from these argumentation steps. On the other extremity, high abstract level patterns are more likely to require human understanding and *supports* will often be links to heterogeneous documentation elements. In that last case, argumentation patterns must be seen as "check lists" and tooling will mainly guide the user to missing information, when some supports, rationale or usage domain are linked to no elements.

Classical envisage process of argumentation pattern is that the certification authority

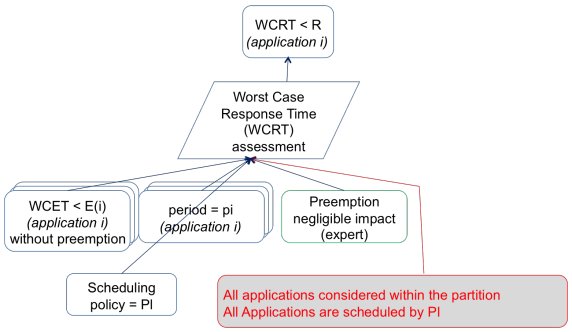

Figure 7: bounded WCRT argumentation pattern

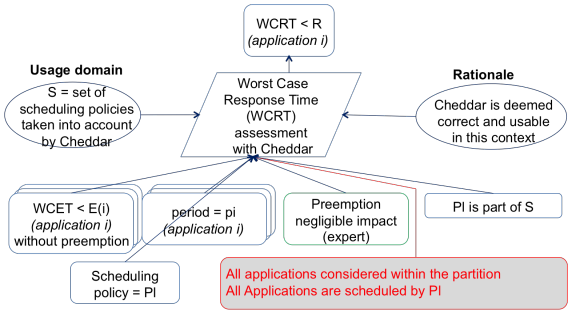

Figure 8: bounded WCRT by software computation pattern

shares validated patterns with industrial entity willing to obtain certification of a product. These patterns will act as guides for building a certification request for the industrial entity while supporting the analysis work of the certification entity, then being profitable for both. Capitalization is also made easier: certification authority builds a new pattern from analyzing a certification request that is not yet formalized this way, and then after an internal validation phase, add this new pattern as a new approved means of compliance. Industrial entity already having strong background in certification may also formalize their way of asserting a given property, and submit it to the certification authority as a new "standard" means of compliance –i.e. a new argumentation pattern– (this formalizes capitalization of the Certification Review Item process often used today when new approaches or technologies are to be evaluated).

6 Conclusion and perspectives

A Model based framework for certification process has been presented that offers to describe and analyze in a common language the system description, requirements that are put on it, and argumentation that is delivered to show the system conformance to its requirements.

Some argumentation patterns have been produced as well as a composition scheme that offers to build an instance of argumentation tree by mainly composing building blocks from rather domain specific ones to very generic ones.

A prototype has been developed to illustrate this approach, it has been tested on a case study containing two functions Fire control and Terrain following (then decomposed in several applications) allocated to a reference architecture with IMA capabilities ; and DGA TA envisages to use the approach on a real case study based on their certification activities. Nevertheless the exact way the methodology presented in this paper should take place is not yet completely defined, a shared tool between certification experts and candidate to certification would clearly be a good support but might not be realistic for tomorrow, an internal tool for the certification team that offers to build the argumentation based on the supports provided and some question/answer with the certification candidate may already help the certification team to organize, keep focused on the objective, and gain confidence in

the decision they will eventually make.

References

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

[2] 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.

[3] Ministry of Defence. *Defence Standard 00-42 Issue 2, Reliability and Maintainability (R&M) Assurance Guidance Part 3 R&M Case*, 2003.

[4] Ministry of Defence. *Defence Standard 00-56 Issue 4, Safety Management Requirements for Defence Systems Part 1 Requirements*, 2007.

[5] RTCA and EUROCAE. *DO-297/ED-124. Integrated modular avionics (IMA) development guidance and certification considerations.*, 2007.

[6] Jean-François Sicard, Ghilaine Martinez, and Florian Many. Specific Certification Issues Concerning IMA. In *Embedded Real-Time Software and Systems (ERTS2 2012)*, February 2012.

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