Extension and Utilization of a Design Framework to Model Integrated Modular Avionic Architecture

Yassine Ouhammou[1(✉)], Emmanuel Grolleau[1], and Pascal Richard[2]

[1] LIAS - Futuroscope, ENSMA, Poitiers, France
{ouhammoy,grolleau}@ensma.fr
[2] LIAS - Futuroscope, Université de Poitiers, Poitiers, France
pascal.richard@univ-poitiers.fr

Abstract. Embedded avionics systems often involve hard real-time constraints intended to ensure full system correctness. Avionics software development costs can be sharply impacted by wrong design choices made in the early stages of development, but often detected after implementation. By using temporal scheduling analysis, designers could detect infeasible real-time architectures, and prevent costly design mistakes. Recently, model-driven engineering paradigm facilitates the use of several analysis tools with standards modeling languages (e.g. SysML, AADL, UML-MARTE, etc.) to get a complete design cycle ranging from modeling up to verification and validation. However, only few model-based researches have studied the difficulty that designers face while seeking appropriate analysis tests which match their designs. MoSaRT (Modeling Oriented Scheduling Analysis of Real-Time systems) framework is one of these researches. It provides various facilities to help designers to model and analyze traditional real-time systems. In this paper, we propose the extension and the utilization of the MoSaRT framework to support avionics architectures. An analysis tool has also been developed and added in line with this kind of architectures. The proposed research is illustrated by discussing a real case study.

1 Introduction

On-board avionic systems are real-time systems which should meet strict temporal requirements. The architecture of these systems is based on the Integrated Modular Avionics (IMA) architecture instead of the traditional federated approach. Only few analysis tools have been dedicated to IMA systems. The integration of these tools with existing design languages, in order to get a model-based process combining modeling and analysis, is error prone. This integration requires a deep knowledge in both the modeling and the temporal analysis of real-time systems. In other words, the temporal analysis test chosen to validate a specific architecture has to be very appropriate to this latter. On the one hand, the test should be conservative because of the safety and the criticality of the system. On the other hand, the test should not be too pessimistic, otherwise

L. Bellatreche and Y. Manolopoulos (Eds.): MEDI 2015, LNCS 9344, pp. 16–27, 2015.
DOI: 10.1007/978-3-319-23781-7_2

the developed system can be costly in terms of equipments and wiring. A design framework called MoSaRT has been developed [16,17] to help designers to choose the most suitable temporal analysis tools referring to the system architecture conceived. Since the MoSaRT framework is based on model-driven engineering facilities, it is extensible and re-utilizable. This paper highlights the extension of the MoSaRT design language to support IMA architectures and shows how to enrich the framework by a prototype analysis tool helping designers to check the temporal requirements.

In order to make this paper self-contained, we introduce the main concepts related to IMA architectures and the MoSaRT framework. Then, the remainder of this paper is organized as follows. Section 2 discusses the specificities of IMA architectures. Section 3 introduces MoSaRT framework concepts and describes the way the framework can be used to ease the design of real-time architecture. Section 4 presents our proposition to extend the MoSaRT design language to support IMA architectures and the utilization of an analysis advisor which leads the temporal analysis process independently from the designer knowledge. A case study showing the significance of such model-based framework is also presented throughout Sect. 4. Finally, Sect. 5 drafts our conclusions and final remarks.

2 Partitioned Scheduling in Integrated Modular Avionics

Processor partitioning is a mean used in Integrated Modular Avionics (IMA) to insure freedom from interference between applications, which may be of different Design Assurance Level (DAL), but sharing the same physical resources. Freedom from interference, as defined in the standard ISO 26262 [7], insures that the failure of a piece of software will not create the failure of another piece of software. It also insures that the interference of two software independent parts is bounded. Moreover, an application can be studied independently from another application, even if they share the same processing resources.

The classic way to arbitrate the computing resources in a partitioned system is to use a hierarchical scheduler. In the IMA standard ARINC 653 [1], a two level scheduler is used: a low level cyclic scheduler is statically assigning the computing resources to the partitions, using a pre-defined table. Then, in each partition, a real-time operating system is assigning the processor to the tasks, using its own scheduling policy. On a timing behavior point of view, freedom from interference means that the interference of a partition on another one should be null or bounded. In most models, like it is the case in this paper, absence of interference is assumed. Nevertheless, cache related preemption delays may occur and create a bounded interference between one partition and other partitions of the same computing resource.

The seminal research on partitioned scheduling of Deng et al. [5] proposes two hierarchical levels. Later, Feng and Mok [6] proposed the resource partitioning model and schedulability analysis based on the supply bound function (sbf), which is an elegant mathematical way to extend non-hierarchical schedulability analysis to hierarchical scheduling. Shin and Lee [14] introduced the concepts of

temporal interface with the periodic resource model. For applications containing Fixed-Task Priority (FTP) schedulers, Kuo and Li [9] and Saewong et al. [13] proposed dedicated hierarchical schedulers. In the context of server based temporal partitioning, numerous techniques have been proposed. Lipari and Bini [11] used the abstraction of a periodic server in order to solve the problem of creating a partitioning such that the system is schedulable. Davis and Burns [4] proposed an upper bound on the response time of the tasks when the partitions are scheduled by an FTP. An exact worst-case response-time can also be obtained [2,3].

In the rest of this section, we present a scheduling model for hierarchical scheduling fitting with the ARINC 653 standard, and techniques allowing to compute the worst-case response times of the tasks.

2.1 Analysis Model

A (temporal) partition can be defined as a collection of time intervals defining statically the allocation of the CPU to a partition.

Definition: a resource partition is noted $\Pi = (\Gamma, P)$ where:

- $\Gamma = \{(S_1, E_1), (S_2, E_2), ..., (S_N, E_N)\}$ is a set time intervals, where S_i is the offset of a time interval allocated to the resource partition, related to the starting time of the period of the partition. E_i is the ending date of the time interval. The size of the i^{th} time interval is given by $E_i - S_i$.
- The size of the period of the resource partition is P.

A resource partition on a single CPU processing resource must obviously satisfy non-overlapping property: $0 \leq S_1 < E_1 \leq S_2 < E_2 ... \leq S_N < E_N \leq P$. For simplicity, we suppose that P is common to every partition of a CPU, corresponding to the MAF (MAjor Frame) in ARINC 653. A simple algebraic transformation using greatest common divisor and lest common multiple could be used to transform a classic ARINC 653 partition using MIF (MInor Frames) into a resource partition model.

An IMA application is a set of tasks, allocated to a resource partition on a CPU. In the sequel, we consider sets of sporadic independent tasks. Each application has its own fixed-task priority scheduler, which is scheduling its tasks in its resource partition.

Definition: an application j contains n_j sporadic tasks $\tau_i^j, i = 1..n_j$. A task τ_i^j is defined by (C_i^j, D_i^j, T_i^j) where:

- C_i^j is the task's Worst-Case Execution Time (WCET),
- D_i^j is the relative deadline of the task, i.e., the maximum allowed interval of time for the task to be completed after each of its activation,
- T_i^j is the period of the task, i.e., the minimum interval of time between two successive activations of the task.

Since we consider that every task τ_i^j is scheduled by a fixed-task priority scheduler, we order the tasks by priority order in each application. The task τ_1^j has the highest priority while $\tau_{n_j}^j$ has the lowest priority.

2.2 Response Time Analysis

In this paper, we consider a simple test tool that will be developed for the case of IMA applications. This tool will compute the worst-case response time of each task in each application. Since we consider the system application per application, for the sake of simplicity, in the sequel, we will drop the exponent j of the application number. Response-time analysis (RTA) in the IMA context is an extension of the classic RTA, which is based on the request bound function (rbf) [8]. When considering a sporadic task τ_i, released at time 0, its worst-case response time cannot be lower than its own WCET C_i.

Classic Non Hierarchical RTA

1. Therefore, we know that the processor will be busy at least in the time interval $[0, C_i)$. Let's denote rbf_i^1 the value of the end of this interval. In this time interval, any release of a higher priority task $\tau_k, k < i$ is postponing the end of the execution of τ_i. The amount of releases of higher priority tasks in the interval $[0, rbf_i^1)$ occurs if all the higher priority tasks are released at the time 0, and execute with their shortest period (they are sporadic). This is called the **critical instant**. Each task $\tau_k, k < i$ can postpone the end of τ_i by up to $\left\lceil \frac{rbf_i^1}{T_k} \right\rceil C_k$.
2. The processor will be busy with tasks whose priority is at least the priority of τ_i at least in the time interval $\left[0, rbf_i^2 = C_i + \sum_{k=1}^{i-1} \left\lceil \frac{rbf_i^1}{T_k} \right\rceil C_k\right)$. In this time interval, the amount of times every higher priority task can postpone the end of τ_i may be higher, and given, for each task $\tau_k, k < i$ by $\left\lceil \frac{rbf_i^2}{T_k} \right\rceil C_k$
3. Going back to point 1, substituting rbf_i^1 by rbf_i^2, we obtain at the smallest fixed point of rbf_i^* the worst-case response time of τ_i as long as it is not greater than its period (because if it was greater than its period, then τ_i may delay itself and the formula should be adapted like in [10].

We therefore consider the classic RTA test stating that:

– the worst-case execution time of a constrained deadline task ($D_i \leq T_i$) is obtained as the smallest fixed-point rbf_i^* of the equation $rbf_i^* = C_i + \sum_{k=1}^{i-1} \left\lceil \frac{rbf_i^*}{T_k} \right\rceil C_k$ if $rbf_i^* \leq T_i$. If $rbf_i^* > T_i$ then τ_i will miss its deadline.

RTA for IMA. The main difference between hierarchical and classical RTA is that, in classical RTA, the processor dedicates on time unit of its time every time unit, while in hierarchical scheduling, the processor dedicates one time unit every time unit to the elected processor partition only. This phenomenon is accounted for by using a supply function of the processor.

The supply function is depending on the beginning of the MAF. Since there is no information concerning the offset between a considered instant for a task under analysis, and the beginning of the MAF, we have to consider any offset

between the tasks critical instant to study and the offset in the MAF. The lowest amount of processor supplied to a processor partition $\Gamma = \{(S_1, E_1), (S_2, E_2), ..., (S_N, E_N)\}$ is given, in an interval of length t, by considering the minimal supply function assuming the critical instant of the tasks coincides with $E_1, E_2, ...,$ or E_N [6]. These points are called **critical points**.

Definition: the supply function of the processor in a time interval $[0, t]$, depends on the critical point $E_1, E_2, ...,$ or E_N coinciding with the critical instant at the date 0. It is denoted $sf(E_i, t)$.

When computing the time required by the processor to compute the workload given by the rbf, we need to consider how long, in the worst case, is necessary for the processor partition to be supplied with enough processor time to compute the rbf. This is given by the reverse supply function sf^{-1} (see [6] for the way to compute this function).

Theorem [6]**:** the worst-case execution time of the first job of a task τ_i of an application scheduled in a processor partition $\Gamma = \{(S_1, E_1), (S_2, E_2), ..., (S_N, E_N)\}$ is given by the maximal for $E_j, j = 1..N$ of the smallest fixed-point $rbf_i^*(E_j) = C_i + \sum_{k=1}^{i-1} \left\lceil \frac{sf^{-1}(rbf_i^*(E_j))}{T_k} \right\rceil C_k$. If this value is not greater than the task's period T_i, then it is giving the worst-case response time of the task.

3 MoSaRT Framework in a Nutshell

The objective behind the MoSaRT framework is to benefit from analysts/researchers skills and modelers/architects skills in order to unify their efforts, then to avoid wrong design choices at an early design phase. MoSaRT framework plays a dual role aiming (i) to help designers to cope with the scheduling analysis and to be more autonomous during the analysis stage, and (ii) to help analysts to alleviate their efforts related to the analysis of design models.

Figure 1 shows the general architecture of the framework, which contains two parts: front-end and back-end as described in the following paragraphs.

Front-End of MoSaRT Framework is based on a design language. It provides a set of concepts to model different architectures of real-time systems characterized by temporal extra-functional properties (e.g. execution times, deadlines, periods). The MoSaRT design language is a graphical language developed using Ecore [15] and Object Constraint Language [12] (OCL, which is a textual language fitting with Ecore and enabling to express invariants). The modeling editor provides different model views such as hardware architecture diagram, software architecture diagram and behavioral diagram.

Back-End Part of MoSaRT Framework is based on a meta-model of an analysis repository enabling researchers and analysts to share their analysis models and tests. Every instantiation of the meta-model leads to obtain a repository

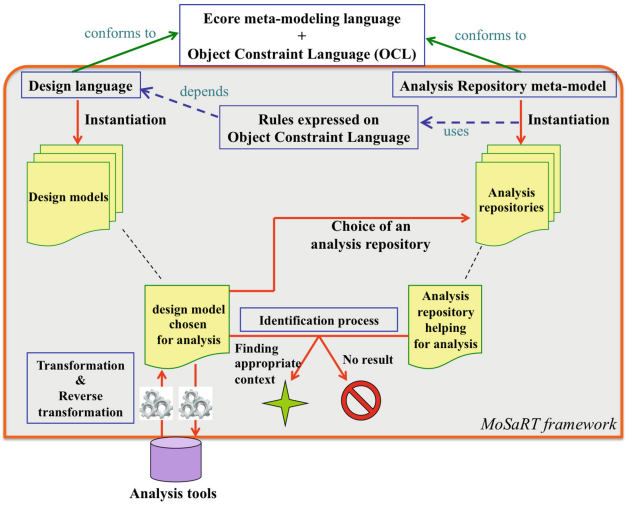

Fig. 1. General MoSaRT framework architecture and excerpt of the main workflow

playing the role of an advisor during its utilization with a model conforming to MoSaRT design language. The utilization of a repository with a model consists on the running an identification process in order to check if the repository contains a context (a set of architectural and timing behavioral characteristics) corresponding the model needs analysis. Once a context is found, the designer can use the proposed analysis tool. This latter can be reached thanks to a transformation process (which can be model-to-model or model-to-text) provided by the framework. The definition of contexts inside an instance of the analysis repository is based on the utilization of OCL and MoSaRT design language in order to express the characteristics.

4 Extension and Enrichment of MoSaRT Framework

This section presents our proposition to deal with IMA systems, which consists in extending the MoSaRT design language. Moreover, we highlight the analysis model of the developed prototype, then the rules a design (expressed using the extended MoSaRT language) has to conform to in order to be correctly analyzed.

4.1 Extension of MoSaRT Design Language

Figure 2 shows an overview of the proposed extension. We have designed this extension using the eclipse modeling framework [15], which is also used for the MoSaRT design language core. The elements added are those whose name are prefixed by "*Ima*". In the following, we detail each element.

– *ImaPartition* class represents the temporal partition. It has been added as an element of software architecture. A partition is related to a process (instance of *SoSpaceProcess* class), which can contain several partitions. Moreover, inside

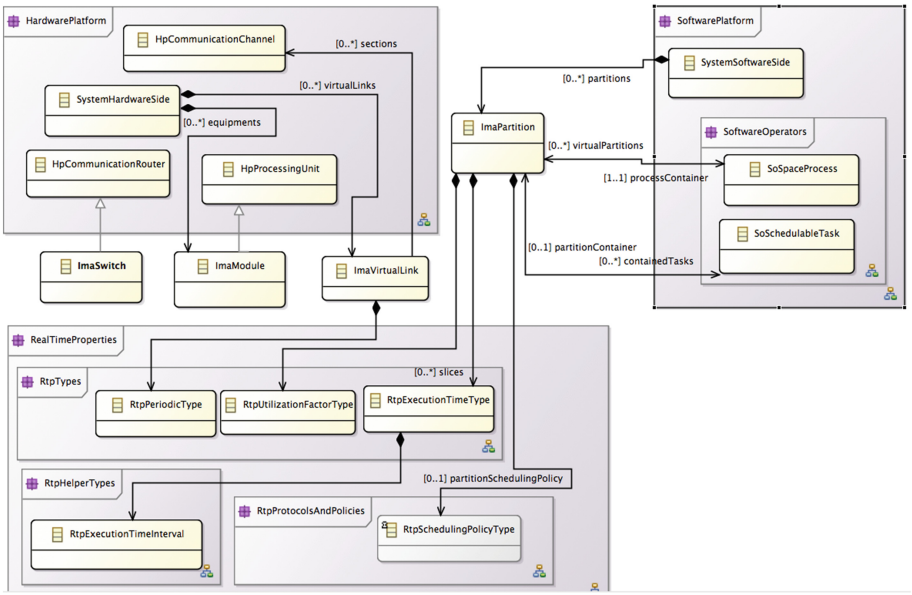

Fig. 2. Overview of the extended MoSaRT language meta-model

a partition, a set of tasks (instances of *SoSchedulableResource* class) can be executed. Thanks to the way MoSaRT design language has been implemented, we can notice that all the temporal properties needed have been meta-modeled as concept, then the enrichment of the *ImaPartition* class has been done easily. For instance, the execution time of a partition is a set of time intervals, which are instances of *RtpExecutionTimeInterval* class (Fig. 2).

- *ImaModule* class represents the module which executes a set of partitions through their container process. It has been added as an element of Hardware architecture. The *ImaModule* class inherits from the *HpProcessingUnit* class. This latter can be uniprocessor or multiprocessor.
- *ImaVirtualLink* and *ImaSwitch* classes have been added as elements of the hardware architecture. Those elements are dedicated to design distributed architectures, which is not presented in this paper due to space limitation.

The extension also consists of adding a set of structural rules which have to be verified in order to have a coherent system design. For instance, the time intervals of a partition may not overlap. The structural rules have been implemented on OCL and can be checked automatically via MoSaRT editor.

4.2 Example: Utilization of the MoSaRT Design Language After Extension

Thanks to the extension added to the MoSaRT language an architecture like the one shown in Fig. 3 can be totally designed. The architecture represents

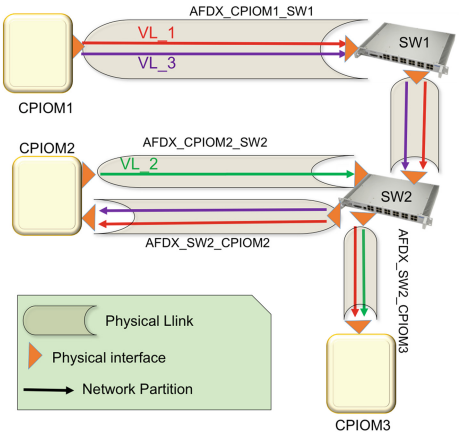

Fig. 3. Distributed architecture

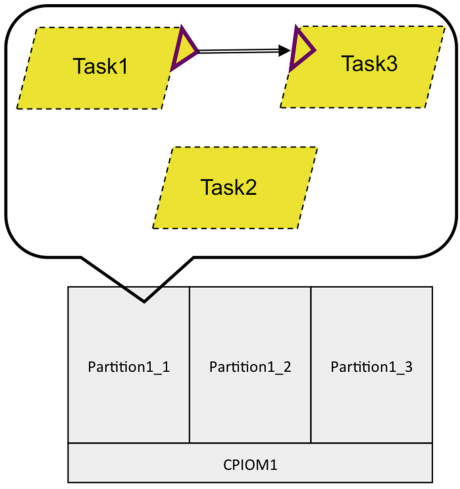

Fig. 4. Content of CPIOM1

Table 1. Timing properties of the task-set

Tasks	Priority	Period (ms)	Deadline (ms)	Execution time (ms)
`Task1`	15	8	8	1
`Task2`	10	14	14	2
`Task3`	5	30	30	4

a communication between a set of tasks executed in different CPIOMs (Core Processor Input/output Module) communicating through an AFDX network (Avionics Full DupleX). The networking is based on various switches and physical links. Therefore, via the extended design language we can model:

- physical links as instances of HpCommunicationChannel;
- switches as instances of ImaSwitch;
- CPIOMs as instances of ImaModule;
- network partitions as instances of ImaVirtualLink;

This paper does not discuss the distributed architecture, then we only focus on the software content of CPIOM1 of Fig. 3. Figure 4 shows a hierarchical architecture composed of three partitions where the first partition contains three independent tasks.

Table 1 presents the timing properties related to tasks and partitions. Moreover, the period of the partition *Partition1_1* is 30 ms and the time slices (i.e. time intervals) allocated to it are (0,5), (10,15) and (20,25).

Figure 5 represents a part of a MoSaRT model compliant with the software architecture of the example. The model highlights the software structure (i.e. tasks and partition) and temporal behavior of tasks (i.e. task activities and triggers).

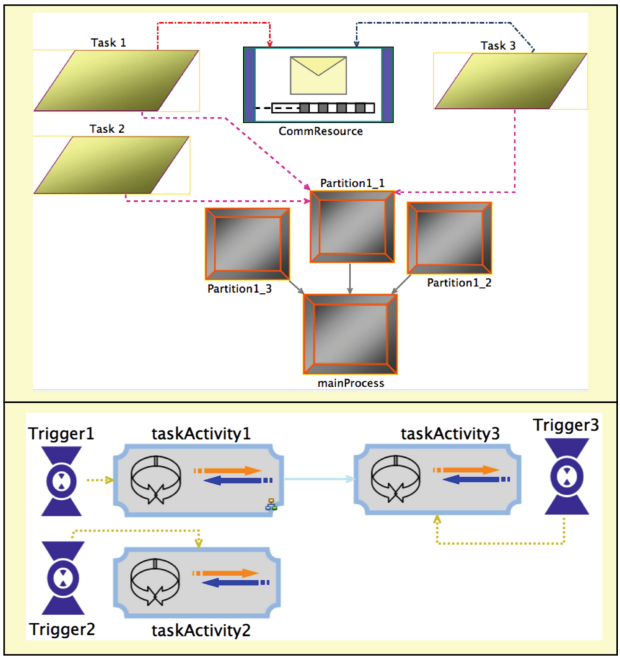

Fig. 5. Excerpt of a model expressed using the extended MoSaRT language

4.3 Implementing a Prototype of the Test

To the best of our knowledge, the test presented in Sect. 2.2 could not be found in any freely available schedulability analysis toolset. It is easy to implement, and the most time consuming part of a tool implementing this test would be to connect it to a design framework in order to call it when necessary, as well as to check if an input model fits with the underlying hypothesis of this test: independent tasks, IMA hierarchical scheduler, sporadic constrained deadlines tasks, uni-processor CPU. Using MoSaRT framework, we show how it is easily possible to connect such a tool to a design language, as well as to check if an input model can be handled by the developed tool. Hereafter, we show how the analysis repository can be instantiated to lead designers to this tool for analysis.

4.4 Example: Instantiation and Utilization of MoSaRT Analysis Repository

In order to assist designers to analyze models compliant with IMA architecture specifications, we have used the analysis repository of MoSaRT back-end. We can create a new repository or enrich an existing one. Due to the lack of space we prefer the first way, because using an existing repository requires to explain its content. Therefore, the new created repository is based on the context Feng analysis model [6], which is defined by a set of characteristics, such that:

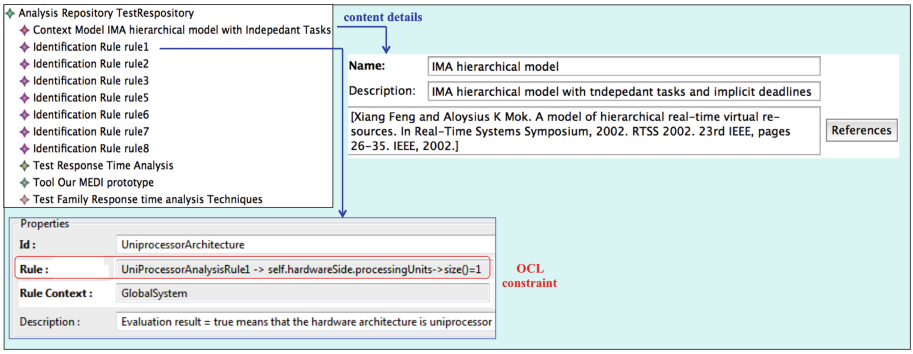

Fig. 6. Excerpt of a repository containing the analysis context related to hierarchical scheduling

- The architecture of CPOM is uniprocessor;
- Tasks are independant;
- No network used in the hardware architecture;
- Fixed priotity scheduling policy is used in partition level;
- etc.

If a design does not respect these characteristics, it does not mean that the structure of the design is incorrect. However, when all characteristics are respected we can conclude that the analysis context of the design is the one presented in Sect. 2. Figure 6 is our instance of the MoSaRT analysis repository. This instance contains several characteristics equivalent to a set of identification rules, where each rule is an OCL constraint (e.g. rule1 on Fig. 6). We also link the implemented analysis test to the context in order to check the schedulability of the design by calculating the response times of the tasks.

Once the new instance of the repository is ready to be used, we can connect it to a design expressed using MoSaRT language. For example, if we launch the MoSaRT identification process from the design of the example treated in Sect. 4.2, the design will be checked if it is compliant with the context added in the new repository. Moreover, the identification process of MoSaRT presents the result of this step. Figure 7 shows the analysis test corresponding to the characteristics of the design. The result window recapitulates the rules that are verified by the design and also the suitable analysis model. References related to tests and models are also presented in case when designers need more details.

Note that if the designer agrees with the proposed tests, this latter can be performed from the MoSaRT framework by doing a transformation to the input formalism of the analysis tools. Indeed, the worst-case response times of our case study are presented in Table 2.

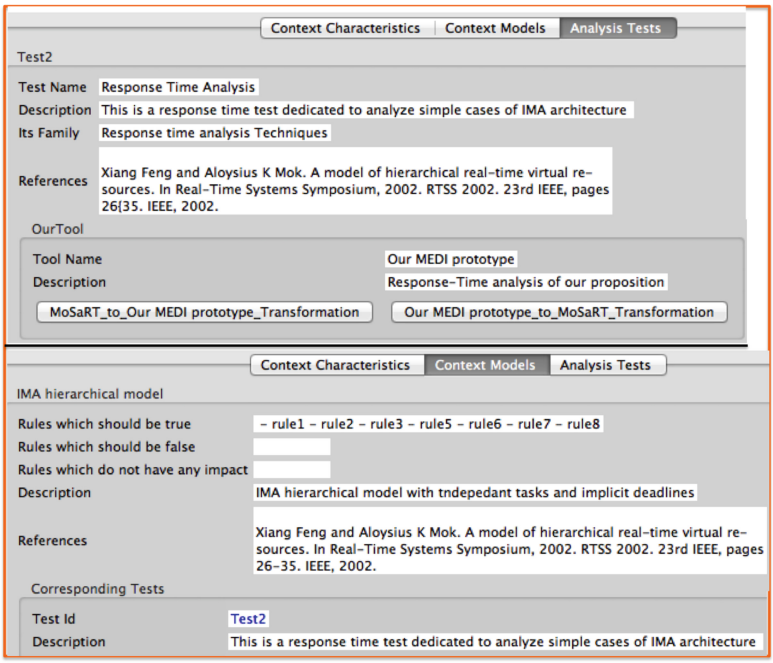

Fig. 7. Result of a MoSaRT identification process

Table 2. Analysis result

Task	Response Time (ms)	Schedulable
`Task1`	6	Yes
`Task2`	8	Yes
`Task3`	27	Yes

5 Conclusion

Numerous analysis tests have been proposed, but their utilization requires a colossal background related to identify the exact underlying hypothesis of each test, and the certainty of the analysis results depends on the context of each analyzed system. We underlined the advantages of having a framework such as MoSaRT framework, which facilitates the analysis phase of designers. Since MoSaRT framework is mainly based on model-driven engineering techniques, we had easily extended it in order to support the design of IMA architectures and ease their analysis. In our ongoing research, we will be interested in distributed avionics architectures by taking AFDX networks into considerations. Moreover, we will add various analysis tools to compare and discuss the output results.

References

1. ARINC. Avionics application software standard interface: Arinc specification 653 part 0. Arinc, Aeronautical Radio Inc. (2013)
2. Balbastre, P., Ripoll, I., Crespo, A.: Exact response time analysis of hierarchical fixed-priority scheduling. In: 15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA 2009, pp. 315–320. IEEE (2009)
3. Bril, R.J., Cuijpers, P.JL.: Analysis of hierarchical fixed-priority pre-emptive scheduling revisited. Technische Universiteit Eindhoven (TU/e) CS-Report, pp. 06–36 (2006)
4. Davis, R.I., Burns, A.: Hierarchical fixed priority pre-emptive scheduling. In: 26th IEEE International Real-Time Systems Symposium, RTSS 2005, p. 10. IEEE (2005)
5. Deng, Z., Liu, JWS., Sun, J.: A scheme for scheduling hard real-time applications in open system environment. In: Proceedings, Ninth Euromicro Workshop on Real-Time Systems, pp. 191–199. IEEE (1997)
6. Feng, X., Mok, A.K.: A model of hierarchical real-time virtual resources. In: 23rd IEEE Real-Time Systems Symposium, RTSS 2002, pp. 26–35. IEEE (2002)
7. ISO. ISO 26262–1:2011 road vehicles functional safety. ISO, International Organization for Standardization, Geneva, Switzerland (2011)
8. Joseph, M., Pandya, P.: Finding response times in a real-time system. Comput. J. **29**(5), 390–395 (1986)
9. Kuo, T.-W., Li, C.-H.: A fixed-priority-driven open environment for real-time applications. In: Proceedings, The 20th IEEE Real-Time Systems Symposium, pp. 256–267. IEEE (1999)
10. Lehoczky, J.P.: Fixed priority scheduling of periodic task sets with arbitrary deadlines. In: RTSS, vol. 90, pp. 201–209 (1990)
11. Lipari, G., Bini, E.: A methodology for designing hierarchical scheduling systems. J. Embed. Comput. **1**(2), 257–269 (2005)
12. OMG. Object constraint language, OMG available specification, version 2.0 (2006). www.omg.org/spec/OCL/2.0/
13. Saewong, S., Rajkumar, R.R., Lehoczky, J.P., Klein, M.H.: Analysis of hierarchical fixed-priority scheduling. In: Euromicro Conference on Real-Time Systems, pp. 173–173. IEEE Computer Society (2002)
14. Shin, I., Lee, I.: Periodic resource model for compositional real-time guarantees. In: 24th IEEE Real-Time Systems Symposium, RTSS 2003, pp. 2–13. IEEE (2003)
15. Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: EMF: Eclipse Modeling Framework. Pearson Education, Boston (2008)
16. Yassine, O.: Model-based framework for using advanced scheduling theory in real-time systems design. Ph.D. thesis, ISAE-ENSMA (2013)
17. Yassine, O., Emmanuel, G., Michael, R., Pascal, R., Frdric, M.: Mosart framework: a collaborative tool for modeling and analyzing embedded real-time systems. In: Boulanger, F., Krob, D., Morel, G., Roussel, J.-C. (eds.) CSDM, p. 12. Springer, Switzerland (2014)

http://www.springer.com/978-3-319-23780-0

Model and Data Engineering
5th International Conference, MEDI 2015, Rhodes, Greece, September 26-28, 2015, Proceedings
Bellatreche, L.; Manolopoulos, Y. (Eds.)
2015, XIX, 343 p. 99 illus., Softcover
ISBN: 978-3-319-23780-0