From Model-based design to Real-Time Analysis

Yassine Ouhammou, Emmanuel Grolleau and Michael Richard
LIAS / ISAE-ENSMA
86961, Futuroscope, France
{*ouhammoy, grolleau, richardm*}*@ensma.fr*

Pascal Richard
LIAS / University of Poitiers
86961, Futuroscope, France
pascal.richard@univ-poitiers.fr

***Abstract*—Real-time and embedded systems are sharply impacted by wrong design choices detected at a very late stage of the life-cycle. This impact is often due to timing constraints which are related to structural and scheduling analysis capabilities. The timing constraints analysis requires an expertise in both design and scheduling analysis. In this paper, we highlight some temporal analysis related difficulties. In order to help designers and to improve the real-time system design to be corrected at an early stage, we propose an approach which is based on modeling oriented scheduling analysis.**

Keywords-model-based design; structure verification; scheduling analysis validation.

I. INTRODUCTION

The temporal validation of real-time systems is mainly based on the scheduling theory or model checking of the system using a formal model (timed Petri nets, timed automata, temporal logic, etc.). For example, the system analyst or expert system designer can analyze the temporal behavior of a set of tasks scheduled by a scheduling algorithm using algebraic methods, called feasibility tests, in order to prove that temporal constraints will be met at run-time.

The complexity of real-time systems leads to use model driven engineering which has gained more popularity among real-time system developers. Recently, the integration of scheduling analysis in a model driven engineering process has improved. On the one hand, several standardized design languages such as the UML-MARTE [1] or AADL [2], provide sets of non-functional properties for the specification, analysis and automated integration of real-time performance of critical distributed computer systems. On the other hand, many commercial and free schedulability analysis tools provide some scheduling analysis tests in order to help designers during the analysis phase such as MAST [3], Cheddar [4], etc. The meta-models of those timing analysis tools differ. Therefore, each of these tools uses a different set of concepts to create the input models for simulation and analysis. Despite all these standard design languages and these analysis tools, the use of these standards for scheduling analysis is still expensive in terms of design expertise.

In this paper, we propose a process assisting the designer step by step in order to verify that a system structure is coherent and respects all real-time architectural and behavioral rules. Once all structural rules are respected, we help designers to choose the feasibility tests corresponding to their design by extracting real-time information, analyzing it as a task model and then checking if the extracted information respects the task model assumptions of a third-party tool. Our approach is based on a decision tree.

The remainder of this article is organized as follows. The next section is an overview of real-time scheduling concerns and some related works. Section III introduces our global process, the relevant concepts and their utilizations. Section IV presents the analysis aspects of our approach. Finally, Section V summarizes and concludes this article.

II. BACKGROUND AND RELATED WORK

A real-time system is interacting with a physical process in order to insure a correct behaviour. For this, it is implemented as a set of parallel, interdependent functionalities. Parallelism is often ensured by the multitasking paradigm, relying on a operational layer offering task scheduling. Thus, the real-time problematic is based on three axes: the hardware equipments, the task models and the scheduling theory.

When validating a critical real-time system, starting from the real task system S, a task model S' which is a worst-case of the real system has to be chosen. Then, the worst-case behaviour of the task model S' is analyzed in an acceptable delay. All along this process, if the task model S' semantics is poor, then the way S is modeled leads to pessimistic feasibility analysis. In order to decrease the modeling gap between S and S', several advances have been made on the task models, like practical factors. As examples, the multiframe models which have been proposed for multimedia systems [5], serial communication systems use transaction models [6] [7], self-suspension tasks [8], precedence constraint anomalies [9], etc.

Real-time scheduling theory provides a set of scheduling algorithms and algebraic methods called feasibility tests. Scheduling theory has been originally studied for the basic Liu and Layland model [10], and extended to cover more advanced and precise task models. Usually, feasibility tests prove that a software model using a set of hardware resources will respect real-time requirements. The time complexity of the analysis is a very important point: as an

 ISBN: 978-1-61208-233-2

example, testing the schedulability of periodic independent tasks with an implicit deadline using a deadline-driven scheduler on a single processor is a very easy problem and requires a linear time in the number of tasks. A small change in the hardware, software or operational context has a big impact on the temporal analysis. For example, with the same hypothesis, if we consider a fixed-priority scheduling policy, in the case where the periodic tasks are either non strictly periodic, or strictly periodic and released simultaneously. Then, the feasibility problem is NP-hard in the weak sense (pseudo-polynomial feasibility test). But, if the tasks are strictly periodic and not released simultaneously, then the feasibility problem is NP-hard in the strong sense, while one could, for test efficiency reasons, choose to use a pseudo-polynomial feasibility test as a sufficient (but non-necessary) test for this problem.

In order to obtain a thin design granularity and to avoid the over-sizing of the hardware resources of critical systems, some research works have proposed several results in this major, such as the sensitivity analysis [11], [12], the priority assignment [13], and the multi-criteria optimization [14].

The architectural verification and temporal validation of real-time systems can be difficult for system designers. Recently some research works aim at helping designers during the modeling phase in order to get a coherent system architecture. Roquemaurel et al. [15] are interested just in the architectural validation to ensure the architecture coherence by using formal methods and constraint satisfaction problems. Plantec et al. [16] have proposed a panel of design patterns, once the system design respects design pattern rules then a scheduling analysis corresponding to this pattern can be be applied [16]. Peres et al. [17] have also proposed a verification method for real-time system by using model checking. These research studies focus on one kind of validation (architectural requirements or temporal requirements) of the real-time systems. Moreover, for those which are interested in real-time scheduling, they do not reduce the gap between the real system and the task model, nor offer the closest scheduling model when the real system does not correspond to an existing analyzable model.

We propose an approach based on model driven engineering in order to assist designers to validate their architecture during the design phase and then to facilitate the choice of the appropriate analysis during the design phase. This approach called MoSaRT (Modeling oriented Scheduling analysis of Real-Time systems) consists of a domain specific language enriched by a formal language. Our goal is not to compete with other design languages or to compete with existing scheduling analysis tools, but we suggest MoSaRT to cope with the modeling difficulty and to reduce the gap between the standardized real-time languages and temporal analysis tools.

III. Modeling oriented scheduling analysis

A. Brief description of MoSaRT language

MoSaRT framework is an intermediate framework between real-time design languages and temporal analysis tools. This framework is based on the MoSaRT language which is conceived as a domain specific modeling language for embedded real-time systems. It is based on four pillars: the platform, the behavior, the analysis and the functional model. These aspects are based on the notion of viewpoint complementarity. Each viewpoint represents a side of the global system (see Figure 1).

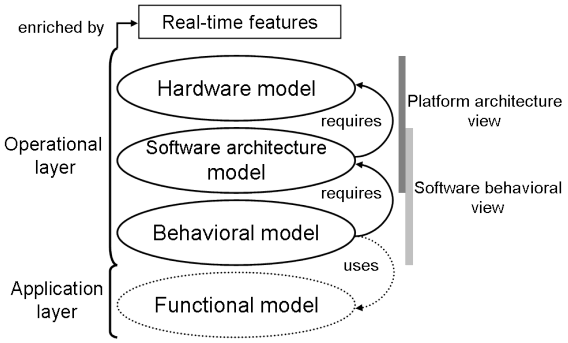

Figure 1. Different views of a real-time system designed through MoSaRT language

MoSaRT language is an instance of Ecore [18], which is a scripting language for meta-models. Moreover, Ecore is an implementation of the MOF (Model Object Facility) [19] under the integrated development environment Eclipse [www.eclipse.org] which is a well-equipped framework.

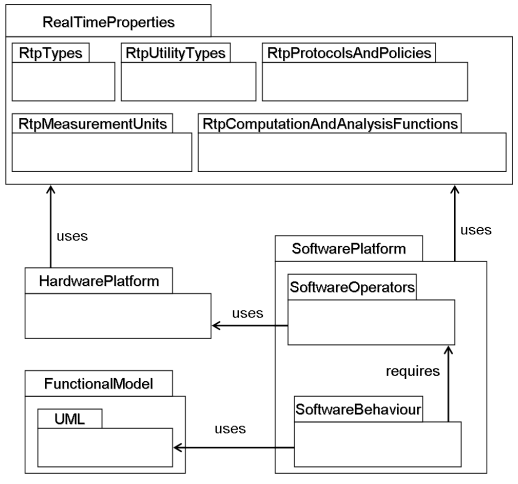

Figure 2. General structure of MoSaRT language

The architecture of MoSaRT language is organized in various packages and sub-packages. Figure 2 shows a global overview of the MoSaRT design language. For more details about MoSaRT language, readers can see related

 ISBN: 978-1-61208-233-2

papers [20], [21].

B. Formal semantics for MoSaRT language

The MoSaRT meta-model uses similar concepts as those used in the real-time system domain. For a good understandability of structural rules (Section IV-A), this section defines the main structural concept semantics and their inter-relationships without introducing the real-time properties.

Definition: Global System: In MoSaRT meta-model, the global system element "gs" represents the real-time system. It is characterized by the software platform S_{gs}, the hardware platform H_{gs} and the application F_{gs}. Then, gs = $<S_{gs}, H_{gs}, F_{gs}>$.

Definition: System's software platform: The software platform is composed of the software operators and the software behavior. MoSaRT language contains many software operators for modeling the tasks, the processes, the interaction resources (remote and local), etc. In order to ensure software operators consistency, especially tasks, behavioral elements offer to the designers the possibility to express the communication relationship, the precedence relationship, the trigger mode, etc. Thus: $S_{gs} = <S_O, S_B>$, where S_O is a subset of software operators set $\mathbb{SO}$, and S_B is a subset of software behavioral elements set $\mathbb{SB}$.

Definition: Software Operators: We define some mathematical notions that are used in this section:
- $\exists!$ means there exists exactly one
- f is a total function from the set E_A to the set E_B, if $\forall$ a $\in E_A$, $\exists!$ b $\in E_B$ such that f(a) = b
- h is a partial function from E_A to E_B, if A $\subset E_A$ and $\forall$ a $\in$ A, $\exists!$ b $\in E_B$ such that h(a) = b
- g is a total bijection from E_A to E_B, if and only if $\forall$ b $\in E_B$, $\exists!$ a $\in E_A$ with b = g(a)
- r is a relation from E_A to E_B, if A $\subset E_A$ and $\forall$ a $\in$ A, $\exists$ B $\subset E_B$ where r(a)= B
- $\oplus$ means "exclusive or"

We admit that:
- $\mathbb{LCR}$ is a set of Local Communication Resources
- $\mathbb{RCR}$ is a set of Remote Communication Resources
- $\mathbb{EMR}$ is a set of Exclusion Mutual Resources
- $\mathbb{TA}$ is a set of Task Activities that we define further

Then:

- $\mathbb{IR}$ = $\{ir_1, ir_2, ..., ir_n\}$ = $\mathbb{LCR} \cup \mathbb{RCR} \cup \mathbb{EMR}$, is a set of interaction resources, where each ir_i denotes an interaction resource, and $\mathbb{LCR}$, $\mathbb{RCR}$ and $\mathbb{EMR}$ are disjoint sets. Thus: $\forall$ $ir_i \in \mathbb{IR}$, $ir_i \in \mathbb{LCR} \oplus ir_i \in \mathbb{RCR} \oplus ir_i \in \mathbb{EMR}$
- $\mathbb{ST}$ = $\{st_1, st_2, ..., st_n\}$ is a set of Schedulable Tasks, each task is characterized by $st_i = <sp_j, \Omega_i, \Xi_i, \Psi_i, ta_i>$ where:
  - belongsTo is a total function defined as: $\forall$ $st_i \in \mathbb{ST}$, belongsTo(st_i) = sp_j, $sp_j \in \mathbb{SP}$
  - writesOn is a relation defined as: $\forall$ $st_i \in \mathbb{ST}$, writesOn(st_i) = Ω_i, $\Omega_i \subseteq \{ \mathbb{LCR} \cup \mathbb{RCR}\}$
  - readsFrom is a relation defined as: $\forall$ $st_i \in \mathbb{ST}$, readsFrom(st_i) = Ξ_i, $\Xi_i \subseteq \{ \mathbb{LCR} \cup \mathbb{RCR}\}$
  - accessesTo is a relation defined as: $\forall$ $st_i \in \mathbb{ST}$, accessesTo(st_i) = Ψ_i, $\Psi_i \subseteq \mathbb{EMR}$
  - representedBy is a total bijection defined as: $\forall$ $st_i \in \mathbb{ST}$, representedBy(st_i) = ta_i, $ta_i \in \mathbb{TA}$
- $\mathbb{SP}$ = $\{sp_1, sp_2, ..., sp_n\}$ is a set of Space Processes, each process is characterized by $sp_i = <T_i, sp_j>$ where:
  - $\forall$ $st_j \in T_i \subseteq \mathbb{ST} \Rightarrow$ belongsTo(st_j) = sp_i
  - inherits is a partial function defined as: $\forall$ $sp_i \in \mathbb{SP}$, inherits(sp_i) = sp_j, $sp_j \in \mathbb{SP}$

Thus, $\mathbb{SO} = \mathbb{IR} \cup \mathbb{ST} \cup \mathbb{SP}$ is a set of Software Operators.

Definition: Software Behavior: Let $\mathbb{TR}$ is a set of Triggers, and:

- $\mathbb{TA}$ = $\{ta_1, ta_2, ..., ta_n\}$ is a set of Task Activities, each task activity is defined as $ta_i = <st_i, tr_j, IA_i, OA_i, \Lambda_i>$ such that:
  - representedBy^{-1}(ta_i) = st_i, $st_i \in \mathbb{ST}$
  - triggredBy is a partial function defined as: $\forall$ $ta_i \in \mathbb{TA}$, triggredBy(ta_i) = tr_j, $tr_j \in \mathbb{TR}$
  - precededBy is a relation defined as: $\forall$ $ta_i \in \mathbb{TA}$, precededBy(ta_i) = IA_i, $IA_i \subseteq \mathbb{TA}$ and $\forall$ $ta_j \in OA_i \subseteq \mathbb{TA} \Rightarrow \exists$ $ta_i \in$ precededBy(ta_j)
  - containedBy is a total function defined as: $\Lambda_i \subseteq \mathbb{AS}$, $\forall$ $as_j \in \Lambda_i$, containedBy(as_j) = ta_i
- $\mathbb{AS}$ = $\{as_1, as_2, ..., as_n\}$ is a set of steps, each step is defined as $as_i = <\kappa_i, IS_i, OS_i>$ where:
  - $\kappa_i \in$ K = {action, acquire, release, send, receive, read, write} which is a set of step kinds.
  - stepPrecededBy is a relation defined as: $\forall$ $as_i \in \mathbb{AS}$, stepPrecededBy(as_i) = IS_i, $IS_i \subseteq \mathbb{AS}$ and $\forall$ $as_j \in OS_i \subseteq \mathbb{AS} \Rightarrow$ $as_i \in$ stepPrecededBy(as_j)

So, $\mathbb{SB} = \mathbb{TR} \cup \mathbb{TA} \cup \mathbb{AS}$ is a set of Software Behavioral elements.

In this subsection, we have shown just a part of some MoSaRT elements and their relationships. We have not introduced the real-time properties, nor how we relate the operational model to the functional model without any impact on the functional layer.

IV. Structural and scheduling analysis

In object-oriented modeling, graphical models are not sufficiently expressive to be able to express an entire precise specification. Moreover, it is often necessary to describe additional constraints on model instances. To specify these

constraints, formal languages have been developed. Generally, these languages use complex notations that require mathematical knowledge. Nevertheless, clarity and simplicity are among MoSaRT purposes. Moreover, MoSaRT can be used by different actors (designers and analysts) who are not necessarily proficient in different fields. Therefore, MoSaRT is enriched by several rules in different severity. It encapsulates these rules and generates just the error or information message which can be understood easily. For implementing structural and analysis rules, we have opted for OCL (Object Constraint Language) [22] for two reasons. The first one is the clarity of OCL, which is a formal language that is conceived to be read and written easily. The second reason is related to the meta-meta-model used, which is Ecore [18]. Ecore language operates very well with OCL and allows to define the constraints, the operations and the derived properties.

A. *MoSaRT structural rules*

In this section, we propose the different kinds of structural rules that must be respected in order to have a coherent design. We have separated structural rules into three groups. Architectural rules, vivacity rules ans safety rules. We treat these different kinds of rules through an example.

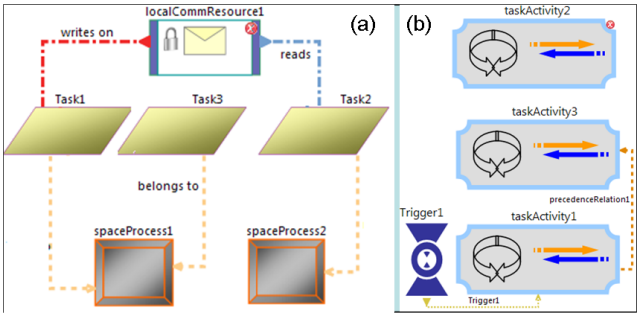

Figure 3. Checked models: Architectural rule (a) and vivacity rule (b) are violated

We consider a real-time system which contains two processes, a set of tasks and a set of interactions resources (see part (a) of Figure 3). The architectural constraints ensure the possibility of a good utilization of all elements that are modeled in the platform architecture. For instance, if *Task1* and *Task2* communicate through a local communication resource then *Task1* and *Task2* must belong to the same space process. This rule is formalized as follow:
$\forall$ $st_i \in \mathbb{ST}$ and $\forall$ $st_j \in \mathbb{ST}$
if $(\Omega_i \cup \Xi_i) \cap (\Omega_j \cup \Xi_j) \neq \{\}$
and $((\Omega_i \cup \Xi_i) \cap (\Omega_j \cup \Xi_j)) \subset \mathbb{LCR}$
then $sp_{st_i} = sp_{st_j}$
Listing 1 shows the implementation of this architectural rule as an OCL invariant.

Listing 1. An expressed architectural rule in OCL

```
invariant SoLocalCommResourceRule1:
(self.oclAsType(SoCommunicationResource).writerTasks
->union
(self.oclAsType(SoCommunicationResource).readerTasks))
->forAll(t1,t2 | t1 <> t2 implies t1.process = t2.process);
```

Consequently, the architecture that is shown in Figure 3 (part (a)) is not validated.

The vivacity rules ensure the correctness and completeness of the global behaviour. This kind of rules is very important especially for behavioral model. For example, in the behavioral model of real-time system designed with MoSaRT language, a scheduling activity must be triggered by a time trigger or external event trigger. Else, a scheduling activity must be preceded by another scheduling activity. This precedence relationship can be designed as a precedence synchronization or as communication relationship. So,
$\forall$ $ta_i \in \mathbb{TA}$
if $\nexists$ $tr_j \in \mathbb{TR}$ where triggredBy(ta_i) = tr_j
then precededBy(ta_i) $\neq \{\}$
By translating this rule to OCL, we obtain:

Listing 2. An expressed vivacity rule in OCL

```
invariant SbTaskActivityRule1:
self.oclAsType(SbSchedulingActivity).trigger
->isEmpty() implies self.oclAsType(SbSchedulingActivity).
inputSquencingRelation->notEmpty();
```

Therefore, *TaskActivity1* and *TaskActivity3* which are shown in Figure 3 (part (b)) respect the vivacity rule that is previously defined, contrariwise to *TaskActivity2*.

Safety rules guarantee that no erroneous behavior will happen especially when a designer enriches the model. For instance, if the objective of a designer is to have a detailed design, then the designer can specify the core of a task activity by adding steps.

Among MoSaRT safety rules, we can find the following one:
$\forall$ $as_i \in \mathbb{AS}$ if κ_i = *acquire*, then $as_i \notin$ *stepPrecededBy*(as_i)

This rule means that an acquire step can not precede itself. An "acquire step" means that task gets a semaphore in order to access to a critical shared resource. Then, the defined rule must be respect to ensure a safe system behavior. The Listing 3 shows the corresponding OCL rule.

Listing 3. An expressed safety rule in OCL

```
invariant SbStepPrecedenceRelationRule1:
self.sourceStep.oclIsTypeOf(SbAcquireStep)
implies not self.targetStep.oclIsTypeOf(SbAcquireStep);
```

In this section, we have shown the structural rules which must be respected during the design phase. Through MoSaRT, the validation of a real-time system structure is incremental. This approach has two advantages. The first one is to cope with scaling problems. So, the verification process stops a the first violation rule. The second advantage is to keep the traceability, then inform the designer about the model element that causes the invalidation of the system. In the next section, we expose the way MoSaRT rules detect

 ISBN: 978-1-61208-233-2

the ability of a real-time system to be analyzable and then, how it proposes the possible scheduling analysis which is matching with the analyzable system.

B. MoSaRT scheduling analysis ability rules

The purpose of MoSaRT language is not to provide scheduling analysis, but to help designers to verify their systems in order to conclude about the schedulability of their systems and to help in its dimensioning. MoSaRT contains several scheduling analysis rules which guide designers to choose the appropriate scheduling analysis tests. Moreover, MoSaRT can also offer to a designer several external scheduling analysis tools that provide these tests, such as Cheddar [4], MAST [3], etc.
Each analysis test depends on the model completion phase. For example, sensitivity analysis [11][12] is a dimensioning technique which can be solicited at an early design phase. Moreover, the model completion is measured by the kind of real-time properties mentioned in a real-time system design. For instance, a design which can be mapped to a Liu and Lalyland task model [10] could be analyzed as a transaction model [6] if offset-time properties were mentioned.

In this paper, we focus on scheduling analysis validation. MoSaRT offers a set of scheduling analysis ability rules in order to check the meta-model instance and then to infer the task model that corresponds to this instance. In the case where no task model corresponds to the designed instance, MoSaRT should suggest the closest task model. To facilitate the understanding about the manner MoSaRT detects scheduling ability of a design, we expose that trough a simple example. This example is based on a simple real-time system. It is composed of three independent periodic tasks which are executed on a processor using a fixed priority scheduling policy. Each task is characterized by a release date r_i, a worst-case execution time Ci, a relative deadline Di, a period Pi and a priority $Prio_i$ (the smallest value is the highest priority). Table I summarizes these characteristics.

Task	r_i	Ci	Di	Pi	$Prio_i$
Task1	0 ms	2 ms	4 ms	4 ms	1
Task2	0 ms	1 ms	4 ms	4 ms	2
Task3	0 ms	1 ms	8 ms	8 ms	3

Table I
VALUE OF TASK CHARACTERISTICS

Figure 4 shows the design of this example through MoSaRT language. This model respects all structural rules; then, we can apply schedulability analysis rules in order to know the possible analysis tests which can match this model.

The scheduling analysis ability rules are a set of assumptions. These assumptions are collected from different task models and they are implemented in MoSaRT language as a set of OCL rules that are not necessarily true. These OCL

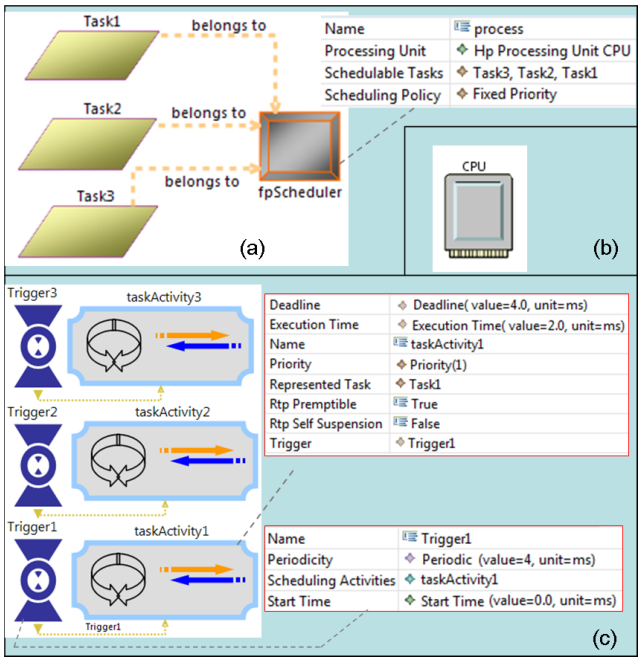

Figure 4. A simple real-time system that is designed using MoSaRT language: software model(a), hardware model(b) and behavioral model(c)

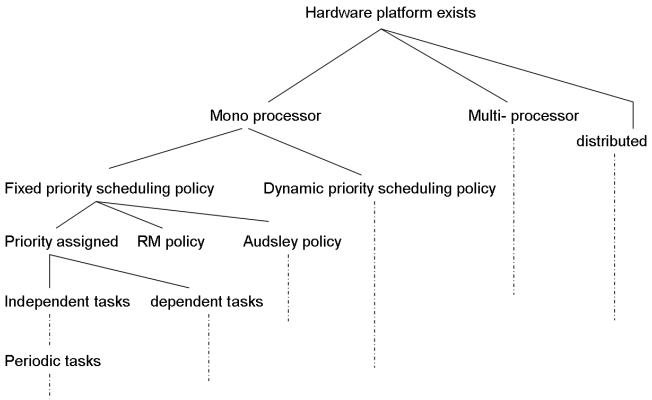

Figure 5. A decision tree of Scheduling analysis ability rules

rules are inter dependent and they are organized as a decision tree (see Figure 5). So, the verification of a rule requires the accuracy of other rules. For example, MoSaRT first checks if a hardware platform exists. If true, then it will verify if the hardware architecture is uniprocessor, else it will verify if the hardware architecture is multi-processor else it will deduce the hardware architecture is distributed. We note that a tree exploration gives the task model which corresponds to the design or else the nearest worst-case model. For instance, the design appearing in Figure 4 respects a set of rules which correspond to Liu and Layland task model [10] assumptions. Therefore, the simulation, the worst case response time calculation and the processor utilization calculation are the analysis tests corresponding to our design example. This

 ISBN: 978-1-61208-233-2

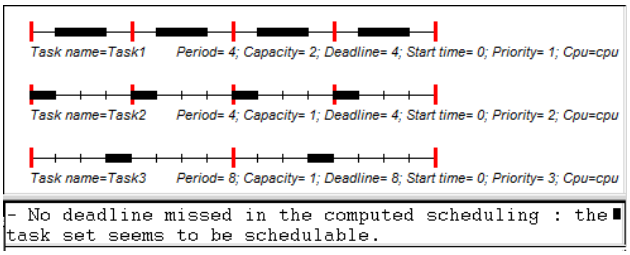

Figure 6. Result provided from Cheddar analysis tool

result does not mean that the system example is schedulable, but the system can be analyzed by a third-party tool providing the appropriate scheduling analysis tests. Figure 6 shows the simulation test applied by the real-time system example. This validation can be provided by Cheddar analysis tool.

V. Conclusion

We presented an approach to cope with design and analysis real-time system difficulty. MoSaRT is an implementation of this approach. It is a language which helps to design, to verify the system structure and to choose a schedulability test. It is based on Ecore and OCL. We presented two kinds of analysis that are based on a set of rules. In order to ease the understanding, the structural rules and scheduling analysis ability rules are presented through some examples. The transformation from MoSaRT to analysis tools is done manually. So, we are focusing on model transformation from MoSaRT to standardized design languages and then we will try to automatize the transformation process.

Acknowledgment

We thank Obeo that has provided us an academic version of Obeo-Designer product. We used this tool to implement our research works about the meta-modeling.

References

[1] OMG, “The uml profile for marte: Modeling and analysis of real-time and embedded systems,” www.omgmarte.org, [retrieved: Sept, 2012].

[2] SAE, “Architecture analysis and design language,” www.aadl.info, [retrieved: Sept, 2012].

[3] J. L. Medina Pasaje, M. González Harbour, and J. M. Drake, “Mast real-time view: A graphic uml tool for modeling object-oriented real-time systems,” in *RTSS 2001*, pp. 245–256.

[4] F. Singhoff, J. Legrand, L. Nana, and L. Marcé, “Cheddar: a flexible real time scheduling framework,” in *SIGAda 2004*, pp. 1–8.

[5] A. K. Mok and D. Chen, “A multiframe model for real-time tasks,” *IEEE Transactions on Software Engineering*, vol. 23, pp. 635–645, 1996.

[6] J. C. Palencia and M. González Harbour, “Schedulability analysis for tasks with static and dynamic offsets,” in *RTSS 1998*, pp. 26–37.

[7] A. Rahni, E. Grolleau, and M. Richard, “An efficient response-time analysis for real-time transactions with fixed priority assignment,” *ISSE*, vol. 5, no. 3, pp. 197–209, 2009.

[8] F. Ridouard, P. Richard, and F. Cottet, “Negative results for scheduling independent hard real-time tasks with self-suspensions,” in *RTSS 2004*, pp. 47–56.

[9] M. Richard, P. Richard, E. Grolleau, and F. Cottet, “Contraintes de precedences et ordonnancement mono-processeur,” in *Real Time and Embedded Systems*, Teknea, Ed., 2002, pp. 121–138.

[10] C. L. Liu and J. W. Layland, “Scheduling algorithms for multiprogramming in a hard-real-time environment,” *J. ACM*, vol. 20, no. 1, pp. 46–61, 1973.

[11] S. Vestal, “Fixed-priority sensitivity analysis for linear compute time models.” *IEEE Trans. Software Eng.*, pp. 308–317, 1994.

[12] E. Bini, M. Di Natale, and G. Buttazzo, “Sensitivity analysis for fixed-priority real-time systems,” *Real-Time Syst.*, vol. 39, pp. 5–30, August 2008.

[13] N. Audsley and Y. Dd, “Optimal priority assignment and feasibility of static priority tasks with arbitrary start times,” 1991.

[14] R. Mishra, N. Rastogi, D. Zhu, D. Moss, and R. Melhem, “Energy aware scheduling for distributed real-time systems,” in *IPDPS 2003*, p. 21.

[15] M. de Roquemaurel, T. Polacsek, J.-F. Rolland, J.-P. Bodeveix, and M. Filali, “Assistance la conception de modles l’aide de contraintes,” in *AFADL 2010*, pp. 181–196.

[16] A. Plantec, F. Singhoff, P. Dissaux, and J. Legrand, “Enforcing applicability of real-time scheduling theory feasibility tests with the use of design-patterns,” in *ISOLA 2010*, pp. 4–17.

[17] F. Peres, P.-E. Hladik, and F. Vernadat, “Specification and verification of real-time systems using pola,” *IJCCBS*, pp. 332–351, 2011.

[18] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks, *EMF: Eclipse Modeling Framework*. Addison-Wesley Professional, 2008.

[19] OMG, “Meta object facility,” www.omg.org/spec/MOF/2.4.1/, [retrieved: Sept, 2012].

[20] Y. Ouhammou, E. Grolleau, M. Richard, and P. Richard, “Towards a simple meta-model for complex real-time and embedded systems,” in *MEDI 2011*, pp. 226–236.

[21] ——, “Model driven timing analysis for real-time systems,” in *ICESS 2012*, pp. 1458–1465.

[22] OMG, “Object constraint language,” www.omg.org/spec/OCL/2.0/, [retrieved: Sept, 2012].

 ISBN: 978-1-61208-233-2