MoSaRT Framework: A Collaborative Tool for Modeling and Analyzing Embedded Real-Time Systems

Yassine Ouhammou, Emmanuel Grolleau, Michaël Richard, Pascal Richard, and Frédéric Madiot

Abstract. The increasing evolution of real-time and embedded systems needs methodologies and design tools in order to reduce design complexity. Moreover, the scheduling analysis is one of the aspects that integrate the development process to reduce development costs and to validate systems. Since model-driven engineering offers interesting solutions to the abovementioned challenges, it is widely used in various industrial and academic research projects. This paper presents an overview of a model-based framework called MoSaRT (*Modeling oriented Scheduling analysis of Real-Time systems*), which aims to help real-time designers to conceive, dimension and analyze real-time systems. The underlying idea behind this proposal is to fill the gap between the academic real-time scheduling theory community and industrial practices. In fact, research results have been exploited in industrial contexts only to a modest extent to date. The MoSaRT framework is also a software tool for technology transfer enabling researchers to promote their works (e.g. analysis models and scheduling tests), then to increase the applicability of the real-time scheduling analysis.

1 Introduction

Real-time and embedded systems have been widely used in different industrial areas, like transportation, nuclear plants, and telecommunications. A

Yassine Ouhammou · Emmanuel Grolleau · Michaël Richard · Pascal Richard
LIAS lab. (ISAE-ENSMA and University of Poitiers) - Futuroscope, France

Frédéric Madiot
Obeo, France
e-mail: {ouhammoy,grolleau,richardm}@ensma.fr,
pascal.richard@univ-poitiers.fr, frederic.madiot@obeo.fr

F. Boulanger et al. (eds.), *Complex Systems Design & Management*,
DOI: 10.1007/978-3-319-11617-4_20

real-time system is a system that must interact with a correct behavior to input events within specified timing bounds [2]. So, a result that is functionally correct, but not temporally correct (i.e. not respecting the deadline), is considered as a wrong behavior.

We are interested in the temporal correctness of hard real-time systems. A hard real-time system has to meet its timing requirements (i.e. in order to be schedulable), otherwise, something unacceptable and catastrophic can occur. The hard real-time system is composed on a set of tasks and messages sharing a set of execution/communication resources. The way of sharing resources depends on the scheduling algorithms, the network protocols and the memory access policies which are chosen by designers. To check if the used resources/policies/protocols are enough and well adapted for that tasks and messages always meet the timing requirements, the "scheduling analysis" is applied during the design phase not only to check the schedulability of hard real-time systems, but also to help designers to dimension system's architecture when the system design is not completely defined. The real-time scheduling analysis can be based on the model checking, the simulation or the analytical methods of the scheduling theory.

Nowadays, the utilization of the real-time scheduling theory in practical cases could be profitable. Unfortunately, it is not sufficiently applied and the research results have been exploited in the industry only to a modest extent to date. The chasm between the scheduling theory and the industrial practices may be due to several reasons. For instance, to master the scheduling techniques, a colossal work related to the real-time theory knowledge is required. However, systems designers may not be well versed in the real-time scheduling theory. Industrial designers are too busy to perform accurate analyses due to cultural and economical reasons (e.g. concurrency/competitiveness), and they are unwilling to take risks with new approaches. Furthermore, transferring the knowledge from the research area to an industrial area can be expensive. Even if many analysis tools exist, a tool cannot offer all the analysis models and techniques. Moreover, whereas an academic researcher develops a prototype easing the exploitation of a special research study, adding this prototype in different analysis environments may require to modify their internal structures. That needs a high development effort for a research group other than the original tool makers.

Our objective is to help designers to cope with the analysis difficulty, then to orient them in order to choose the most appropriate analysis tests and to ease the design modifications due to refinement or dimensioning actions. Therefore, designers will be guided to build scheduling-aware models. The second objective is to enhance the applicability of the real-time scheduling theory. Therefore, it is needful to provide an easy way for transferring the research studies from academia to industrial practices. To achieve the above objectives, we propose a framework called MoSaRT (Modeling-oriented Scheduling analysis of Real- Time systems). MoSaRT is also an intermediate framework between real-time design languages and schedulability analysis tools. In this paper, we

gather different parts of the MoSaRT framework which have been already published separately [14] [15] [13], then we present to readers a global view of this framework.

The rest of the paper is organized as follows. In the next section, we give a brief overview of real-time scheduling concerns. Section 3 gives a general idea introducing the MoSaRT framework. Section 4 describes the MoSaRT design language. Section 5 presents the MoSaRT analysis repository. Section 6 highlights some typical MoSaRT usage scenarios. Finally, Section 7 summarizes and concludes the paper.

2 Background and Related Work

Since 1970s, researchers of the real-time community have presented several research works dedicated to the scheduling analysis of hard real-time systems [18] [3]. On the one hand, these works consist on a set of analysis models. Indeed, an analysis model represents formal expressions admitting mathematical evaluations and based on temporal properties taking into consideration different task characteristics (like the precedence relationship or the self-suspension) and hardware architectures (uniprocessor, multi-cores processors, distributed systems, etc.). On the other hand, the research works have also tackled various analysis tests helping designers (especially analysts) to check the temporal validation of the real-time applications. While every analysis model is an extraction of the non-functional temporal properties from a system design, then the analysis tests depend on the analysis models. In fact, the kind of the analysis tests depends on the completion stage of the system design (i.e. does the system design need to be dimensioned or validated ?). Moreover, the efficiency and the consistency of an analysis test depend on the design and temporal characteristics of the system.

The steep learning curve behind many of the current analysis methods has been one of the major impediments to their adoption and their exploitation in the industry. Several works have treated the difficulty of the scheduling analyses utilization through a model-based engineering process. They proposed modeling languages and tools to decrease this difficulty. Recently, UML-MARTE [12] and AADL [1, 10] are among standard modeling languages that have been proposed. MARTE (Modeling and Analysis of Real-Time and Embedded Systems) is a UML profile that offers several stereotypes and tagged values helping designers to annotate their UML models (e.g. class diagrams). The purpose is to add temporal characteristics and constraints for further scheduling analyses. However, since UML-MARTE does not follow a specific standard methodology, semantics of the stereotypes differ from an utilization to another. Hence, a set of methodologies have been proposed using only a subset of UML-MARTE and with different semantics (like Optimum [9] and MADES [17]). The underlying idea behind those methodologies is not only to ease the utilization of a subset of UML-MARTE, but also to help designers

by proposing a task-set (i.e. design patterns of the tasks architecture) that fits with the functional model. AADL (Architecture and Analysis Description Language) is a component-based language leading to get hierarchical system architectures close to the reality, containing a set of hardware and software components. Although AADL does not allow designers to define the functional part of real-time systems, the architectures are expressed in a modular way. Nevertheless, the utilization of AADL only through the development life-cycle of embedded system can not help to get refined models iteratively. In other words, AADL does not enable to dimension models (e.g. allocation of tasks, mapping of functions, partitioning).

The implementation of the analysis techniques has also taken advantage of the model-driven engineering. Recently, several academic and industrial tools were proposed as providers of the well-known analysis techniques by offering the possibility to apply some subsets of schedulability tests during the design phase. Some examples of those tools are RT-Druid [4], SymTA/S [5] and Cheddar [19]. The utilization of the analysis tools provides often a simple Yes/No answer to the question "does the system meet all its deadlines?". This kind of information is not efficient and not helpful enough for real-time designers, in particular, when the analysis tool is not able to analyze the system. Moreover, analysis tools do not help designers to choose the appropriate tests which match the model requesting the analysis. So, even if the analysis result is provided, it may be very pessimistic due to the choice of a wrong analysis test.

3 Objectives of MoSaRT Framework

As the modeling and the scheduling analysis of real-time systems are both in constant evolution and improvement in distinct scientific communities, the design methodologies, design languages and analysis tests are sharply improved. Indeed, the methodologies are impacted by the hardware equipments, the software operating systems and the programing languages. While the model-driven engineering offers a relative independence regarding technological changes, and provides a re-usability of the design elements, that are measurable, predictable, and manageable, hence we are based on the model-driven engineering: (i) to unify modeling and analysis efforts,(ii) to achieve a friendly utilization taking benefits from standard design languages and timing analysis tools and (iii) to increase the applicability of the real-time scheduling theory. Consequently, we propose an intermediate framework named MoSaRT (Modeling-oriented Scheduling analysis of Real-Time systems). To partition the efforts, the intermediate framework plays the role of a bridge between the real-time design languages and the analysis tools (see Figure 1). It is based on two metamodels interacting with each other:

– MoSaRT Design Language, which is a domain specific language offering enough concepts and semantics to obtain models independent from any methodology, and to cover with few modifications, most existing analysis models and easily extended to include concepts enabling to support future notions.
– MoSaRT Analysis Repository metamodel allows analysts to plug different theoretical studies and prototypes and ensures the interoperability with different analysis tools in order to compare their output results.

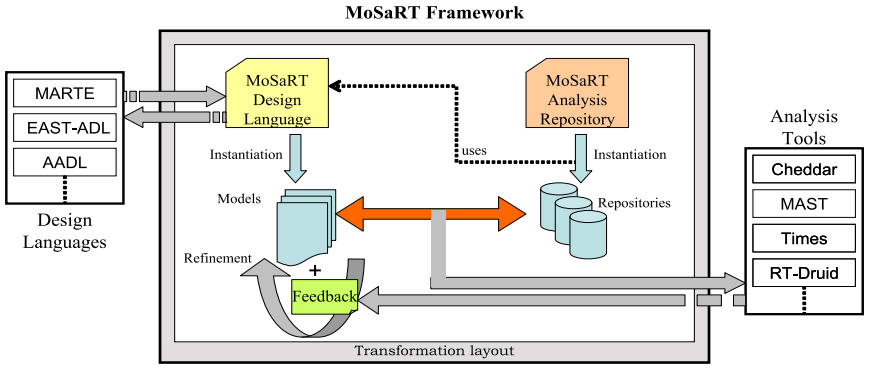

Fig. 1 MoSaRT Framework

Figure 1 gives only an overview of our contributions. It shows a generic scenario due to the usage of the MoSaRT framework. This latter helps designers to analyze step by step their system designs during the design phase, which can be increasingly improved by applying the three following processes:

– Using the MoSaRT design language for system modeling, or for refining imported models.
– Selection of the analysis models and the relevant tests, by helping the real-time designers to extract the relevant elements from the models.
– Scheduling analysis, by offering to the real-time designers the equipped analysis tests which correspond to their models via the proposition of one or several analysis repositories. The next sections discuss the details of each contribution.

4 MoSaRT Design Language

MoSaRT design language [14] [15] is conceived as a domain specific modeling language for real-time systems. It contains several concepts which are very close to the real-time analysis. The MoSaRT language is based on the notion of viewpoints complementarity by proposing different kind of models: hardware model, software architecture model, behavioral model and functional

model. The implementation of MoSaRT language is based on Ecore language [20] and Sirius (see Section Acknowledgment).

The MoSaRT language generic real-time properties: every real-time concept (like execution-time property) has been meta-modeled to support different kinds of systems in different design stages. Furthermore, every model expressed in MoSaRT language can be checked by several structural rules implemented in OCL (Object Constraint Language) [11] ensuring the vivacity, the safety and architectural correctness.

Figure 2 shows a hardware architecture of a real-time system. The system is composed of two nodes communicating through a CAN (Controller Area Network) network. The first node is uniprocessor, and the second node is a multi-core processor containing four cores. The software architecture

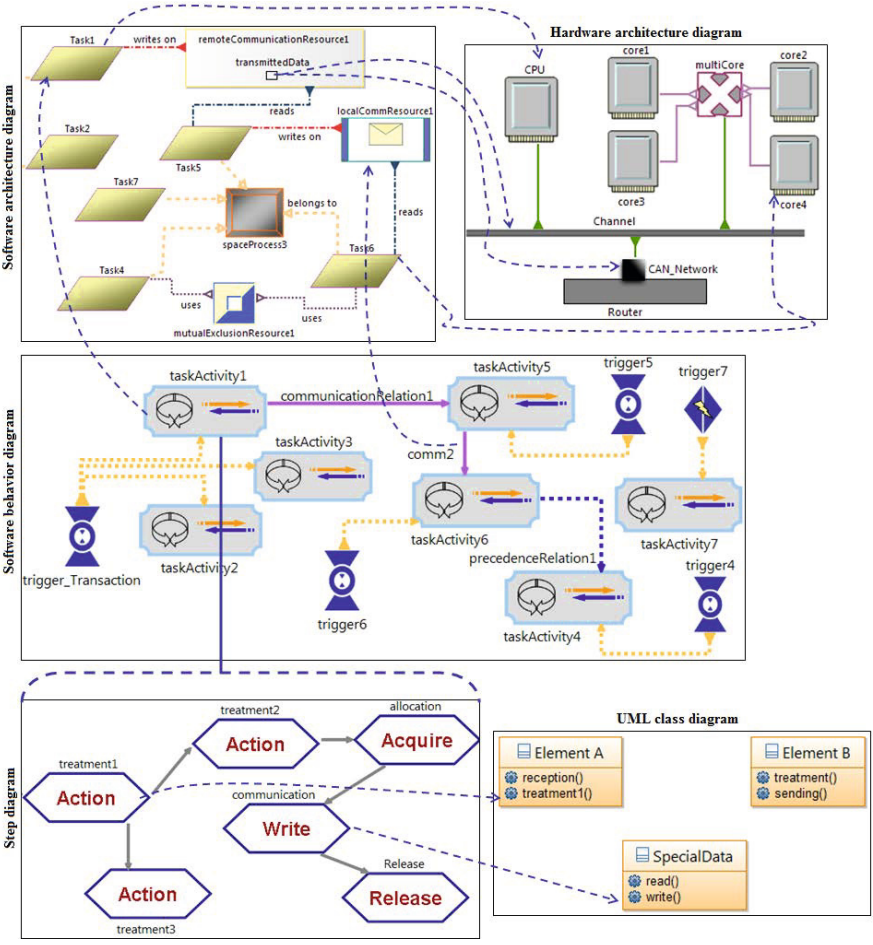

Fig. 2 Different steps of the identification process

diagram of Figure 2 represents a software architecture model that contains seven tasks managed by three schedulers, and two of them are hierarchical. It also contains two interaction resources shared by three tasks (mutual exclusion resource and box communication resource), and a remote communication resource allowing to transmit data between two tasks. The software architecture model can be mapped to different behavioral models. The one shown in Figure 2 represents the global behavior of the system and requires a root trigger meaning the timing reference of the remainder triggers. The model contains several task activities. A task activity can be triggered by its own trigger or it can be triggered by another task activity. The precedence relationship represents a synchronization between the task activities. The existence of a communication relationship between two task activities implies the existence of a shared communication resource in the software architecture model. Through the behavioral model, the content of every task activity can be defined thanks to the step diagram. Every step describes the elementary actions of the task activity like the read action, the release action, etc. The importance of step elements is also their capability to allocate the operational side of a real time-system (hardware, software architecture and behavioral models) to the functional side (e.g. UML models).

5 MoSaRT Analysis Repository

We have noticed the absence of an instrumented method guiding the designers to the best model and tests for their systems. Moreover, the passage from the system modeling to the system analysis requires dual skills, in order (i) to identify the appropriate analysis situation of the system design and (ii) to find the suitable analysis tests. We note, $Ar = <\mathcal{R}, \mathcal{X}, \mathcal{G}, \mathcal{T}, E>$ is the MoSaRT analysis repository, where:

- $\mathcal{X}$ is a set of real-time contexts. Every real-time context represents a set of specific assumptions, where each assumption is related to the software architecture, the timing behavior or the hardware architecture. The real-time context represents the analysis situation to which the system design corresponds. In other words, thanks to the real-time context we can know the analysis model that matches the system design.
- $\mathcal{G}$ is a set of generalization relationships between some real-time contexts. The generalization relationship is an order-relation treated in the real-time scheduling theory. A real-time context "a" is a generalization of the real-time context "b", if the behavior of "a" includes all possible behaviors of "b".
- $\mathcal{T}$ is a set of analysis tests. Every analysis test is based on a real-time context. When it is applied to a system, the result provided by the test is correct if the system respects all the context assumptions.
- E is a set of analysis tools. The analysis tool is an engine proposing an independent functionality inside an analysis framework (like MAST or

Cheddar). The same analysis functionality inside another analysis framework is considered as another engine.
– It is common to find several real-time context characterized by the same subset of assumptions, or the opposite subset of assumptions. So, for factoring the number of assumptions and to guarantee a good scalability of the analysis repository, we suggest that the analysis repository contains also a set of identification rules $\mathcal{R}$. They will help for identifying correctly the closest real-time context matching the design model which requests analysis. Every identification rule is characterized by a formal expression as an OCL constraint, this latter depends on MoSaRT design language.

5.1 Instantiation of the MoSaRT Analysis Repository

The metamodel of the MoSaRT analysis repository (see Figure 3) is dedicated to be instantiated by schedulability-aware theorists/analysts in order to obtain analysis decision supports. Thus, the designer's orientation will be based on the richness and the correctness of the analysis decision support. This latter may be fed and enriched by theorists/analysts (i) to compare their results with existing ones (already stored in the decision support), (ii) and to facilitate the use of their works (and their prototypes) by designers.

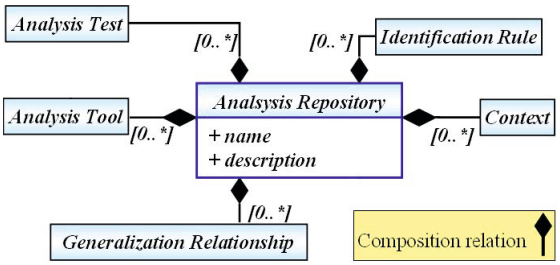

Fig. 3 Analysis repository Metamodel

In order to create a new repository instance from scratch, we start by instantiating the `IdentificationRule` to get a set of identification rules. Next, we instantiate the `Context` to create a real-time context based on the existing identification rules. Furthermore, we add to the repository instance some tests and analysis tools corresponding to the created real-time context.

First of all, we provide an analysis repository containing a real-time context corresponding to the Liu and Layland model [8]. This context is based on several identification rules (some of them are shown in Figure 4). Each rule is mapped to a formal expression implemented as an OCL constraint related to the design language of the system that needs analysis (i.e. the MoSaRT design language). Figure 4) shows the formal expression of the identification

rule called "`UniprocessorArchitecture`". Moreover, we have chosen the response time analysis test presented in [6] as an analysis test for the context corresponding to the analysis model of [8]. This test is implemented by several tools like Rt-Druid [4].

Furthermore, we have added the real-time context of the transaction model [16]. This latter represents a "generalization" of the periodic model which had been previously created in the analysis repository. A second response time analysis test is added to the repository. It is devoted to analyze the transaction model. We mention MAST as a provider of this second test. Besides, we have embodied the generalization relationship, and connect it with a transformation program enabling the transition from the periodic model to the transaction model (when it is possible). The transformation program is done with ATL (Atlas Transformation Language) [7]. It represents an endogenous transformation (i.e. a transformation from MoSaRT design language to MoSaRT design language).

6 MoSaRT Usage Scenarios

6.1 The Back-end of the MoSaRT Framework

Figure 5 indicates the back-end and the front-end of the MoSaRT framework. MoSaRT back-end provides to analysts various capabilities. So, one may propose a new analysis repository (related to a specific company/laboratory) by instantiating the analysis repository model (Action (1) of Figure 5). In

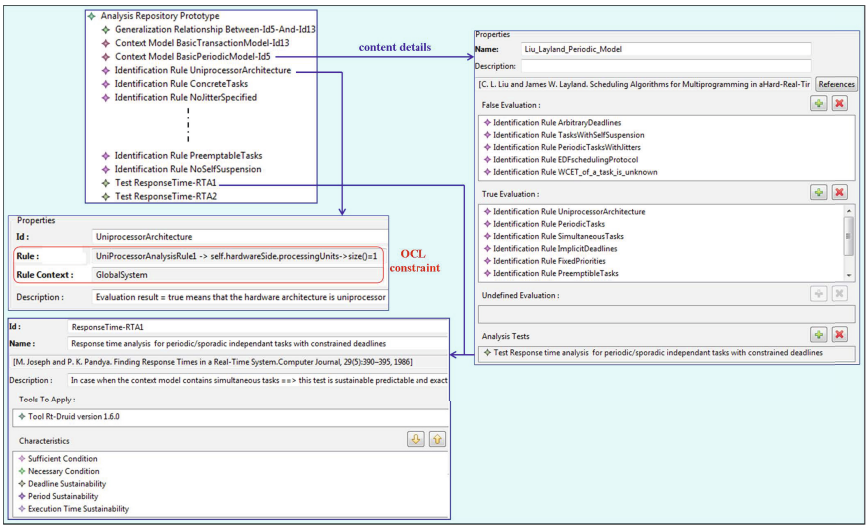

Fig. 4 Part of an analysis repository highlighting the details of a context

this case, one should define at least the contexts and tests to guarantee a minimum usability of the repository. To rise the usability and to get a full coherence of an analysis repository, this latter can be enriched progressively by adding new contexts, new tests, new tools, etc (Action (2) of Figure 5). Therefore, every real-time context already existing in such a repository can be refined by specifying more accurate identification rules, more characteristics of the analysis tests, and automatizing the transformation to analysis tools, etc.(Action (3) of Figure 3). Once an analysis repository becomes ready for use, it can be shared in order to be used by designers (Action (4) of Figure 5).

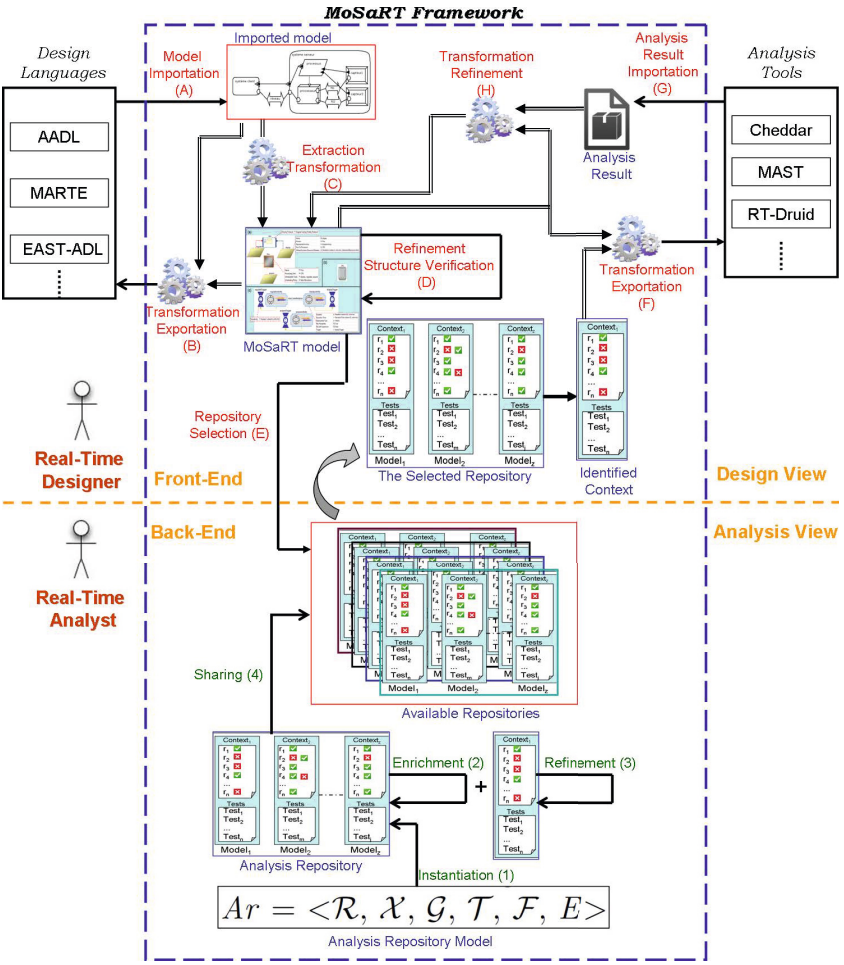

Fig. 5 Some of the relevant utilization scenarios related to the MoSaRT Framework

6.2 The Front-end of the MoSaRT Framework

The MoSaRT front-end is totally related to the MoSaRT design language. Figure 5 gives an overview of some capabilities. The design models expressed in a standard language like MARTE can be imported (Action A of Figure 5). Then, designers can use the transformation process offered by MoSaRT framework in order to transform their models to MoSaRT design language. The transformation process from a standard language to MoSaRT language is based on the extraction of timing details (Action C of Figure 5). While the MoSaRT framework gives the possibility to use its modeling environment, actions A and C are not mandatory. Once designers obtain models expressed in MoSaRT language, they can refine them and check the correctness of their structure (Action D of Figure 5). Hence, the analysis stage starts when designers select an available MoSaRT analysis repository (Action E of Figure 5). When identifying the corresponding real-time context (if the repository is rich enough), a customizable transformation to analysis tools can be provided (Action F of Figure 5). Once getting the analysis result, MoSaRT framework gives the possibility to import the tool output files (Action G of Figure 5). Due to technical transformation reasons, the Action H of Figure 5 requires both the original MoSaRT model and the analysis result file. The refinement and the analysis actions can be repeated many times until getting accurate models. The transformation of the MoSaRT analyzed model to an external design language can be done only if the model was imported (Action B of Figure 5).

7 Conclusion

By proposing the MoSaRT framework, our objective is to benefit from the analyst's skills and designer's skills in order to unify their efforts, then to avoid wrong design choices at an early design phase. The framework is based on the MoSaRT design language and the MoSaRT analysis repository, and presents a helpful modeling support for both designers and analysts. Since we have tried to facilitate the use of concepts based on theoretical studies and dedicated to a concrete industrial utilization, we may consider the MoSaRT framework as a tool for technology transfer.

We are working to enrich the MoSaRT Framework, in particular, in case of the non schedulability of the system, currently the MoSaRT framework proposes only a dimensioning analysis if it exists (it depends on the system context). However, some works like Optimum (applied to UML-MARTE) proposes to change down-right the system architecture if the functional model exists. Such works can be connected to the framework in order to be called after the restitution step (e.g. obtaining the analysis result). It will be also helpful to generate a design expressed in MoSaRT language by choosing a priori the real-time context to which the design corresponds. In this case,

the real-time context is used as a design pattern. For example, generating a design model respecting the pattern "Ravenscar Profile" which is widely used in industry can be very interesting.

Acknowledgment. We thank Obeo[1] that has provided us the Sirius product which is a new Eclipse project (http://www.eclipse.org/sirius). We used this tool to implement the concrete graphical syntax of the MoSaRT design language. It is particularly adapted to define a DSL (Domain Specific Language) that needs graphical representations to better elaborate and analyze a system and improve the communication with other team members, partners or custom. All shape characteristics and behaviors can be easily configured with a minimum technical knowledge. This description is dynamically interpreted to materialize the workbench within the Eclipse IDE. No code generation is involved, the specifier of the workbench can have instant feedback while adapting the description. Once completed, the modeling workbench can be deployed as a standard Eclipse plugin. Thanks to this short feedback loop a workbench or its specialization can be created in a matter of hours.

References

1. W. SAE AADL. The SAE Architecture Analysis & Design Language Standard, vol. 2009 (2009)
2. Burns, A., Wellings, A.: Real-Time Systems and Programming Languages: Ada, Real-Time Java and C/Real-Time POSIX, 4th edn. Addison Wesley (2009)
3. Davis, R.I., Burns, A.: A survey of hard real-time scheduling for multiprocessor systems. ACM Comput. Surv. 43(4), 35 (2011)
4. Gai, P., Natale, M.D., Serreli, N., Palopoli, L., Ferrari, A.: Adding timing analysis to functional design to predict implementation errors. SAE Technical Paper 2007-01-1272 (2007)
5. Henia, R., Hamann, A., Jersak, M., Racu, R., Richter, K., Ernst, R.: System level performance analysis–the symta/s approach. IEE Proceedings-Computers and Digital Techniques 152(2), 148–166 (2005)
6. Joseph, M., Pandya, P.K.: Finding response times in a real-time system. Computer Journal 29(5), 390–395 (1986)
7. Jouault, F., Kurtev, I.: Transforming models with ATL. In: Bruel, J.-M. (ed.) MoDELS 2005. LNCS, vol. 3844, pp. 128–138. Springer, Heidelberg (2006)
8. Liu, C.L., Layland, J.W.: Scheduling algorithms for multiprogramming in a hard-real-time environment. J. ACM 20(1), 46–61 (1973)
9. Mraidha, C., Tucci-Piergiovanni, S., Gerard, S.: Optimum: a marte-based methodology for schedulability analysis at early design stages. ACM SIGSOFT Software Engineering Notes 36, 1–8 (2011)
10. Society of Automotive Engineers (SAE). The SAE architecture analysis & design language standard, `http://www.aadl.info` (last access: April 15, 2014)
11. Object, O.: constraint language, omg available specification, version 2.0 (2006), `http://www.omg.org/spec/OCL/2.0/`
12. OMG. Uml profile for marte: Modeling and analysis of real-time embedded systems (2009), `http://www.omgmarte.org`

[1] `www.obeo.fr`

13. Ouhammou, Y., Grolleau, E., Hugues, J.: Mapping aadl models to a repository of multiple schedulability analysis techniques. In: IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC), p. 8 (2013)
14. Ouhammou, Y., Grolleau, E., Richard, M., Richard, P.: Model driven timing analysis for real-time systems. In: IEEE International Conference on Embedded Software and Systems (ICESS), pp. 1458–1465 (2012)
15. Ouhammou, Y., Grolleau, E., Richard, M., Richard, P.: Reducing the gap between design and scheduling. In: Real-Time and Network Systems (RTNS), pp. 21–30. ACM (2012)
16. Palencia, J.C., González Harbour, M.: Schedulability analysis for tasks with static and dynamic offsets. In: IEEE Real-Time Systems Symposium (RTSS), pp. 26–37 (1998)
17. Quadri, I.R., Brosse, E., Gray, I., Matragkas, N.D., Indrusiak, L.S., Rossi, M., Bagnato, A., Sadovykh, A.: Mades fp7 eu project: Effective high level sysml/-marte methodology for real-time and embedded avionics systems. In: International Workshop on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC), pp. 1–8 (2012)
18. Sha, L., Abdelzaher, T., AArzén, K.-E., Cervin, A., Baker, T., Burns, A., Buttazzo, G., Caccamo, M., Lehoczky, J., Mok, A.K.: Real time scheduling theory: A historical perspective. Real-Time Systems 28(2-3), 101–155 (2004)
19. Singhoff, F., Plantec, A., Dissaux, P., Legrand, J.: Investigating the usability of real-time scheduling theory with the cheddar project. Real-Time Systems 43(3), 259–295 (2009)
20. Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: EMF: Eclipse Modeling Framework. Pearson Education (2008)