Formal Behavioral Modeling of Real-time Operating Systems

Cédrick Lelionnais[1], Matthias Brun[1], Jérôme Delatour[1], Olivier H. Roux[2] and Charlotte Seidner[2]
[1]*ESEO-TRAME, 4 Rue Merlet de la Boulaye, B.P.30926, 49009 Angers cedex 01, France*
[2]*LUNAM Université, IRCCyN-Ecole Centrale de Nantes, 1 Rue de la Noe, B.P.92101, 44321 Nantes cedex 03, France*

Keywords: Model Driven Engineering, Real-time Operating Systems, Behavioral Modeling, Formalization, Time Petri Nets, Application Deployment.

Abstract: Faced with the growing problems of complexity, heterogeneity and upgradability of Real-Time Embedded Systems (RTESs), model-based frameworks dedicated to the application deployments facilitate the design and the development of such systems. Within these frameworks, taking into account the Real-Time Operating Systems (RTOSs) has become essential. These frameworks include transformation tools able to generate a code that is portable to the specified RTOS. Moreover, certain tools can generate formal models that are used for the verification and validation of the RTESs. However, the RTOSs technological concepts are considered in an implicit way, which involves a lack of genericity of the transformations. Some works have focused on the explicit description of the RTOSs. Such a description offers the possibility to take into account a model entirely dedicated to a targeted RTOS as a parameter of the transformation. Nevertheless, this method does not allow to verify the expected properties on the application, since the RTOSs behavior is not observable. The methodology presented in this paper tends to explicitly consider the formal description of the RTOSs behavior during an application deployment. This approach aims both at making each transformation generic and at verifying the deployment correctness.

1 INTRODUCTION

Nowadays, Real-Time Embedded Systems (RTESs) increasingly surround us in various domains (aircrafts, automotive, cell phones, robotics...). RTES engineers are confronted with the challenge of developing more complex, higher quality systems, with shorter development cycles at lower costs. Within this context, reuse, maintainability and portability have become major issues in RTES design processes.

From this point of view, some frameworks have been developed with the aim of generating code. Starting from a detailed model written in such a language as AADL (Society of Automotive Engineer (SAE), 2004) or UML (Object Management Group (OMG), 2007b), the specific code for the deployment of an application on a Real-Time Operating System (RTOS) is then generated. These code generating frameworks are generally used after some Verification and Validation (V&V) activities have been performed on the models. In this way, some research works have dealt with the transformation of those models into formal models, using such formalisms as Time Petri Nets (TPNs) or Finite State Machines (FSMs)...) as presented for instance in (Renault et al., 2009) (Berthomieu et al., 2010).

However, the correctness of the application deployed on the RTOS is not verified during these V&V activities. It is therefore difficult to ensure that the generated code effectively corresponds to what was designed. Moreover the consideration of the RTOSs is often embeded within the processus of generation (or seen in simplified terms). This approach is then far from flexible since each translation must be adapted in accordance with the targeted RTOS. Due to the significant number of existing RTOSs, this results in having to maintain a large amount of translations.

In consequence, this paper relates to the following issues. Which methodology should be adopted in order to both deploy an application on a targeted RTOS and check the correctness of such a deployment? Is it possible to consider any RTOS during the deployment? (and if so, how?) How can the behavior of any RTOS be described within such frameworks? To bring an answer to these questions, a method has been experimented to take these factors into account within a single framework. The basic idea is to add the formal description of any RTOS. Each description can

Lelionnais C., Brun M., Delatour J., H. Roux O. and Seidner C..
Formal Behavioral Modeling of Real-time Operating Systems.
DOI: 10.5220/0004156204070414
In *Proceedings of the 14th International Conference on Enterprise Information Systems* (MDDIS-2012), pages 407-414
ISBN: 978-989-8565-11-2

then be considered as a parameter of the transformation, so as to keep the framework generic. Lastly, this formalization should let V&V tools able to verify the correctness of such deployments.

This paper is divided into the following sections. Section 2 presents some works in conjunction with the issues described above. As a result of that research, the approach adopted is developed in Section 3. It has proven the need to consider the behavior of RTOSs during a deployment of an application. Then Section 4 presents the strategy proposed to formally take into account the behavioral description of RTOSs. Consequently the benefits and the limits of the strategy are showed in Section 5 and discussed in Section 6. Finally we conclude in Section 7.

2 RELATED WORK

2.1 Code Generating Frameworks

Code generating frameworks are used in a context of application deployment. Code generation is the process of transforming code from one representation to another one. Often, this is from a higher level, such as an UML diagram modeling an application, to a lower level, such as a C code program that is portable to an execution software platform.

From a real-time point of view, code generating frameworks are used in a context of application deployment to a specific RTOS. The information concerning RTOSs are given in a program which implements the deployment. For instance, Ocarina described the information of a middleware with Ada code in its Gaia generator (Vergnaud and Zalila, 2006).

Similarly, the Real-Time Workshop (RTW) tool (The MathWorks, 2007) uses templates catching both technological concepts of the RTOS and programming language information. These templates structure the source code to generate.

As a result, these frameworks involve an implicit description of the RTOSs during the processus of generation. This raises the problem of genericity of these frameworks since each deployment corresponds to a unique code generator.

2.2 RTOSs Modeling

In order to describe the Operating Sytems (OSs) concepts, some model-based frameworks have been developed. Some works have focused on both structural and behavioral aspects of the OSs. Revolving around these two axes, the description of the OSs is made explicit and observable through formal models. Explicit means that such a description is entirely dedicated to the representation of the OS.

Metropolis (Team, 2004) uses formal languages such as the Linear Temporal Logic. It offers the possibility to verify some properties of an application deployed on a wide range of OSs.

Generic Modeling Environment (GME) (Davis, 2003) contributes to the description of OSs which are not necessarely real-time and analysis platforms. GME proposes assistance tools for the migration from one OS to another. In addition, this environment offers the possibility to insert formal metamodels. The descriptions thereby made can be used by V&V tools.

Nevertheless, the genericity of the modeling languages used within these frameworks does not make the modeling of platforms easier dedicated to a particular domain, such as the real-time domain, any easier.

More specifically to the RTESs, Ptolemy (Lee, 2003) is a framework which describes execution models. Most of these models support the actor-oriented design focused on the concurrency and the communication between the components of a RTES. However, the concepts of structural representation from this approach are only intended for the applications modeling, but not for RTESs themselves.

In spite of the fact that the OSs description is explicit, the works outlined above are less in line with the RTESs. Furthermore, the separation of the concerns does not clearly appear. This impacts on the intervention of each domain specialist such as the application or the OSs technology. The lack of genericity previously noted arises once again.

Other researches have led to consider the description of RTOS during the application deployment. This approach relies on a Model Driven Engineering (MDE) context. MDE promotes the distinction of platform-specific modeling artefacts from those that are platform-independent. The platform (RTOS) model is considered as a parameter of this deployment. Each model is described by a modeling language. Concerning the RTOS modeling, we can note the Software Resource Modeling (SRM) (Thomas et al., 2008) package which is a MARTE's UML profile (Object Management Group (OMG), 2007a). With SRM, RTOSs can be modeled using stereotyped concepts from the real-time software domain.

For another example, RTEPML (Real-Time Embedded Platform Modeling Language) (Brun and Delatour, 2011) was developped with the aim of defining concepts dedicated to the real-time domain for modeling RTOSs.

Contrary to the first noted frameworks, SRM

and RTEPML better separate the concerns previously highlighted (RTOSs structure, transformations, deployments choice...). Nevertheless, only the structural aspect is taken into account with these modeling languages. The behavior resulting from the concurrent activities of the RTESs components is not observable in the deployed application models. As a result, no V&V activities can be applied to check the correctness of such a deployment.

3 APPROACH ADOPTED

The previous section showed the interest to well distinguish the concerns for the intervention of each specialist for real-time applications deployment. In addition, it has been raised to explicitly consider the description of the RTOSs behavior within a code generating framework. Our approach relies therefore on the extension of a model-based framework to formally observe the behavior of a deployed application on a RTOS. This should add other concerns such as the RTOSs behavior and some V&V activities on the deployment at design phase.

As a result, we have chosen the RTEPML framework as it has been developed within our team. In addition this modeling language already contributes to the explicit description of the RTOSs. However, the behavioral aspect of RTOSs can not be modeled with RTEPML. Therefore modeling behavior with RTEPML means extending this language. That is why we have sought to enrich it in order to define behavioral concepts of RTOSs.

3.1 RTEPML with MDE

Based on the MDE initiative, the application deployment emanates from semi-generic transformations of models. MDE is based on an architecture (Model Driven Architecture (MDA)) (Object Management Group (OMG), 2001) (see Figure 1) involving a design process where a Platform-Independent Model (PIM) (description of the application without considering the platform) is transformed into a Platform-Specific Model (PSM) (description of the application deployed on the platform), according to a Platform Description Model (PDM) (description of the platform). Each model conforms to a modeling language, also named metamodel.

In accordance with Figure 1, RTEPML is a metamodel which has been built to model PDMs. The concepts of the targeted RTOS are then defined by a PDM. So far, the modeling with RTEPML goes through the structural characterizations (resources

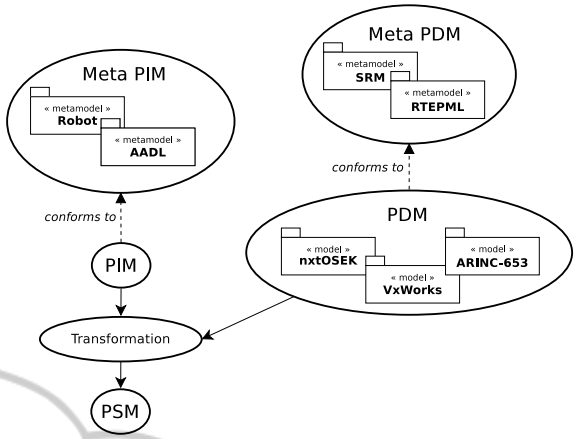

Figure 1: MDA principle.

and services) of RTOSs. The resources represent the concepts offered by the platform. For instance, in a RTOS, the tasks, the semaphores, the mailboxes are considered as resources. The services are offered by the resources and caught at API's level (Application Programming Interface). For example, services associated with semaphores are the *take* and *release* primitives.

Consequently, in conjunction with this approach, the deployment consists both in instantiating RTOS resources and in using RTOS call services for implementing the application. Then the generated PSM is provided for a code generation process. Finally, it is worth noting that the transformation serving to the deployment is generic thanks to the explicit consideration of the RTOSs.

3.2 Illustration around Semaphore Sharing

We have mentioned the idea to extend RTEPML at the beginning of this section. This extension should enable to consider the RTOS behavior during an application deployment. To illustrate this behavioral lack, a deployment is presented Figure 3 [1] based on an example of Robot application (see Figure 2). Obviously several concepts instantiations of the considered RTOS are possible following the application. Accordingly, this illustration is focused on a particular case : the semaphore sharing.

In this way, the PIM representing the robot application combines two periodic activities (a *Driver* activity for driving the robot and a *Vigil* activity for avoiding barriers). On the one hand, the *Driver* in-

[1]The languages used in Figures 2 to 5 come from the graphic description of each metamodel (their concrete syntax).

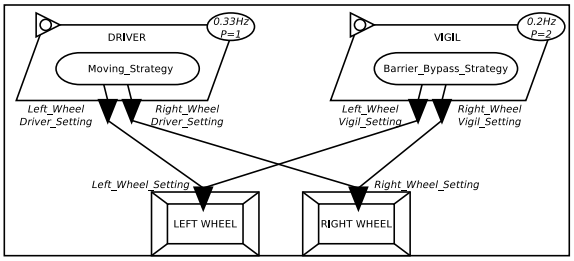

Figure 2: Model of a robot application.

vokes a *Moving Strategy* program. On the other hand, the *Vigil* invokes a *Barrier Bypass Strategy* program. Then two wheels allow the robot to move thanks to two settings updated according to the activities. Finally each periodic activity owns a priority according to its role.

The PDM described by RTEPML for this deployment is a model of the nxtOSEK's RTOS (nxtOSEK, 2009). nxtOSEK is based on OSEK/VDX standard (OSEK/VDX Group, 2005) and is widely used in LEGO MINDSTORMS NXT educational projects. According to our needs, some resources have been defined to be instantiated with regards to the application. The *Task* concept is instantiated to represent each activity, the *Function* concept to execute each program and the *Resource* (semaphore concept in nxtOSEK) concept to protect each wheel setting updating.

The PSM results in a model of the application deployed on the nxtOSEK platform. However the concurrent activities which are governed by both application and technological concepts do not concretely appear in the final model. For instance we do not know whether each *Resource* (protection for updating each wheel setting) is available or not (empty or full). Moreover the state in which each *Task* (the *Driver_T* and the *Vigil_T*) is set cannot be clearly distinguished. Finally the impact of the properties such as the *priority* and the *period* of the tasks is not directly verifiable.

Consequently this illustration shows how difficult it is to interpret the behavior of the resources instantiated. The behavioral aspect of the generated model is not observable. In this approach, it is therefore difficult (if not impossible) to undertake some V&V activities to verify the correctness of the deployment. In order to be able to check the properties on the deployed application, formal behavioral aspects have to be introduced within the description of the software platform.

4 STRATEGY OF BEHAVIORAL FORMALIZATION

In accordance to the previous conclusion, RTEPML has been enriched to bring a behavioral aspect in the RTOSs description. On this point, we have extended the abstract syntax of RTEPML in which executive concepts have been defined to explicitly describe any platform.

Once RTEPML extended (newly called RTEPML_BEHAVIOR), the implementation of a transformation process could have been put in place to make the model of the deployed application behavioral. To meet the requirements of V&V activities, the formalization of the PSM is necessary. RTEPML_BEHAVIOR has therefore been designed to take into account any formal language. This solution has been thought in order to make the transformation independant of other formal languages.

4.1 Behavioral Consideration

Taking into account the RTOSs behavior necessitates the identification of the behavior of each resource (scheduling task, state of a semaphore...). The same applies to the behavioral description of each service offered by resources (task terminating, semaphore acquisition...). Lastly, each behavioral impact associated with each resource property (task priority, task period, semaphore tokens number...) should also be treated during the deployment.

To keep on the previous focus, an example of behavioral consideration around the semaphore is depicted Figure 4. As explained earlier, some concepts have been added in RTEPML_BEHAVIOR to represent the behavior of the RTOSs resources. Thus, the behavioral prototype of the semaphore concept, called *ResourceBehavioralPrototype*, has been defined on the Behavioral PDM. As a reminder, *Resource* (on Behavioral PDM) represents the semaphore concept of nxtOSEK RTOS, whereas "Resource" is the RTEPML_BEHAVIOR metaconcept allowing to describe any RTOS resource. In other words, *Resource* conforms to MutualExclusionResource.

The formalism used to express the behavioral prototype is Time Petri Net (TPN) (see (Boyer and Roux, 2008) for a survey). TPNs have been prefered because of their expressiveness. This formalism is indeed adapted to the description of concurrency activities such as resources schedulability, resource sharing, call services sequence, time constraints... Furthermore, TPNs propose known time extensions (such as stopwatch TPNs) which allow to model real-time

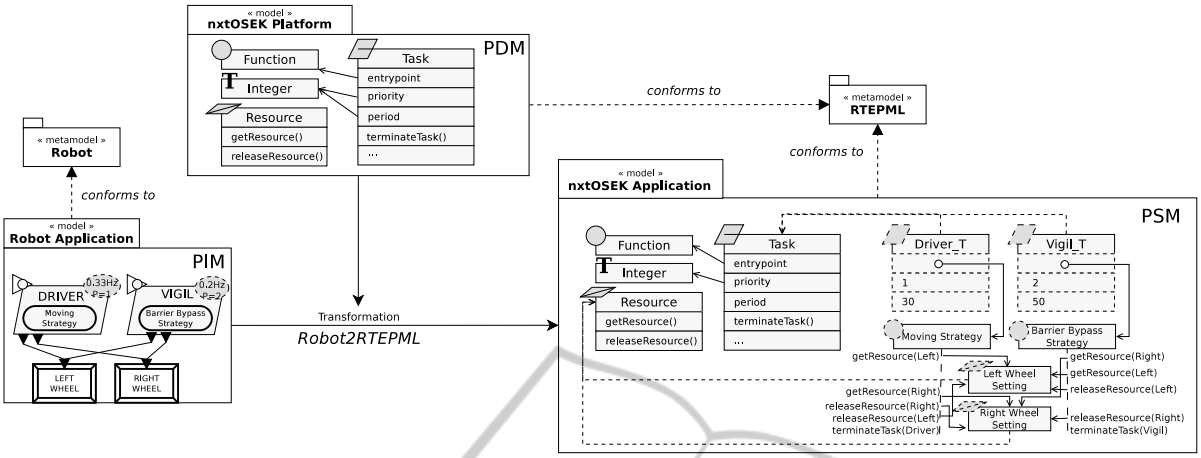

Figure 3: Robot application deployment.

schedulings implemented in V&V tools like Romeo (Gardey et al., 2005) (Lime et al., 2009) or Tina (Berthomieu et al., 2004).

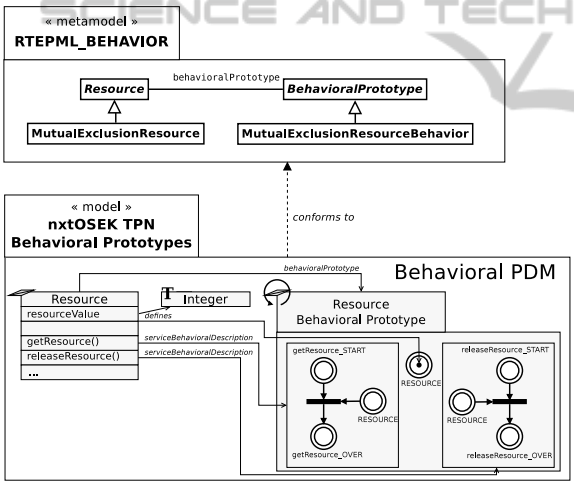

Figure 4: MER behavior.

As can be seen on Figure 4, a behavioral prototype can be composed of several behavior elements. Indeed, a RTOS resource includes services and properties. The appearance of roles (defined at metamodel level) within the Behavioral PDM, enables to precise certain behavior elements. In the nxtOSEK semaphore case, *serviceBehavioralDescription* role comes to describe the services behavior of *getResource* and *releaseResource*, whereas *defines* role comes to define the value of the semaphore.

4.2 Deployment with RTEPML_BEHAVIOR

The behavioral description of RTOSs has been made possible with RTEPML_BEHAVIOR. Moreover the possibility to define formally the behavior elements, contributes to the formalisation of the application deployment. This important point should enable to verify its correctness before the code generation.

In accordance with the adopted approach, a transformation has been developed to consider explicitly a behavioral PDM of the targeted RTOS. We have taken the same robot application from the previous section to illustrate an example of formalisation of the deployment (see Figure 5). For the sake of simplicity and clarity, we have reduced the Robot PIM to the *Driver_T* instance representing the periodic *Driver* activity. *Driver_T* has an entry point on the *Moving Strategy* in which only a service is called for terminating itself.

The involved prototypes in the transformation are those of the resources for the needs of the application. Once the prototypes are detected, the latter are cloned according to each instance of the application. It may be observed that the cloned prototypes corresponding to the behavior of the *Driver_T* and the *Moving Strategy* (including the service of terminating) have been composed through this example.

As a result, the TPN Behavioral PSM achieved is more expressive than the previous one. Its expressiveness helps us to easily see its liveness (the different states of *Driver_T*, its periodicity...).

The composition of the building blocks represented by cloned behavioral prototypes (semaphore, task, communication, priority...) make up the generated model. In TPNs case, the composition is car-

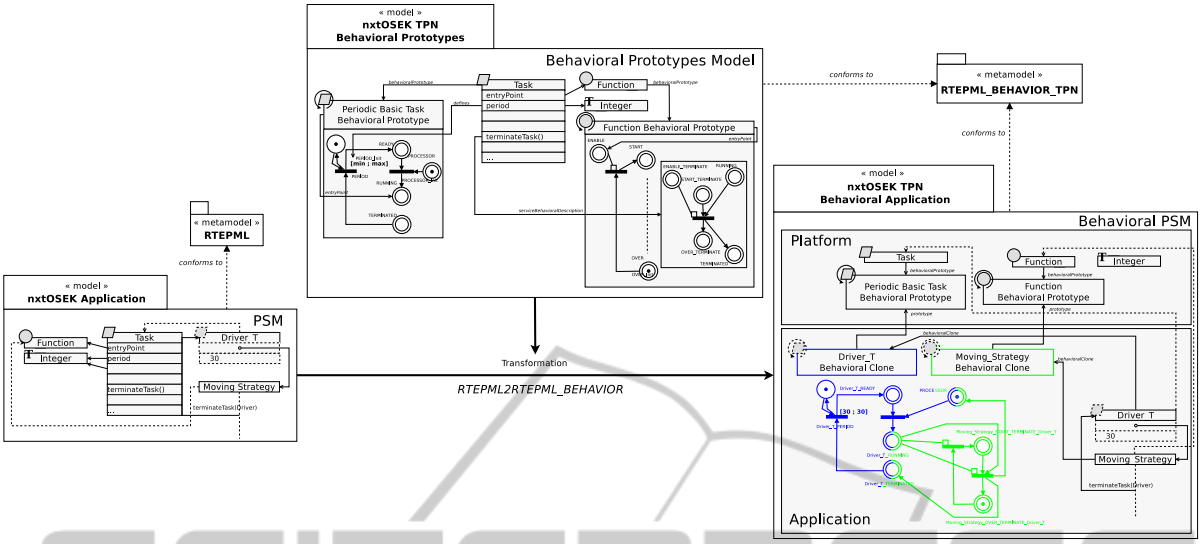

Figure 5: Formalisation of robot application in TPN.

ried out by merging places interfacing all the blocks. This presents some advantages. The semantics of the places merging is obvious, while the one of the transitions merging is ambiguous. Indeed, the fact of assigning different time constraints on mergeable transitions makes their merging difficult. In addition, the places merging preserves the blocks behavior, which enables to validate more easily the building.

In this paper, the purpose is neither to describe exhaustively the modeling of the behavioral prototypes and the composition of their clones, nor to prove that it is correct. The aim is only to present the approach and the methodology which allows it. That is why no proof of the building validity has been given. Notwithstanding, this proof is greatly simplified by our composition approach.

5 BENEFITS OF THE STRATEGY

The main benefits of our methodology are both the formalization and the moving of the application deployment into the design phase. This should allow to ensure its correctness before generating code. Basing our approach on MDE, a formalization of the deployed application model can be carried out. V&V activities will become applicable thanks to the use of model-checking tools.

A second benefit is the separation of the skills highlighted in the second section. The distinction of modeling artefacts allows both RTOSs behavior and formal verification experts to interact on the application deployment (see Figure 6). Thanks to RTEPML_BEHAVIOR, the specialist of the RTOSs behavior can be assisted by the one more highly skilled in the formal verification. The latter one can also interpret the most adapted formalism following the case.

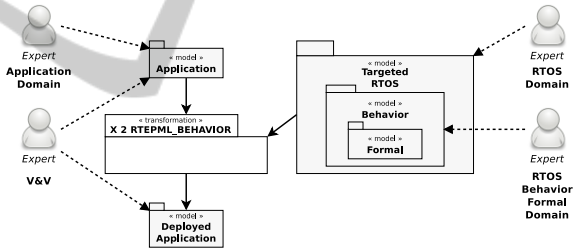

Figure 6: Intervention of specialists.

To put our strategy into practice, we have chosen to verify a necessary and well-known condition in the RTESs domain: the absence of deadlock. Once the PSM generated in TPN, the formal model can be processed by ROMEO (Gardey et al., 2005). ROMEO has been developed within the Real-Time team at IRCCyN [2] lab. One of the reasons for choosing ROMEO is that this tool is dedicated to Transition-TPN (T-TPN) verification, i.e. the time is only assigned to the transitions. Algorithms were proposed for applying verification on T-TPN extended to scheduling.

Despite the combinatorial explosion risk of the generated TPN, ROMEO has given a deadlock outcome. This diagnosis could seem foreseeable from the specification since the wheels settings updating is protected by semaphores. Notwithstanding, this verification helps us to evaluate the right protocol to use

[2] http://romeo.rts-software.org

for this application. Indeed, the Priority Ceiling Protocol (PCP) (OSEK/VDX Group, 2005) should have been chosen to remedy this situation during the deployment.

We have exposed the deadlock case but other behavioral properties could have been tested. For instance we could deal with liveness properties such as termination of actions, occurency of expected events... But we could also handle time constraints such as Worst-Case Execution Time (WCET), respect of a deadline...

6 LIMITS

This experimentation has showed the feasability of generating a formal model to observe the behavior of the application deployed on a RTOS. Nonetheless, the behavior of some of concepts (the notification resources such as the events or the communication resources such as the messages...) are not yet taken into account within RTEPML_BEHAVIOR. As a result, an optimization of the latter modeling language must be widened in order to cover such concepts.

Another limit raised is the interpretation of other formalisms to model the behavior during the deployment. Only TPNs have been presented through our strategy but the implementation of other formal languages deserves to be examined.

Lastly, a deadlock has been detected during the deployment. It has been deduced that the deployment was not correct. That being so, this decision does not affect the correspondence between the application and the targeted RTOS. Indeed, a protocol changement (Priority Ceiling Protocol (nxtOSEK, 2009) (OSEK/VDX Group, 2005)) is just sufficient to prove it. This leads us to say that the framework does should integrate decision support tools for guiding the experts involved in the RTOS choice.

7 CONCLUSIONS

Through this study, a methodology has been presented to consider the behavior of a real-time execution software platform (RTOS) during an application deployment. The deployment which enables to implement an application on a specific RTOS, has been moved at design phase. The purpose of this choice is to enable the different specialists to intervern more easily on the deployment following its domain. Indeed, thanks to the modeling with RTEPML and the MDE approach, the separation of concerns (application and RTOS) has been made explicit.

The description of the RTOSs behavior has been made feasible by enriching RTEPML (RTEPML_BEHAVIOR). In addition, each behavioral model can be formally described. A transformation has been developed to generate a formal model of the deployed application. The formalization has the advantage of applying V&V activities for checking the correctness of the deployment, even before generating the code useful to the implementation of the application.

However, certain limits of our methodology have arisen. Future prospects are the subject of our next works to improve and to check the feasibility of such a strategy. In accordance to the previous section, the behavioral concepts missing in RTEPML_BEHAVIOR will have to be created. Then, to ensure the genericity of the formalisation, other languages will have to be interpreted. Finally, a large number of formal proofs will have to be written within our framework. This should contribute to the verification of necessary and sufficient conditions for validating such deployments.

REFERENCES

Berthomieu, B., Bodeveix, J.-P., Dal Zilio, S., Dissaux, P., Filali, M., Gaufillet, P., Heim, S., and Vernadat, F. (2010). Formal Verification of AADL models with Fiacre and Tina. In *ERTSS 2010 - Embedded Real-Time Software and Systems*, pages 1–9, TOULOUSE (31000), France. 9 pages DGE Topcased.

Berthomieu, B., Ribet, P.-O., and Vernadat, F. (2004). The tool tina – construction of abstract state spaces for Petri nets and time Petri nets. *International Journal of Production Research*, 42(4).

Boyer, M. and Roux, O. H. (2008). On the compared expressiveness of arc, place and transition time Petri nets. *Fundamenta Informaticae*, 88(3):225–249.

Brun, M. and Delatour, J. (2011). Contribution on the software execution platform integration during an application deployment process. *First Topcased Day*.

Davis, J. (2003). GME: the Generic Modeling Environment. In *OOPSLA '03: Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 82–83, New York, NY, USA. ACM.

Gardey, G., Lime, D., Magnin, M., and Roux, O. H. (2005). Roméo: A tool for analyzing time Petri nets. In *17th International Conference on Computer Aided Verification (CAV'05)*, volume 3576 of *Lecture Notes in Computer Science*, pages 418–423, Edinburgh, Scotland, UK. Springer.

Lee, E. A. (2003). Overview of the Ptolemy project. Technical Report UCB/ERL M03/25, EECS Department, University of California, Berkeley.

Lime, D., Roux, O. H., Seidner, C., and Traonouez, L.-M. (2009). Romeo: A parametric model-checker for Petri

nets with stopwatches. In Kowalewski, S. and Philippou, A., editors, *15th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2009)*, volume 5505 of *Lecture Notes in Computer Science*, pages 54–57, York, United Kingdom. Springer.

nxtOSEK (2009). NXTOSEK *Operating System, version 2.10*. http://lejos-osek.sourceforge.net/.

Object Management Group (OMG) (2001). *Model Driven Architecture (*MDA*) Guide, version 1.0.1*. http://www.omg.org/mda/.

Object Management Group (OMG) (2007a). UML *Profile for Modeling and Analysis of Real Time and Embbeded systems (*MARTE*), second revision submission*. http://www.omg.org/marte/.

Object Management Group (OMG) (2007b). *Unified Modeling Language (*UML*) : Superstructure, version 2.1.2*. http://www.omg.org/mda/.

OSEK/VDX Group (2005). OSEK/VDX *Operating System Specification, version 2.2.3*. http://www.osek-vdx.org/.

Renault, X., Kordon, F., and Hugues, J. (2009). From aadl architectural models to petri nets: Checking model viability. In *ISORC*, pages 313–320.

Society of Automotive Engineer (SAE) (2004). *Architecture Analysis & Design Language (*AADL*) AS5506, version 1.0*.

Team, M. P. (2004). The metropolis meta model - version 0.4. Technical Report UCB/ERL M04/38, EECS Department, University of California, Berkeley.

The MathWorks (2007). *Real-Time Workshop User's Guide*. The MathWorks Inc., Natick, MA, USA.

Thomas, F., Grard, S., Delatour, J., and Terrier, F. (2008). *Embedded Systems Specification and Design Languages, Selected Contributions from FDL'07*, volume Embedded Systems Specification and Design Languages of *FDL selected papers*, chapter Software Real-Time Resource Modeling, pages 169–182. Springer, Barcelona, Spain, springer science+business media b.v. edition.

Vergnaud, T. and Zalila, B. (2006). Ocarina, a compiler for the AADL. Technical report, Paris, France. http://ocarina.enst.fr.