International Journal of Applied Engineering Research ISSN 0973-4562 Volume 12, Number 5 (2017) pp. 622-631

A Model-Driven Approach for the Validation of RTOS Constraints in Real-time Application Models

Nour Moadad, Wassim El Hajj Chehade, Riham Abdel Kader

Mathematics and Computer Science Department, Beirut Arab University, Beirut, Lebanon.

ORCID: 0000-0001-7360-5447 (Wassim) *ORCID:: 0000-0003-0063-0910 (Riham)*

Abstract
Designing real-time embedded systems requires coping with different target platforms, and therefore handling heterogeneous constraints related to time, synchronization, and memory footprint. For this end, Model-Driven Development has been proposed in which a generic application model is suggested regardless of the target platform. This application model will then be transformed into a platform specific model through model transformations. From the obtained platform specific model, code is generated. The generated code must conserve the functional and non-functional requirements specified in the generic application model on the target execution platform. Since every target platform has its own APIs and implementation patterns that vary significantly from those of another platform, the designer must be able to validate his design before starting the model transformation step. In this paper, we propose to precede the model transformation step with a feasibility test step that runs a check to detect whether any real-time properties specified in the generic application model do not match the concrete real-time properties of the target platform. To this end, the solution we present includes providing an explicit description of the constraints of the target platform using our proposed Platform Constraint Language (PCL).

Keywords: UML; MARTE; Real-Time Embedded Systems; MDD; Real-Time Validation; OCL; Platform Modeling;

INTRODUCTION

Nowadays, the use of Real-Time Embedded Systems (RTES) ranges from small simple devices such as household electronics to complex large control systems such as airplanes and power plants. Under the existing economic and competitive pressure, companies are forced to create these RTES products in a fast pace and at lower prices. Therefore, application portability, deployment process reusability, and maintainability have become a crucial issue for these companies.

Model-driven development (MDD) is an approach that aspires to respond to these issues by taking RTES development to a higher level of abstraction and exploiting models at the very core of the development process [1]. One pillar of MDD for real-time embedded software is the separation of concerns between application model and target platform. The Model-Driven Architecture (MDA) initiative[1], undertaken by the Object Management group (OMG) and one of the most mature MDD formulation, involves a Y-chart design process in which a platform-independent model (PIM) of the system is transformed into a platform-specific model (PSM), given a platform description model (PDM). Such models can be described using the Unified Modeling Language (UML) [2].

The transformations from a PIM into a PSM can be classified into two groups: specific and generic transformations. In the former, fully automated code generation (model-to-text transformers) is used. No human intervention is required in the generation process since all platform specific implementation aspects related to timing communication and synchronization properties are embedded in the code generators. This approach offers many advantages to RTES developers, including increased productivity and enhanced source code consistency [3]. However, these generators are hard to port to new platforms as they contain a mixture of application concerns and platform-specific implementation concerns. A change in either of these requires expensive rewrites and a rare combination of skills of the tool-chain implementors.

To overcome this shortcoming, other Model Driven Development approaches have been proposed based on **generic transformations** [4][5]. These approaches promote a real separation of concerns between three actors: the developer of transformation tools who specifies a generic model transformation, platform providers who provide detailed models of their platforms, and the multitasked system designer that models the application system.

Although the above is an innovative way to achieve generic deployment, the consequent generated model might not match the original platform independent model. This scenario happens in the case where the values of the real-time properties specified in the platform-independent model do not fall in the range of the values described in the concrete platform model. In that case, the designer will have to iterate over the application model, searching for the source of mismatch between the generated model and the platform model, then modifying the real time values in the application model, and re-deploying it to get an implementable solution. These modifications usually take a considerable amount of time and require the designer to own a rare combination of

[1] *http://www.omg.org/mda/*

skills. As a result, the deployment and validation processes of the system become a time and energy consuming task.

With these MDD approaches available in the field, we are generating platform specific models with no guarantee that the generated system will conform to the concrete platform properties and therefore we cannot be sure it will be operational. Our research overcomes the above shortcoming by proposing to add a feasibility test step before the transition from the platform independent model to the platform specific model. The purpose of this step is to detect any possible mismatch between the real-time properties specified at the design model and the concrete real-time properties defined in the target platforms. To achieve this approach, we propose to augment the target platform with an explicit description of the platform constraints using our proposed Platform Constraint Language (PCL).

The paper is structured as follows. We first give a quick background on software platform modeling and how they are used to establish generic model driven development approaches (Section 2 and Section 3). Section 4 describes our proposed platform constraint language. Section 5 describes how this language could be used to achieve a valid model driven development process. Section 6 presents a use case to evaluate our approach. We describe related work in section 7. The final section of the paper presents our conclusion and future work.

TOWARDS A GENERIC PLATFORM MODELING

In a model-driven context, the design of RTES consists of two folds: developing a platform independent model (PIM) and specifying a platform description model (PDM). The former specifies real-time properties in an abstract way, modeling, on the one hand, quantitative features, such as priority and period, and on the other hand, qualitative aspects related to synchronization and communication. The PDM is specifically designed for each intended RTOS. It models the resources and the services provided by the RTOS.

As the PIM is refined to a PSM, the abstract resources in PIM are mapped to concrete resources of the target RTOS. This mapping is done through model transformations specifically designed for the intended RTOS by the tool chain implementer. In this case, the PDM is not a standalone model but it is embedded in the transformations.

In [5], we proved that the use of a generic model transformation, which is based on an explicit standalone PDM specifically designed for each target RTOS is the most suitable to reduce the cost of multi-platform deployment. To build generic transformations, and therefore, a generic mapping process between abstract and concrete platform resources, a common description language must be used between these two platforms. Several work have been interested in the definition of Domain Specific Languages (DSLs) or profiles [12][13] for platform description. Among these work, Software Resource Modeling (SRM) subprofile of the OMG standardized UML profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE) [11] seems to be the best choice. On the one hand, SRM defines a language and a conceptual framework that permit the description of RTOS resources and services in a standardized way. On the other hand, it plays a pivotal role for transitions between different platform models.

Based on SRM, we have proposed in [6], a methodology for platform modeling that is based on representing each platform resource by a UML class and annotating this class with stereotypes. Figures 1 and 2 show, respectively, the Java platform model and the C++/POSIX platform model described using our methodology. In these figures, the Java "Thread" resource is represented as a Thread class while the POSIX "Pthread_t" resource is depicted is a "Pthread_t" class. Since these 2 resources are identified as concurrent resource concepts, their classes are annotated with the stereotype "*SwSchedulableResource*", which represents concurrent execution threads in SRM.

In our methodology, we also propose to model services offered by a given platform resource using the appropriate standardized SRM roles of its corresponding stereotype. For instance, in the "Thread" resource, the init() and initialization() operations used to create respectively the Java and POSIX threads is associated with the generic "createServices" property of the "SwSchedulableResource" stereotype. Consequently, we bind the init() and initialization() operations to this property as indicated in Figures 1 and 2.

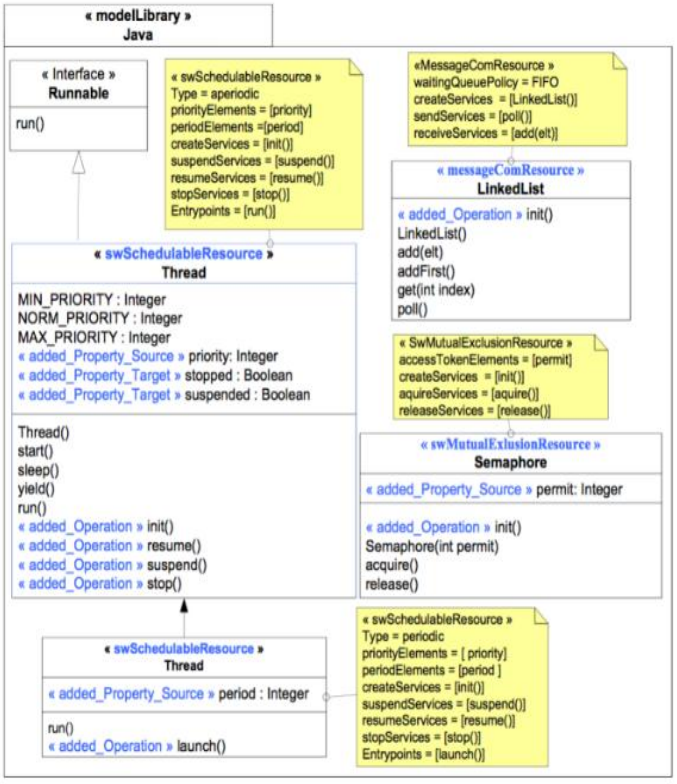

Figure 1. Detailed Java Platform Model

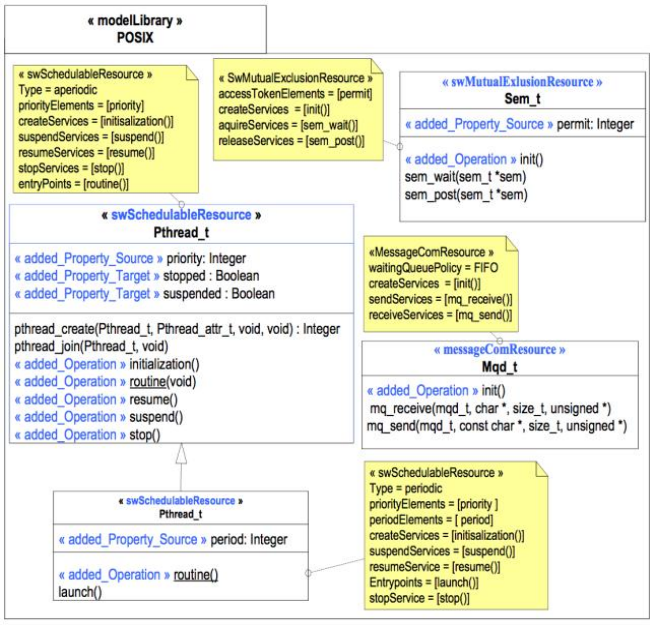

Figure 2. Detailed C++ Platform

Moreover, the init() and initialization() operations are not native operations of the platforms. Following our methodology, they were added to the Java "Thread" and POSIX "Pthread t" to capture the behavior of creating a thread. Therefore, they are annotated by the "*added_Operation*" stereotype as shown in the previous 2 figures. Figure 3 shows the detailed implementation of the added initialisation() operation.

The "*added_Property_Source"* stereotype is applied to properties that take their values from the application model. For instance, the property "priority" in the "initialization()" operation of Figure 3 is used to set the value of the thread priority that will be specified by the designer at the application level.

```
void initialisation() {
    sched_param param;
    param.__sched_priority = priority;
    int ret = pthread_setschedparam(thread,
            SCHED_OTHER, &param);
    int ref = pthread_create(&thread, NULL,
            routine, (void *) this);
    pthread_join(thread, NULL);
}
```

Figure 3. C++/POSIX Create Service Implementation

GENERIC DEPLOYMENT PROCESS

Based on the detailed platform model described above, we have proposed in [5], a new Model Driven framework to facilitate the implementation of RTES across different real-time execution platforms. The core of this framework is a generic model transformation that takes as input 3 models, the application model, the mapping model and the detailed platform model, and that generates one output model. The three input models are depicted in Figure 4. Next we will give a detailed definition of the application model and mapping model.

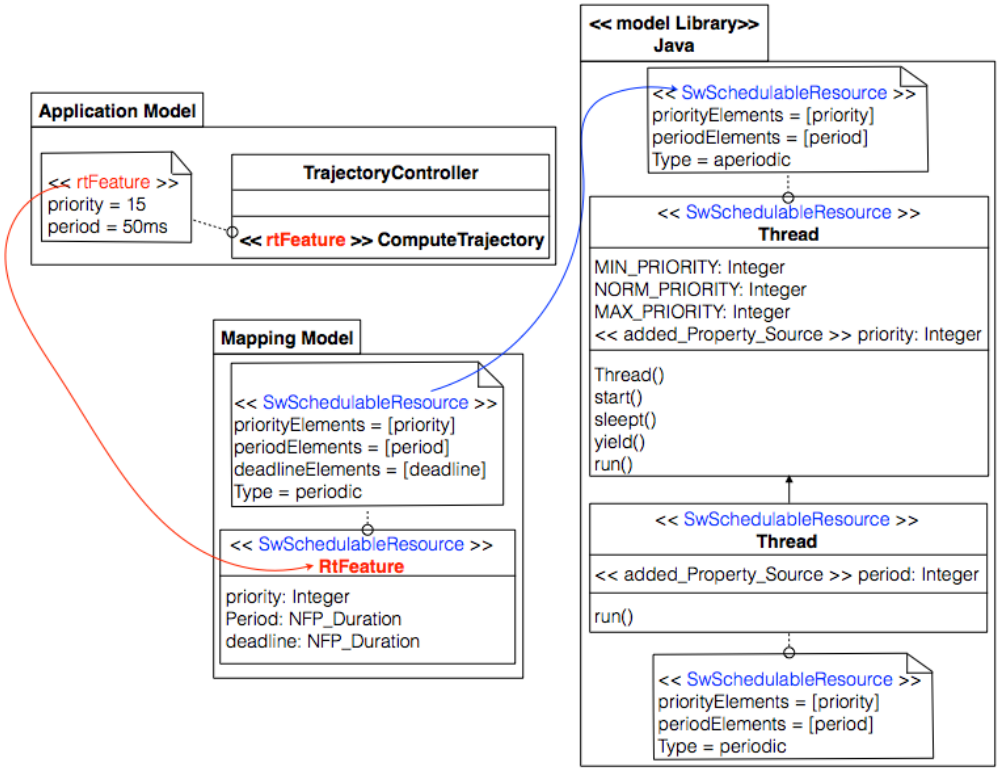

Figure 4. Example of an application model (PIM), Java platform description model (PDM), and their associated mapping model.

Application Model: The main UML diagram to describe the system structure of the application is the class diagram. This diagram describes the static structure of the RTES under design. It shows all classes that are responsible or related to the handling of the functional requirements. At this level of abstraction, software designers can specify quantitative features such as periods and deadlines and qualitative features related to concurrency and synchronization. These RTES features can be captured using UML profiles dedicated for real-time embedded domains (e.g., HLAM, RT-UML [7]). In this work, we choose to use the HLAM subprofile of MARTE. Figure 4 shows an application model of a trajectory controller system designed using this approach. We can notice that the computeTrajectory method is annotated by the "rtFeature" stereotype to capture that this method will run on it is own thread with a priority equal to 15 and a period equal to 50 ms.

Mapping model: The qualitative and quantitative features captured in the application model through HLAM stereotypes must be mapped to the correspondent elements in the detailed platform model which in turn are annotated using SRM stereotypes. To achieve this mapping, we offer a mapping model that links the HLAM stereotypes to their corresponding SRM stereotypes. This proposed mapping model represents the core of our generic model transformation. Figure 4 shows an example mapping model.

Each domain concept used in the application model is represented in the mapping model, as a UML Class and annotated with the corresponding stereotype of the SRM profile. The mapping of HLAM stereotypes used in the application model to SRM stereotypes is shown in Figure 4 using arrows. Thus, the *TrajectoryController* class of type *<<rtFeature>>* corresponds to the stereotype *<<SwSchedulableResource>>* which in turn is mapped to a *Thread* in the PDM. Therefore, the abstract resource *TrajectoryController* will be transformed to a concrete resource that implements a Thread.

As can be seen, the SRM stereotype is used as a semantic pivot in order to realize the mapping between a concept at the application model and a specific element in the detailed platform model.

The main strengths of this model are: first, the designers have the option to use any real-time design language (e.g. HLAM, RT-UML) in the application model, therefore the design model is not constrained to any specific design technology, and second, it provides a correspondence relationship between real-time design languages and resources of different platforms (e.g., between "RtFeature" and Java and POSIX threads).

PLATFORM CONSTRAINT LANGUAGE

Once the application model (PIM) is transformed to a platform specific model (PSM) on the specific RTOS, one of the key concerns is to ensure a correct deployment of the application. The previous section showed how SRM can be used in platform modeling and to integrate these platform models in a generic deployment framework. However, in order to guarantee a correct deployment, there are several platform constraints that must be taken into consideration that are not captured using SRM. For instance, in Java, the priority level for a thread should be between 0 and 10 [8] . In Figure 4, the class *TrajectoryController* of type *<<rtFeature>>* specifies that the priority of the Thread should be set to 15. In this case, the value is outside the allowed range of [0, 10] and this will result in a deployment error.

In POSIX, the constraint on thread priorities is more complicated. The priority level varies according to the scheduling policy. If the scheduler is set to SCHED_FIFO, the priority level is between 1 and 99. If it is set to SCHED_OTHER which is the standard round-robin time-sharing policy, the priority level is between 0 and 0 [9].

As constraints vary from a platform to another, there is a need to capture, these platform constraints on the platform models to help the designer detect early potential deployment problems and thus reduce development costs. This needs to be done in a generic way to still guarantee a generic deployment of the application across different platforms.

To capture constraints on UML models, we propose the use of the Object Constraints Language (OCL) [10] mixed with SRM. The constraints described in OCL will be included inside the platform model. In Java, the constraint on the thread priority level can be expressed using OCL in the following way:

context Thread

inv: self.priority <= 10 and self.priority >= 0

In POSIX, the priority level constraint is written using OCL in the following way:

context pthread_t

inv: self.scheduler.policy = 'SCHED_FIFO' ***implies***

self.priority <= 99 and self.priority >= 1

inv: self.scheduler.policy = 'SCHED_OTHER' ***implies***

self.priority <= 0 and self.priority >= 0

As can be seen, the OCL constraint starts with the *context* keyword followed by the name of the class on which the constraint must be applied. In the Java case, it is the class Thread, and in the POSIX case it is the class pthread_t. The keyword *inv* indicates the beginning of the specification of the constraint and that this constraint is a static one. Therefore, the above expression "self.priority <=10 and self.priority >= 0" must be true for every instance of a Java Thread. This means that, for every resource of type Thread referred to as *self* (*self* is always any object of the context), *self* must always have its priority attribute greater or equal to 0 and less or equal to 10. In the POSIX case, the *inv* constraint is more complex since the allowed values of the priority attribute depends on the chosen scheduler policy. This is specified using the keyword **implies** and 2 inv clauses, one for each choice of scheduler policy (SCHED_FIFO and SCHED_OTHER).

As an example, let us apply the above OCL constraints on the method *computeTrajectory* of the class trajectoryController shown in Figure 4. The method is annotated with the stereotype "rtFeature" which is mapped through the mapping model to the *SwSchedulableResource* stereotype which in turn corresponds to a Thread resource if Java is the deployment platform, and to a pthread_t if POSIX is used. The validation process must detect that this application cannot be implemented properly in Java since a priority of value 15 does not correspond to the Thread priority levels ranging between 0 and 10. On the other hand, this same validation process must detect that the application could be implemented on POSIX only if the scheduling policy is FIFO.

If we use the constraints written in pure OCL as shown above, the validation process will need to have a different checker for each platform. This is because each checker will be implemented in such a way to parse certain class names and attribute names. Therefore, if a need to target a new platform arises, we will need to provide a new checker. As a result, the validation process is not generic and will be hard to port to new platforms.

To tackle this problem, we propose to use the Platform Constraint Language (PCL). PCL combines OCL with SRM. It allows to write the OCL constraints independent of the target RTOS resources and properties by replacing the platform specific resources and properties (e.g. Thread or pthread_t) with their corresponding SRM resources and properties (e.g. *SwSchedulableResource)*. This approach will let us develop a generic validation process framework of application models. Below, we show how the above two OCL constraints for the thread priority in the two platform Java and POSIX are written using PCL.

context SwSchedulableResource

inv: self.priorityElements <= 10 and self.priorityElements >= 0

context SwSchedulableResource

inv: self.scheduler.schedPolicy = 'SCHED_FIFO' ***implies***

self.priorityElements <= 99 and self.priorityElements >= 1

inv: self.scheduler.schedPolicy = 'SCHED_OTHER' ***implies***

self.priorityElements <= 0 and self.priorityElements >= 0

The PCL constrains use the SRM resources and properties instead of the platform ones. Therefore, instead of using the Java *Thread* and *self.priority* resources and the POSIX *pthread_t* and *self.priority* resources, we use the *SwSchedulableResource* and *self.priorityElements* for both platforms. In Java, *SwSchedulableResource* and *self.priorityElements* are mapped respectively to the *Thread* and *self.prirority* resources. The same holds for POSIX where *SwSchedulableResource* and *self.priorityElements* are mapped respectively to the *pthread_t* and *self.prirority* resources.

Figure 5 shows how the two PCL are embedded inside their corresponding platform model.

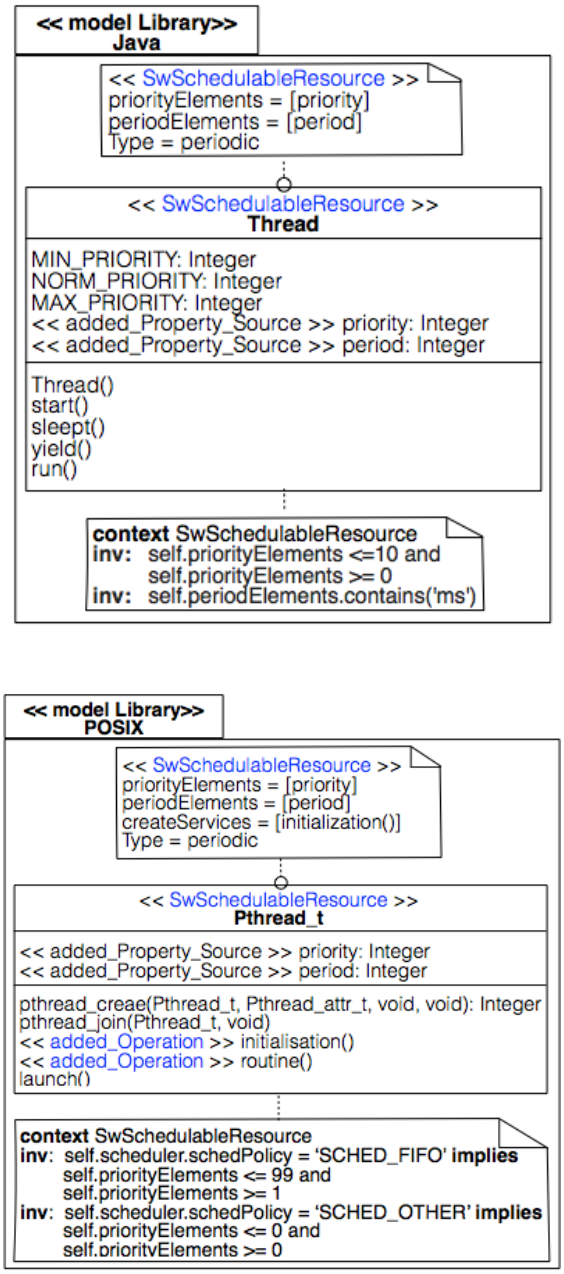

Figure 5. Detailed Java and POSIX platform with constraints.

TOWARDS A VALID IMPLEMENTATION

This section describes a framework that guarantees valid platform specific models. A valid platform specific model is a model that preserves the properties set in the application model on the target execution platform. The proposed framework is based on a feasibility step that detects possible non conformation between real-time properties specified at the design model and the concrete real-time properties of the target platforms. It provides feedback to the designer and aids him in detecting the source of any existing problem.

Feasibility Step: This step uses the target platform model discussed in section 3, and the application model and the mapping model described in Section 4. The output of this step

is a report that tells the application designer if potential errors or warnings exist in his design.

An *error* is generated when the application model is not implementable on the target RTOS. For example, in Figure 4, the computeTrajectory method in the TrajectoryController class has a priority value of 15 which is higher than the Java allowed levels ranging between 0 and 10. In this case, an error is generated with an associated message indicating that the assigned priority is higher than the allowed Java priority level.

A *warning* is generated when the application model is implementable on the target RTOS with some modifications to make. For example, when implementing a periodic thread in Java, the unit of the period is "ms". If the designer specifies a value of the period with a unit different from that of the target platform, a warning will be generated with an associated message indicating that the unit for the period in Java is specified in "ms". In this situation, the designer can adapt accordingly the value of the period in his design.

Next we describe the validation algorithm that validates the properties specified in the application model against the constraints set in the PDM, and generates the feasibility report of the application.

The Validation Algorithm

This section presents in details the validation algorithm.

Algorithm 1 - validatePIM: This algorithm validates an application model (PIM) against the target platform model (PDM). It takes as input the PIM, PDM, and the mapping model (MM).

The algorithm starts **(lines 1-2)** by finding for each stereotype used in the application model (st_{PIM}), its corresponding target plato

rm resource (st_{PDM}) using Algorithm 2 (getEquivalentPlatformResource). Then for each target platform resource (st_{PDM}), the algorithm finds its corresponding platform constraint (pc), if any is defined **(line 3)**. This is accomplished using Algorithm 3 (getPlatformConstraint). Once the platform constraint is retrieved, each invariant condition in the constraint is checked against the values specified in the application model **(lines 4-5)** using Algorithm 4 (matchConstraint).

Algorithm 1: validatePIM

Input: PIM, PDM, MM

Notations:

st_{PIM}	stereotype used in PIM
st_{PDM}	stereotype used in PDM
pc	a platfom constraint
inv	invariant in platform constraint
(attr, val)	a specific attribute in PIM with its as signed value
$attr_{SRM}$	attribute as specified in SRM

1 FOR EACH st_{PIM} in PIM
2 st_{PDM} ← **getEquivalentPlatformResource**(st_{PIM}, MM)
3 pc ← **getPlatformConstraint**(st_{PDM}, PDM)
4 FOR EACH inv in pc
5 **matchConstraint**(inv, PIM)
6 END
7 END

Algorithm 2 - getEquivalentPlatformResource: Given an application model stereotype st_{PIM}, this algorithm finds its corresponding target platform resource (st_{PDM}). It takes as input the application model stereotype and the mapping model.

Each class in the mapping model is a UML Class having a class name and annotated with an SRM stereotype. For instance, the *RtFeature* class in the mapping model in Figure 4, is annotated with the stereotype *SwSchedulableResource*. This same stereotype corresponds to a stereotype on the platform model (PDM). Therefore, for each pair *(className, st_{PDM})* in the mapping model, the algorithm checks if the className is equal to the input application model stereotype st_{PIM}, and then returns the corresponding st_{PDM} **(lines 1-5)**.

As an example, let's suppose that we want to find the st_{PDM} corresponding to the st_{PIM} *RtFeature* annotating the method computeTrajectory in the TrajectoryController class in the application model in Figure 4. Given as input the mapping model in Figure 4, the algorithm matches the st_{PIM} *RtFeature* to the class *RtFeature* found in the mapping model. The class is annotated with the stereotype *SwSchedulableResource*

which corresponds to a target platform resource in the PDM. The *SwSchedulableResource* stereotype is then returned by the algorithm.

Algorithm 2: getEquivalentPlatformResource

Input: st_{PIM}, MM

Output: st_{PDM}

Notations:

st_{PIM}	a stereotype used in PIM
st_{PDM}	a stereotype used in PDM
(className, st_{PDM})	the class name and its corresponding stereotype.

1 FOR EACH (className, st_{PDM}) in MM
2 IF st_{PIM} = className
3 Return st_{PDM}
4 END
5 END

Algorithm 3 - getPlatformConstraint: Given a platform stereotype, this algorithm finds its corresponding platform constraint if any is defined. It does so by checking the input stereotype to each pair of (st_{PDM}, pc) defined in the target platform model. If a match is found, the corresponding pc is returned **(lines 1-5)**.

As an example, suppose that we want to find the constraints attached to the *SwSchedulableResource* stereotype. The algorithm needs to check the PDM of the corresponding platform, find the class annotated by *SwSchedulableResource* in the PDM, and then retrieve the PCL constraint attached to it. If the platform is Java, the algorithm will retrieve the constraint attached to the Thread class in the Java PDM shown in figure 5.

Algorithm 3: getPlatformConstraint

Input: stereotype, PDM

Output: pc

Notations:

st_{PDM}	a stereotype used in PDM
pc	a platfom constraint
(st_{PDM}, pc)	the pair of stereotype and its corresponding platform constraint

1 FOR EACH (st_{PDM}, pc) in PDM
2 IF stereotype = st_{PDM}
3 Return pc
4 END
5 END

Algorithm 4 - matchConstraint: The algorithm takes as input an invariant *inv* from a platform constraint and the application model *PIM*. For each constraint specified in the invariant clause **(line 1)**, we find the corresponding attribute and value set by the designer **(lines 2-3)**, then we check if the assigned value matches the invariant condition **(line 4)**.

For Example, the PCL attached to the stereotype *SwSchedulableResource* in the Java PDM states that the value of the *priorityElement* of a thread must be between 0 and 10. The *priorityElement* of a thread corresponds to the property *priority* (figure 5). Therefore the attribute/value pair (priority, 15) in the PIM will be checked against the conditions (priority, >, 0) and (priority, <, 10) specified in the PDM.

Algorithm 4: matchConstraint

Input: inv, PIM

Notations:

inv invariant in platform constraint

AV_{PIM} set of (attribute, value) pairs specified in PIM

ACV_{INV} set of (attribute, condition, value) triplets specified in inv

1 FOR EACH (*attr*, *cond*, *val*) in ACV_{INV}

2 FOR EACH (*attr'*, *val'*) in AV_{PIM}

3 IF *attr* = *attr'*

4 **checkCondition**(*attr, cond, val, attr', val'*)

5 END

6 END

CASE STUDY

In this section, we illustrate the proposed approach through a case study. We consider the application model shown in Figure 6. It consists of two producers that periodically generate measurements, and of a consumer that evaluates them. Whenever abnormal values of measurements appear, a worker console is notified. The last part in the system is a counter that counts the number of measurements produced and not consumed yet. The counter implemented as a shared resource between the producer and the consumer.

The objective of this case study is to check the feasibility of this application on two different target execution platforms Java and C++/POSIX. To do so, we need the application model, the mapping model, and the two target platform models.

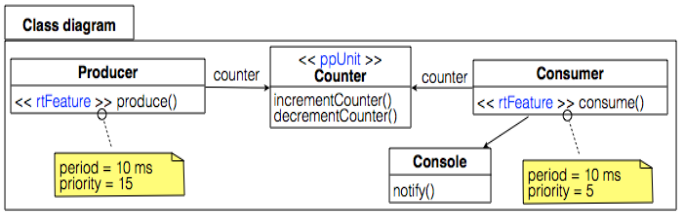

Figure 6. Specification of the structure of the platform-independent application.

Application model: Figure 6 shows the class diagram of our case study example. The Producer and Consumer classes are two active objects. Therefore, operations "produce()" and "consume()" are annotated with "RtFeature" to capture the fact that they need to be executed on their own execution threads. The period and the priority of the operations are also specified. The Counter class is shared between the Producer and the Consumer classes and the access to this object shall be synchronized. Therefore, the Counter class is annotated with the "ppUnit" stereotype [11].

Checking the feasibility of the design: We will use our proposed approach to check the feasibility of the producer consumer application model when deployed on Java and POSIX. We will use the Java platform model and the POSIX platform models of Figure 5. These two models include the correct PCL constraints.

The result of the feasibility check on Java is shown in Figure 7. We can see that there is one error and one warning. The error is due to the fact that the designer has specified a priority for the produce which is higher than the max priority value allowed in Java. In this case the designer should update his design and assign a value to the producer within the range of the allowed Java priority levels. The warning is due to the value of the period of the method produce at the application model which is specified in "s" and it should be specified in "ms".

```
-------------------------------------------------------
Feasiblity check in progress ...

Feasiblity check complete
ERROR: priority in Producer must be between 0 and 10
Warning: period in Producer must be in ms
-------------------------------------------------------
```

Figure 7. Feasibility report Generated for the Java Platform

Figure 8 shows the result of the feasibility test on the POSIX platform. We notice that there are two warnings. The first warning tells the designer that the priority values set at the application model are valid if the scheduling policy is SCHED_FIFO or SCHED_RR. The second warning is also related to the unit of the period that must be set in "ms".

```
-------------------------------------------------------
Feasiblity check in progress ...

Feasiblity check complete
Warning: priority in Producer is valid if schedPolicy
 is SCHED_FIFO or SCHED_RR
Warning: period in Producer must be in ms
-------------------------------------------------------
```

Figure 8. Feasibility report Generated for the POSIX Platform

RELATED WORK

A considerable amount of work has already been done on execution platform modeling and the consideration of this platform during the development cycle to transform a system specification to an implementation.

MoPCoM approach [16] is a co-design methodology that represents a refinement of the MDA Y-Chart and is based on OMG standards. This methodology uses UML and MARTE to represent the different models and SysML [17] for the expression of functional and non-functional requirements enhanced with some MARTE elements. MoPCoM methodology defines three design levels: Abstract Modeling Level (AML) is the first design level, where the goal is to model system behavior. Execution Modeling Level (EML) is the intermediate level, where performance analysis can be done, thanks to the availability of execution nodes topology of the system. Detailed Modeling Level (DML) is the last modeling level, which allows code generation. Although MoPCoM allows the deployment of RTES and performs performance analysis, the validation process is not generic and depends on the underlying target platform.

The authors in [4] proposes a model driven framework that allows explicit platform description using SRM and the generation of platform-specific models. Although this approach enables the generation of platform-specific models using domain-independent transformation rules, it permits just a structural generation. In this approach, explicit platform description lacks the support of platform resource constraints. They make the assumption that the deployment is always possible, so no feasibility test is done before deployment.

In the same way, the deployment process, proposed in [12], based on RTEPML language aims to provide generic transformation rules to automate the deployment process of application models. This approach claims the necessity to perform feasibility tests in particular to verify the availability of a concept on the target platform before the deployment. However, this approach doesn't deal with the specific constraints of the platform resources of the RTOS and its influence on the validity of generated model.

In [14], the author proposed a two steps approach based on an explicit description of platform models. The first step consists in feasibility tests whose role is to help the designer detecting the potential refinement problems. The second step is a mapping step that ensures the compliance of the implementation model with the design model taking into consideration the characteristics of the target execution platform. Although, this approach performs a feasibility step before deployment, the feasibility step is based on implicit description of RTOS constraints. They didn't propose a systematic approach that describes how to model constraints.

Aspect oriented model-driven engineering approaches, in particular the work presented in [15], propose to handle RTES requirements at the application level using aspects. Aspects platform-specific implementations are specified via scripts in the GenERTiCA tool. This tool enables automating the generation of executable code. This approach provides generic models of aspects that handle non-functional requirements starting from early development stages and proposes to handle aspect implementations independently of the transformation tool. However, our approach offers a more transparent way for managing non-functional requirements implementations and enhances separating of concerns by using the detailed platform modeling approach.

CONCLUSION AND FUTURE WORK

In this paper, we have proposed a MDD approach to automate the validation of the RTES application models. This approach aims at providing a generic and automatic guidance framework to the designer in order to guarantee that generated models conserve non-functional properties of the application model on the target platform. The proposed approach is based on an explicit definition of the concrete platform (RTOS) constraints using what we called the platform constraint language which is a combination of OCL and SRM. We presented, also, a small case study that shows how our proposed framework works.

We should mention that the completeness of our approach is based on the capability of SRM to cover resources, resource services and resource properties of the execution platforms. A full study on the coverage of SRM is presented in [4].

Future research will focus on extending our framework to support validation of additional non-functional properties, such as schedulability analysis and the integration of the proposed validation framework in a model driven development approach that aims to generate executable code.

REFERENCES

[1] B. Schtz, A. Pretschner, F. Huber, J. Philipps. Model based development of embedded systems, Lecture Notes in Computer Science, vol 2426, 2002, Springer, 2002, pp.331-336.

[2] Object Management Group, OMG Unified Modeling Language (OMG UML), Superstructure, V2.5, March 2015, OMG document number: formal/15-03-01.

[3] Jack Herrington. 2003. Code Generation in Action. Manning Publications Co., Greenwich, CT, USA.

[4] F. Thomas, S. Gérard, J. Delatour and F. Terrier, Towards a Framework for Explicit Platform-Based Transformations, 11th IEEE International Symposium 2008, Orlando, FL, 5-7 May 2008.

[5] Chehade, W.E.H., Radermacher, A., Terrier, F., Selic, B., Gerard, S.: A model-driven framework for the development of portable real-time embedded systems. In: Proceedings of the 16th IEEE International Conference on Engineering of Complex Computer Systems, pp. 45–54. IEEE Computer Society, Las Vegas (2011).

[6] Chehade, W. E. H., Radermacher, A., Gerard, S., & Terrier, F. (2010, November). Detailed real-time Software platform modeling. In 2010 Asia Pacific Software Engineering Conference (pp. 108-117). IEEE.

[7] Object Management Group, UML profile for Schedulability, Performance and Time, Object Management Group, inc., 2005. OMG document: formal/05-01-02.
[8] Oracle. Java Constant Field Values (2016, October 26). Retrieved from: http://download.oracle.com/javase/6/docs/api/constant-values.html#java.lang.
[9] sched(7) - Linux manual page (2016, October 26) . http://man7.org/linux/man-pages/man7/sched.7.html.
[10] Object Management Group. OCL 2.4 Specification (2014).
[11] Object Management Group, A UML profile for MARTE (version Beta 2), June 2008, OMG document number: ptc/2008- 06-09.
[12] M. Brun. Contribution à la considération explicite des plates-formes d'exécution logicielles lors d'un processus de déploiement d'application. PhD Thesis university of Nantes. October 2010.
[13] Tivadar Szemethy: Domain-Specific Models, Model Analysis, Model Transformations. Phd Thesis, University of Vanderbilt, Nashville, Tennessee, USA, May 2006.
[14] Mzid, R., Mraidha, C., Babau, J. P., & Abid, M. (2012, September). A MDD Approach for RTOS Integration on Valid Real-Time Design Model. In *2012 38th Euromicro Conference on Software Engineering and Advanced Applications* (pp. 9-16). IEEE.
[15] Wehrmeister, M.A. An Aspect-Oriented Model-Driven Engineering Approach for Distributed Embedded Real-Time Systems, PhD Thesis, Faculty of Electrical Engineering, Computer Science and Mathematics, Department of Computer Science, Paderborn, September 2009.
[16] A. Koudri, D.A. Joel Champeau, P. Soulard. Mopcom/marte process applied to a cognitive radio system design and analysis. In proceeding of the Model Driven Architecture, Foundations and Applications (2009).
[17] OMG, Systems Modeling Language version1.3, formal/2012- 06-01, 2012.