Real–Time Design Models to RTOS-Specific Models Refinement Verification

Rania Mzid, Chokri Mraidha
CEA List, Laboratory of model driven engineering for embedded systems
Point Courrier 174, Gif-sur-Yvette, 91191, France
rania.mzid@cea.fr
chokri.mraidha@cea.fr

Jean-Philippe Babau
Lab-STICC, UBO, UEB
Brest, France
Jean-Philippe.Babau@univ-brest.fr

Mohamed Abid
CES Laboratory
National school of engineers of Sfax
Sfax, Tunisia
Mohamed.Abid@enis.rnu.tn

ABSTRACT
One key point of Real-Time Embedded Systems development is to ensure that functional and non-functional properties (NFPs) are satisfied by the implementation. For early detection of errors, the verification of NFPs is realized at the design level. Then the design model is implemented on a Real-Time Operating System (RTOS). However, the design model could be not implementable on the target RTOS. In this paper, we propose to integrate between the design and the implementation phases, a feasibility tests step to verify whether the design model is implementable on the target RTOS and a mapping step to generate the appropriate RTOS-specific model. This two-steps approach is based on an explicit description of the platform used for verification and the RTOS which is the implementation platform. Moreover an additional verification step is needed to ensure the conformity of the implementation model to the design model with regard to NFPs.

Keywords
MDD, Design Model, RTOS-specific Model, Real-Time Validation;

1. INTRODUCTION
In order to overcome the increasing complexity of Real-Time Embedded Systems (RTES), Model-Driven-Development (MDD) [1] promotes a rise in level of abstraction by introducing intermediate models from specification to implementation, while passing through design, and enabling validation at each level.

At the design level, scheduling analysis [2] may be applied to validate design choices in terms of timing requirements. Several tools are available to carry out such validation, one can cite as example Qompass-Architect [3], Cheddar [4], and MAST [5]. However, to achieve that, each of these tools considers some implementation assumptions (e.g. scheduling policy, communication mechanisms) which are related to a *validation platform*. On the other hand, there is an important number of Real-Time-Operating-System *(RTOS)* in the market. Some are compliant to a specific standard such as POSIX [6], OSEK-VDX [7] and μITRON [8], some are commercial and others are free and may be not compliant to any standard. These RTOS or standards share common concepts but with specific features [9]. The choice of the target RTOS depends on the considered community and the intended use [10].

From these considerations, the refinement of the design model, making some implementation assumptions to be validated, to an RTOS-specific model is error-prone.

In fact, the selected RTOS may be too restrictive with regard to the validation platform or incoherent correspondence between properties of the validation platform resources and the RTOS ones may occur. In that case, the designer iterates on the design model, modifying and re-validating it, looking for an *implementable* solution. These modifications are usually based on the designer experience and reduce portability of design model: the design model becomes specific to an RTOS.

Several works are interested in the deployment of an application on a real platform. In [11], the authors propose a generative process to transform an application deployed on one RTOS to another based on an explicit description of the latter using the Software Resource Modeling (SRM) UML profile, which is part of the UML profile for MARTE [12] . This work makes the assumption that the deployment is always possible and did not paid any attention to the incoherence between the characteristics of the different platform resources and its influence on the validity of the obtained model. The author in [13] proposes a deployment process of an application on a RTOS. This process considers also an explicit description of the latter but using a Domain Specific Language (DSL) called RTEPML and focuses on defining generic transformations to automate the process. Compared to [11], this approach claims the necessity to verify the availability of a concept on the target RTOS before the deployment.

In previous work [14], we proposed a two-steps approach that ensures the generation of valid implementation model from design model fulfilling timing properties. This approach is based on an explicit description of the validation platform and the target RTOS using SRM [12]. The first step, which is a set of feasibility tests, aims at verifying whether the implementation assumptions made at the design level are implementable on the target RTOS. The second one is a mapping step that performs the mapping between the validation platform resources and the RTOS resources to obtain a RTOS–specific model. In [14], we have focused on concurrency aspects, scheduling policies, tasks and we have especially treated the tasks' priority problem. In this paper, we extend the proposed approach by treating the shared resources aspect. Thus, we consider that tasks in the design model may be dependent by sharing resources and we describe the required

ACES-MB'12, September 30 2012, Innsbruck, Austria Copyright 2012 ACM 978-1-4503-1800-6/12/09…$15.00.

resources in the validation and RTOS platforms. We discuss also the additional feasibility tests and mapping necessary from this perspective. On the other hand, one important issue is to verify whether the generated RTOS-specific model is valid with respect to the design model. So, in this paper, we focus also on identifying the required verification to confirm the correctness of such model.

The paper is organized as follows. Section 2 presents the assumptions of the paper. In section 3, we give an overview of the two-steps approach to generate valid implementation models and we add the necessary treatments for the shared resources aspect. In section 4, we explain how to verify the validity of the RTOS-specific model with respect to the design model. Section 5 illustrates on an example the approach and the verification phase. Finally, section 6 concludes the paper.

2. ASSUMPTIONS

We assume that timing validation is performed at the design level using Optimum methodology [3] supported by the Qompass-Architect tool. This methodology introduces timing validation from the specification level in order to guide the design of the concurrency model that satisfies the timing constraints.

In this paper, we assume that the design model, generated by Optimum consists of a set of tasks $\tau = \{\tau_1, \tau_2, \ldots \tau_n\}$ executing the different functions of the system. All tasks in the model are scheduled according to their priority. So each task τ_i is characterized by its priority P_i and runs at a base period T_i. Besides, we assume that two tasks in the model may be dependent by sharing resources. Consequently, the design model consists also of a set of resources $R = \{r_1, r_2, \ldots, r_m\}$ such as each $r_i \in R$ is shared between two tasks or more.

Finally, we suppose that the hardware architecture corresponds to a single execution node (mono-processor architecture).

From this correct model (design model), one objective of this work is to ensure a correct transition to the implementation model while respecting the timing properties. More precisely, we focus on platform aspect because validation is based on a validation platform, here the platform used by Qompass-Architect, while implementation is based on the RTOS.

3. MODEL-DRIVEN APPROACH

In this section we give an overview of the two-steps model-driven approach which has been explained in details in previous work [14]. Then, we extend this approach by considering shared resources aspect.

3.1 Overview

One key point of our approach is to ensure a correct deployment of a design model satisfying non–functional requirements on an RTOS. The obtained RTOS-specific model (implementation model) must conserve the properties that have been validated at the design level.

The design model translates the system specification and fulfills its timing constraints under the assumptions made by the validation platform related to the validation tool (c.f. Figure 1). In our case, the validation platform is the Optimum platform as we assume that timing validation is performed at the design level using Qompass-Architect [3]. In fact, the validation platform includes all concepts provided by RTOSs and that are necessary to perform timing validation. This makes this platform independent from a particular RTOS and provides a flexible framework to the designer to make different design choices.

In our approach, we choose an explicit description of the validation platform and the RTOS using SRM. Indeed, SRM allows capturing the semantics of the different concepts defined in both platform models and serves as a pivot language to automate the refinement of the design model to an implementation model.

As shown in Figure 1, the approach introduces two steps between the design and the implementation levels:

- *Feasibility tests step:* this step generates an *error* when the design model is not implementable on the target RTOS and provides a feedbacks to the designer to inform him about the source of the problem. It generates a *warning* when the design model is implementable but the RTOS provides an implementation that is probably more optimized than the one chosen at the design level. Otherwise, this step mentions that there is *no problem* and that the mapping step can be performed.

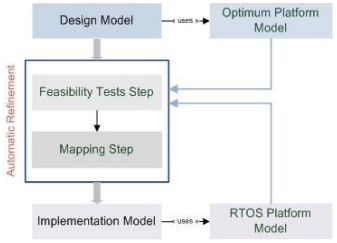

Figure 1.Model-Driven approach

- *Mapping step:* this step generates the RTOS-specific model by performing the mapping of concepts and the mapping of properties of these concepts. This mapping is based on the notion of matching between the resources of the validation platform and the RTOS one. This matching is ensured, in our case, by the use of SRM to describe both platforms (c.f. Figure 2).

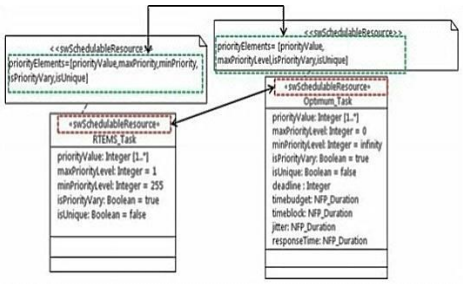

Figure 2.Matching using SRM

In [14], we were interested in concurrency aspects such as scheduling policies, tasks and their properties that describe the application behavior in a design model with *independent tasks*. The greater emphasis was on the priority problem. Briefly, we describe the tests invoked by the feasibility tests step which are related to the priority aspect.

- *Test of scheduler:* this test verifies the scheduling policies adequacy between the validation platform and the RTOS. For instance, if the scheduling policy used at the design level is priority-based and the RTOS does not offers a priority-based policy. So in that case, this test generates an error to mention that the input design model is not implementable on the target RTOS
- *Test of number of priority levels:* this test computes the number of priority levels used in the design model and

verifies whether the platform supports that number. If the number of priority levels allowed by the RTOS is lower than the number used at the design level, this test generates an error to indicate that the design model is not implementable on the target RTOS.

- *Test of equal priority levels:* this test verifies if, at the design level, there are tasks that share the same priority levels. In that case, if the target RTOS does not support such situation, this test generates an error to inform the designer that his design model is not implementable.

In order to generate the RTOS-specific model, the mapping step provides also different mapping strategies of the priority values to give a flexible framework to the designer. We give also a brief description of these mapping strategies:

- *Direct mapping* keeps at the implementation the same priority values used in the design model. This type of mapping does not ensure always valid implementation models.
- *Linear mapping* generates consecutive values from the available minimum priority level of the used RTOS. If feasible, it ensures always valid implementation models. However, the generated priority is less convenient to insert new task at run-time.
- *Mapping by step* is similar to the previous one, but adding a step between two consecutive levels of priority. The validity of the obtained implementation model depends on the step size. Like for direct mapping, it is necessary to add a supplementary test to verify whether this mapping is possible.
- *Proportional mapping* distributes applicative priority values over the maximal range offered by the RTOS. It guarantees valid implementation models. Nevertheless, this type of mapping is not possible if the RTOS does not provide an upper bound of priority levels.

3.2 Consideration of shared resources

We suppose that tasks in the design model may be dependent by sharing resources (c.f. section 2). The sharing of a data resource, when the use of the data must be atomic, necessitates choosing three architectural parameters: the *synchronization protocol*, the *allocation policy* and the *access protocol*. The synchronization primitive (e.g. Semaphore, Mutex) is needed to ensure that one and only one task can use the resource at a time. The allocation policy or the waiting queue policy (e.g. FIFO, priority-based) determines what happens when a request is made for the resource when the resource is busy. Finally, the access protocol (e.g. PCP, PIP) is used to avoid priority inversion situations or deadlock by modifying the priority of the task during the execution. The combined choice of synchronization protocol, allocation policy and access protocol corresponds to a *possible implementation* of the shared resource (critical section). In next subsections, validation and RTOS platform models are enriched to support the creation of design and implementation models with shared resources. Then, we discuss the feasibility test and mapping steps from this perspective.

3.2.1 Validation platform model for Optimum

In order to perform timing validation, the designer has to make implementation assumptions on how to implement the critical section. As already discussed in section 3.1, the validation platform is an "ideal" platform that offers unlimited design choices for the designer and is independent from a particular RTOS. Consequently, this platform covers all the ways for implementing a shared resource. To this end, we add a *Shared_Resource* concept to the Optimum platform (c.f. Figure 3) and we annotate the latter with *"swMutualExclusionResource"* stereotype from SRM. The choice related on how to implement this shared resource corresponds to setting the values of the *mechanism*, *waitingQueuePolicy* and *concurrentAccessProtocol* properties of the *"swMutualExclusionResource"* stereotype which correspond respectively to the *synchronization protocol*, *allocation policy* and *access protocol* parameters already mentioned. In Figure 3, we choose a default implementation of the shared resource (*PCP_Semaphore*). However, the designer can change this implementation by modifying the values of these properties. Depending on the designer choices, the validation tool involves the corresponding analysis test.

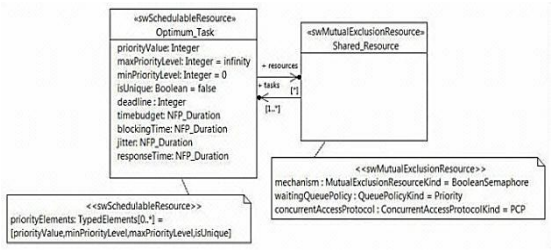

Figure 3.Excerpt of the Optimum platform model

Some combined choices of these three parameters do not correspond to real implementations. An example of non-meaningful implementation is; we choose a semaphore mechanism with a FIFO waiting queue and a PCP protocol. In order to avoid such situation, we propose to add an OCL constraint [15] for each non-meaningful implementation. The current implementation of SRM imposes to express these choices at the profile level (i.e. set the properties values of the *"swMutualExclusionResource"* stereotype). Consequently, the OCL constraints that prevent insignificant implementations of the critical section are also defined at the profile level. For instance, the previous unsound situation corresponds to a constraint associated to *"swMutualExclusionResource"* stereotype from SRM and is given just below:

```
context swMutualExclusionResource
inv:
(Self.mechanism = BooleanSemaphore) and
(Self.waitingQueuePolicy = FIFO) implies (not
(Self.concurrentAccessProtocol = PCP)
```

3.2.2 RTOS Model

To tackle the issue of shared resources, the RTOS model should describe the possible implementations of critical section provided by the considered RTOS. We choose, in this paper, RTEMS [16] as a target RTOS and we give in Figure 4 an excerpt (a view for the shared resources) of the RTEMS platform model.

RTEMS provides three possible implementations of a shared resource. Each of these implementations corresponds to a class in the RTEMS model annotated *"swMutualExclusionResource"*. For each class, we give default values to the *mechanism*, *waitingQueuePolicy* and *concurrentAccessProtocol* properties of the *"swMutualExclusionResource"* stereotype which define the considered implementation.

For example, The *FIFO_Semaphore_Resource* concept corresponds to a shared resource implementation using a Boolean semaphore as a synchronization protocol and FIFO as an

allocation policy. This implementation should not define an access protocol this is why it does not appear in Figure 4.

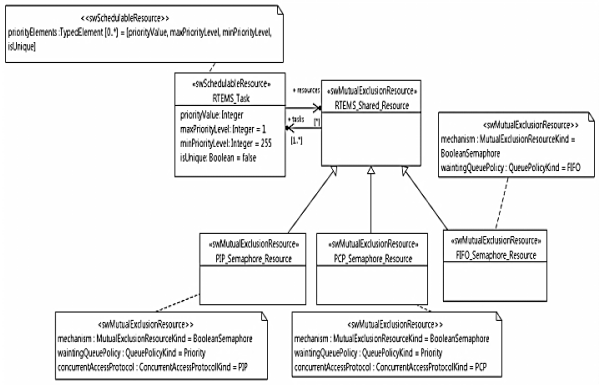

Figure 4.Excerpt of the RTEMS model

3.2.3 Feasibility tests and mapping steps

The extension of our approach to support tasks dependencies by sharing resources requires additional feasibility test to verify whether the target RTOS provides the critical section implementation chosen at the design level. If it is not the case, the feasibility tests step generates *an error* to inform the designer that the corresponding design model is not implementable on this RTOS.

In some cases, the implementation choices made by the designer to implement the critical section are implementable; however the RTOS provides another implementation that offers better real-time performance. In that case, the feasibility tests step generates *a warning* in order to propose to the designer to change the implementation. An example of such situation is; when the designer chooses a FIFO semaphore at the design level and the target RTOS provides a PIP semaphore. So the feasibility tests step highlights a warning to inform the designer that the target RTOS provides a PIP semaphore which is more adapted for real-time application [17]. The designer at this point can choose to keep his design model and to perform the mapping or to modify the implementation choices for the critical section taking into consideration the generated warning.

The mapping step for the shared resources is straightforward. If the design model is implementable, this step creates an instance at the implementation level of the resource that defines the same critical section implementation choices made at the design level.

4.RTOS-SPECIFIC MODELVERIFICATION

One key point of the proposed approach is to generate correct RTOS-specific models from valid real-time design models. In order to confirm that the obtained model is correct (i.e. preserves design model timing properties); some properties must be verified at the implementation level. In our case, we identify three properties:

- **P1**: the priority values of the different tasks must be always within the range of priority values allowed by the RTOS
- **P2**: the execution order of the different tasks defined at the design level must be preserved at the implementation level.
- **P3**: the access order to shared resources must be preserved

To address the first property (**P1**), we propose to add an OCL constraint to the RTOS model. The role of this constraint is to verify that the priority values of the different tasks in RTOS-specific model are meaningful to the considered RTOS. As an example, we give just below the constraint that we add to the RTEMS model and which corresponds to (**P1**).

Depending on the priority order (increasing or decreasing) which is determined by the *minPriorityLevel* and *maxPriorityLevel* attributes of the *RTEMS_Task* (c.f. Figure 4), this constraint verifies whether the priority values of the different tasks instances of the *RTEMS-task* are between the minimum and the maximum priority levels (given by *minPriorityLevel* and *maxPriorityLevel* and correspond respectively to 255 and 1 in RTEMS).

```
context RTEMS_Task
inv:
if(Self.minPriorityLevel< Self.maxPriorityLevel) then
Self.minPriorityLevel < Self.PriorityValue <
Self.maxPriorityLevel
else
Self.maxPriorityLevel< Self.PriorityValue <
Self.minPriorityLevel
endif
```

For the second property (**P2**), we don't focus on the priority values (which is already verified by the first property) but on the execution order of the different tasks which must be equivalent at the design and implementation level. In order to verify this property, we propose the meta-model given in Figure 4.

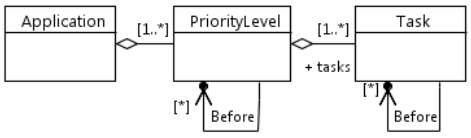

Figure 5.Verfication-oriented meta-model

This meta-model considers an application as a relation of precedence among priority levels. As we may have tasks that share same priority level, we consider in the meta-model that at each priority level one or more tasks may also have a relation of precedence. So this meta-model considers that the most important is not the values of priority but the relation of precedence between them. In order to verify the second property, we transform the design and the RTOS-specific models to models that conform to this meta-model and we verify if they are equivalent.

The access order to the shared resource in the model is preserved at the implementation level, if and only if, the execution order of tasks that share this resource is also preserved. Consequently, the third property (**P3**) is verified, if and only if, the second property (**P2**) is verified.

As a conclusion, the generated RTOS-specific model is correct with respect to the design model, if and only if, these three properties (**P1**), (**P2**) and (**P3**) are verified.

5. CASE STUDY

In this section we illustrate our approach with the example presented in [3]. This example corresponds to a classical case study in the automotive domain, i.e. the antilock control sub system.

This subsystem is a classic sensor-controller-actuator system. Figure 5 gives a structural view of the main functions inside the controller: a data processing function for data coming from the sensor, the anti-locking brake function calculating the command

to send to the actuator, and a diagnosis function that disables the anti-locking function in case a fault in the subsystem is detected.

Each function above has an associated behavior modeled here as an activity. Figure 6 below complements the structural functional model with the description of the system end-to-end scenarios. Two events (*acquisitionForAbs* and *acquisitionForDiagnosis*) are triggering the sequences of functions behavior execution.

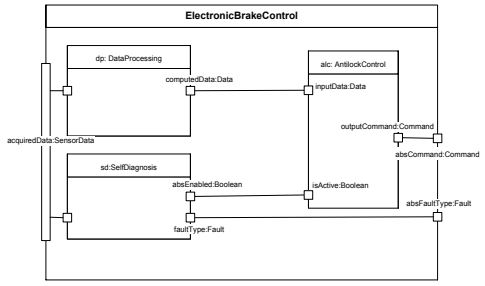

Figure 6. System functional model

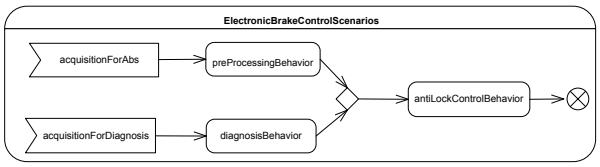

Figure 7 .System-level behaviors

From this behavioral description, Qompass-Architect generates the design model given in the next subsection. This generation relies also on some additional parameters defined in the system specification such as the activation periods of events and time budget of actions.

5.1 Design model

Figure 7 gives a schematic view of the design model of the antilock control subsystem generated by Qompass-Architect.

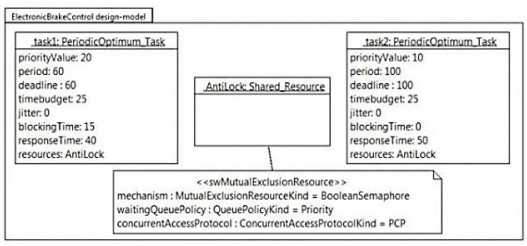

Figure 8.Design model

This model consists of two periodic tasks *taks1* and *task2* which are instances of the *PeriodicOptimum_Task* concept of the Optimum platform. The first task, *task1,* is triggered by the event *acquisitionForAbs;* consequently its period corresponds to the period of this event (*60 ms*). Besides, this task executes *preProcessingBehavior* and *antiLockControlBehavior* actions and then its execution time (*timebudget*) is the sum of the execution times of these two actions. Similarly, *task2* is triggered by the *acquisitionForDiagnosis* event and executes the *diagnosisBehavior* and *antiLockControlBehavior* actions. Qompass-Architect gives to *task1* a priority value equals to 20 and to *task2* a priority value equals to 10. The priority order in the Optimum platform is increasing as specified in Figure 3 (given by *minPriorityLevel* and *maxPriorityLevel* attributes of the *Optimum_Task*). Accordingly, *task1* has a higher priority than *task2*. These two tasks are dependent by sharing the *AntiLock* resource which corresponds to the *antiLockControlBehavior* action. Qompass-Architect chooses to implement this critical section with a PCP_Semaphore (i.e. Boolean semaphore as a mechanism, a priority-based as a waiting queue and PCP as an access protocol).

Based on these different implementations choices (priority assignment, critical section implementation, tasks number…), Timing validation is performed to verify whether this design model meets its timing requirements. Indeed, Qompass-Architect computes the blocking time depending on the implementation choices of the critical section and then computes the response time of the different tasks. The result of this validation is also given in Figure 7. From this figure, we can conclude that this model is valid from a real-time perspective since the response times of *task1* and *task2* are lower than their deadlines.

We aim at generating a correct RTEMS-specific model from this valid design model. This implementation model must conserve the timing properties while considering the characteristics of the RTEMS platform. In the following subsection, we give this RTEMS-specific model.

5.2 RTEMS-specific model

The generation of the RTEMS-specific model following our approach requires passing through two steps; feasibility tests and mapping.

For this design model, the feasibility tests step involves the different feasibility tests related to the priority aspect (c.f. section 3.1) and the shared resources aspect (c.f. section 3.2.3). For this design model, this step does not raise any feasibility concern: the design model is implementable on RTEMS. Consequently, we process the mapping step to generate the RTEMS-specific model. Figure 8 gives a schematic view of the RTEMS-specific model.

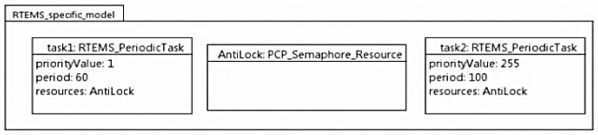

Figure 9.RTEMS-specific model

This step performs the mapping between Optimum platform resources and RTEMS resources; and the properties of these resources. Consequently, RTEMS-specific model consists of two tasks instances of *RTEMS_PeriodicTask* concept which corresponds to the appropriate type that matches the *PeriodicOptimum_Task* resource in Optimum platform. It consists also of an *AntiLock* resource instance of the *PCP_Semaphore_Resource* which defines the same implementation of critical section chosen in the design model.

For the properties, the mapping step detects that the only three properties that require mapping are the *priorityValue*, the *period* and *resources* (the other properties have a default value or they are not referenced in both platform models). This step proposes different strategies to perform the mapping of priority values (c.f. section 3.1). In Figure 8, we choose the proportional mapping of the priority values. The period values are expressed in *ticks* in RTEMS and the duration of a *tick* is configurable. We suppose here that the *1 tick* is equal to *1 ms.* This is why; the values of the period are kept the same at the implementation model. Finally, for the resource property we perform a direct mapping to keep the information that this resource (*AntiLock*) is shared between *task1* and *task2*.

5.3 RTEMS-specific model verification

In order to verify the correctness of the generated RTEMS-specific model, three properties already explained in section 4 should be fulfilled.

The first property **(P1)** is that the priority values of the different tasks in the implementation model are between the minimum and maximum priority levels allowed by the RTOS which correspond respectively to 1 and 255 for RTEMS. We can conclude from Figure 8 that this property is verified as the priority values of *task1* and *task2* are within this interval.

The second property (**P2**) is that the execution order of the different tasks is equivalent at the design and the implementation level. To this end, we transform the design model to a model instance of the verification-oriented meta-model given in Figure 4. This model (c.f. Figure 9) considers that our application is a relation of precedence among two priority levels LD1 and LD2. Each priority level references one or several tasks from the design level. In our case, we have just one task for each level since the design model does not define tasks with equal priority levels. From this model, the most important information is that, at the design level, *task1* is executed before *task2*.

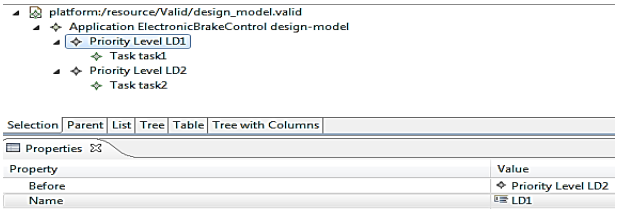

Figure 10.Design model as an instance of the verification-oriented meta-model

In the same way, we transform the RTEMS-specific model to a model conforming to the verification-oriented meta-model given in Figure 4. This model is given in Figure 10 and defines also a relation of precedence among two priority levels LI1 and LI2. Each level references also one task from the implementation model.

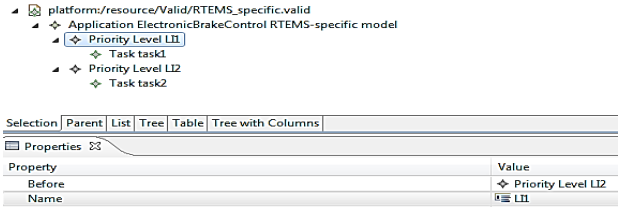

Figure 11.RTEMS-specific model as an instance of the verification-oriented meta-model

From Figure 9 and Figure 10, we conclude that the second property is also fulfilled. In fact, even the priority values at the design and the implementation levels are different; the execution order of the two tasks is conserved.

The third property (**P3**) is verified since the second property is fulfilled.

All the properties are verified and thus the generated RTEMS-specific model is correct.

5. CONCLUSION

In this paper, we propose an approach to ensure an automatic correct transition from a valid design model to an RTOS-specific model that conserves timing properties. This approach is based on two steps; the first step verifies whether the design choices are implementable on the target RTOS and the second step perform an appropriate mapping to generate the RTOS-specific model. In order to assess that the obtained RTOS-specific model is correct with respect to the design model, we identify the properties that should be verified and we propose a way to check them at the implementation level.

As future work, we aim at considering other aspects such as activation patterns, communications in a distributed platform. For each aspect, we define the additional feasibility tests and mapping strategies. Another perspective consists in refactoring the design model, when the latter is not implementable, based on the feasibility tests step feedbacks.

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] L. Sha, T. Abdelzaher, , K. E. Arzen., A. Cervin, T. P. Baker, A. Burns , G. Buttazzo, M. Caccamo, J. Lehoczky, and A. K. Mok. Real time scheduling theory: A historical perspective. Real-Time Systems 28(2/3): 101155. 2004.

[3] C. Mraidha, S. Tucci Piergiovanni and S. Gerard: Optimum: a MARTE-based methodology for schedulability analysis at early design stages. ACM SIGSOFT Software Engineering Notes 36(1): 1-8 (2011)

[4] F. Singhoff, J. Legrand, L. Nana, and L. Marcé. Cheddar: a Flexible Real Time Scheduling Framework. International ACM SIGADA Conference, Atlanta, November 2004.

[5] M. GonzAlez Harbour, J.J. GutiCrrez Garcia, J.C. Palencia GutiCrrez, and J.M. Drake Moyano. MAST: Modeling and Analysis Suite for Real Time Applications. Real-Time Systems, 13th International Euromicro Conference. Delft, June 2001.

[6] The Open Group Base Specifications, Portable Operating System Interface (POSIX), ANSI/IEEE Std 1003.1, 2004.

[7] OSEK Group. OSEK/VDX Operating System Specification. http://www.osek-vdx.org.

[8] T-Engine Forum. µITRON 4.0 Specification, July 2010. http://www.t-engine.org

[9] R. Yemhalli. Real-time operating systems: An ongoing review. In Work-In-Progress Sessions. The 2lrl IEEE Real-lime System Symposium (RTSSWIPOO), Orlando, Florida, November 2000.

[10] H. Takada, Y. Nakamoto, and K. Tamaru, "The ITRON Project: Overview and Recent Results", 5th International Conference on Real-Time Computing Systems and Applications (RTCSA), pp.3-10, Oct. 1998.

[11] F. Thomas, J. Delatour, F. Terrier, and S. Gerard. Toward a framework for explicit platform-based transformations. In Proceeding of the 11th IEEE Symposium on Object Oriented Real-Time Distributed Computing (ISORC). Orlondo, Florida, USA, May 2008.

[12] Object Management Group, UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems, Object Management Group, Inc., September 2010, OMG document number: ptc/2010-08-32

[13] 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.

[14] R. Mzid, Ch. Mraidha, J-P. Babau, M. Abid. A MDD Approach for RTOS Integration on Valid Real-Time Design Model. The 38th Euromicro Conference On software Engineering and Advanced Applications (SEAA'12), Cesme, Izmir, Turkey, September 2012.

[15] Object Management Group, Object Constraint Language (OCL). Object Management Group, Inc., May 2006, OMG document number: formal/06-05-01

[16] RTEMS C Users Guide. Edition 4.6.5, for RTEMS 4.6.5. August 2003.

[17] Mark H. Klein, Th. Ralya, B.Pollak, R. Obenza and M.Gonzalez Harbour. A Practitioner's Handbook for real-Time Analysis. Guide to Rate Monotonic Analysis for Real-Time Systems. Kluwer Academic Publisher. ISBN 0-7923-9361-9. p. 5-30.