A MDD Approach for RTOS Integration on Valid Real-Time Design Model

Rania Mzid*, Chokri Mraidha*, Jean-Philippe Babau† and Mohamed Abid‡
* *CEA, LIST, Laboratory of model driven engineering for embedded systems*
Point Courrier 174, Gif-sur-Yvette, 91191, France
Email: {rania.mzid, chokri.mraidha}@cea.fr
†*Lab-STICC, UBO, UEB, Brest, France*
Email:Jean-Philippe.Babau@univ-brest.fr
‡ *CES Laboratory, National school of engineers of Sfax, Sfax, Tunisia*
Email: Mohamed.Abid@enis.rnu.tn

***Abstract*—The transition from the design model to the implementation model is a critical phase in Real-Time Embedded Systems development process. Indeed, this model must conserve functional and non-functional requirements of the design model on the target execution platform. In this paper, we propose a two-steps approach based on an explicit description of two types of platform: the abstract platform used at the design level to validate the different design choices, and the concrete execution platform. 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.**

***Keywords*-Real-Time Embedded Systems; MDD; Real-Time Validation; Abstract platform; Concrete platform;**

I. Introduction

In order to overcome the increasing complexity of Real-Time Embedded Systems (RTES), researches are moving toward a rise in level of abstraction using Model-Driven- Development (MDD) [1]. MDD introduces intermediate models starting from the model of the specification and going to the model of the implementation, while passing through the design model (see Figure. 1). The specification model concentrates on functional and behavioral description of the system. At design level, model concentrates on architectural concerns, focusing on concurrency by introducing tasks and by defining a scheduling policy. Whereas implementation models integrate technological concerns like, in our case, specific Real-Time Operating System (RTOS).

Validation is a key point of RTES: at every modeling level a timing validation is performed. At specification level an analysis of time budgeting [2] may be applied. At design level, scheduling analysis [3] and performance analysis [4] validate the design choices in terms of non-functional requirements. Accurate analysis may be performed at implementation, to verify whether the timing properties are met on the real platform.

To enable timing validation, a convenient level of abstraction for the underlying platform must be considered. At specification level, analysis does not make any assumptions about resource limitations. The corresponding platform, called *virtual platform* is considered as an ideal platform. At design level, the topology of the hardware architecture is assumed to be known. For software aspect, the *abstract platform* is considered as a generic platform that offers unlimited software resources (tasks and communication mechanisms). At implementation level, the *concrete platform* model integrates the target RTOS technological constraints.

In order to reduce development cost, one main objective of generative MDD approaches is to apply the correct-by-construction paradigm. One challenge is then to automate a correct transition from design to implementation models, which preserves timing properties.

For platform aspect, refinement of a design model to an implementation model corresponds to a mapping between the abstract resources and the concrete ones. However, the consequent implementation model may not match the original design model. This scenario happens in the case where the concrete platform is too restrictive with regard to the abstract one or when incoherent correspondence between properties of abstract and concrete resources occurs. 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 a concrete platform.

To tackle this issue, we propose to add a feasibility tests step and a mapping step between the design and the implementation phases. The purpose of the first step is to help the designer detecting early potential refinement problems and their sources. The second step aims at keeping the design model independent of a concrete platform. The proposed approach is based on an explicit description of the abstract platform, used for validation, and the concrete one used for implementation. These platforms are decribed in models that are independant of the application. The platform models play a primary role in performing the feasibility tests and the mapping to obtain an implementation model preserving the timing properties of the design.

The paper is organized as follows. Section 2 gives a

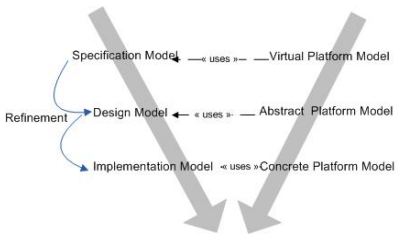

Figure 1. Platform levels from a validation point of view in a MDD approach

background on design model and related timing validation. Section 3 presents the platform modeling framework and its main principales. Section 4 explains in details the approach to obtain a valid implementation model from the design model. Section 5 illustrates on an example the proposed approach. Section 6 discusses some related works and Section 7 concludes the paper and presents some perspectives.

II. Design Model and Optimum overview

In this paper we assume that timing validation is performed at the design level using Optimum [5]. The Optimum methodology introduces timing validation from the specification level in order to guide the design of the concurrency model that satisfies the timing constraints. This methodology is based on two phases:

- the first phase considers as input the high-level functional model, to which time budgets have been allocated to functions and assessed with regard to the application timing requirements;
- the second phase deals with software architecture exploration. In that phase, Optimum seeks to produce a software architecture model that satisfies timing requirements. This objective achievement is guided by scheduling analysis. At the end, the obtained concurrency model consists of a set of tasks characterized by their priorities. Each task is allocated to a specific hardware component (execution node or communication media). It is important to notice that, in order to generate and validate the design model, Optimum considers *implicitly* some assumptions on the software platform such as scheduling policy, task activation and communication mechanisms.

In this paper, the design model is based on a set of independent tasks $\tau = \{\tau_1, \tau_2, , \tau_n\}$ executing the different functions of the system. We assume that all tasks are scheduled according to their priority. So each task $\tau_i \in \tau$ is characterized by its priority P_i and runs at a base period T_i. Besides, we suppose that the hardware architecture corresponds to a single execution node (mono-processor architecture).

The different design choices ensure that the current design model translates the system specification and fulfills its timing constraints under the assumptions made by Optimum and that are related to the software platform (scheduling policy, task activation and communication mechanisms). These choices are validation-oriented and not implementation-oriented.

From this correct model (design model), the 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 the abstract platform while execution is based on the concrete one.

III. Platform Modeling Framework

In this section, we present the proposed platform modeling principles. Then we give excerpts of the abstract and concrete platform models.

A. Platform modeling

One key point of the proposed approach is to ensure a correct deployment of the design model on a specific RTOS. In that context, [6] presents 8 approaches, grouped into 3 groups: embedded, implicit and explicit platform representations. It appears that embedded and implicit representations are faster to implement code generation. But they are less appropriate to target several platforms. Consequently, an explicit representation of the concrete platform appears to be more suited to our needs. In the explicit approach, the target platform is described as a standalone model and is an input for the deployment model transformation.

Several works gave interest in the definition of Domain Specific Languages (DSLs)[6][7] [8] or profiles [9][10] for platform description. However, Software Resource Modeling (SRM) sub-profile of the MARTE standard [9] seems to be the best choice. It specifies a set of modeling artifacts that can be used to describe the resources and services provided by any RTOS. The use of SRM as a pivot language to describe both abstract and concrete platforms allows to capture the semantic of the different concepts defined in the platform models and thus enables an automatic transformation between the design and the implementation models.

The abstract platform represents the set of software resources necessary to perform validation. We propose the following principles for its definition:

- the abstract platform defines all the resource concepts necessary for validation, here schedulability analysis;
- the abstract platform is independent from a particular concrete platform but it shares the same concepts in order to prepare deployment stage;
- the abstract platform provides all possible concepts provided by concrete RTOSs to avoid making concrete platform specific assumptions that would limit the designer among design choice options;

The abstract platform may be viewed as an ideal platform for validation purpose, that contains all the possible concepts provided by a set of concrete platforms.

A concrete platform corresponds to a specific RTOS or to a standardized RTOS API, like POSIX [11] or OSEK [12]. The numerous RTOS or standards share common concepts but with specific features.

In the following subsections, we present the abstract and the concrete platform models considered in this paper. The abstract platform is the Optimum platform since we assume that timing validation at the design level is performed with Optimum. RTEMS [13] has been chosen as a concrete platform. Nevertheless, our approach is applicable for any timing validation tool and any RTOS. Platform models conform to the meta-model UML enriched with SRM profile. For the present work, platform modeling based on SRM, concentrates on features used for schedulability analysis:

- information on task and their activation mechanism;
- information related to scheduling: scheduling policy and priorities;
- applicative timing constraints;
- timing precision;

B. Abstract platform model for Optimum

The Optimum platform acts as the abstract platform. The abstract platform model characterizes the set of RTOS concepts which are mandatory to correctly create the design model with respect to schedulability analysis. For reason of space, we present just an excerpt of the Optimum platform model, however it contains many other concepts such as shared resources or communication mechanisms for multiprocessor environments.

The necessary RTOS concepts are encapsulated in three concepts (see Figure. 2) which are *Optimum_Task, Optimum_Scheduler* and *PeriodicOptimum_Task*. Each of these concepts is annotated with appropriate SRM stereotype holding its corresponding semantics. For example, *Optimum_Task* class is identified as *"swSchedulableResource"* to mention that it serves to execute application functions. We characterize each concept by the properties that are *needed by Optimum to perform timing validation*. The *Optimum_Scheduler* has two properties which are *policy* and *isPreemptive*. These properties represent the information of the platform scheduler that is necessary for validation.

For each concept, we give a default value to the properties that characterize the platform and are independent of the application. For the *Optimum_Task* concept the *maxPriorityLevel* and *minPriorityLevel* attributes correspond respectively to the maximum and minimum priority level authorized by the Optimum platform. 0 is the the highest priority of an *Optimum_Task*. In addition, *isPriorityVary* and *isUnique* attributes of the *Optimum_Task* are used to mention respectively that an Optimum task can have more than one

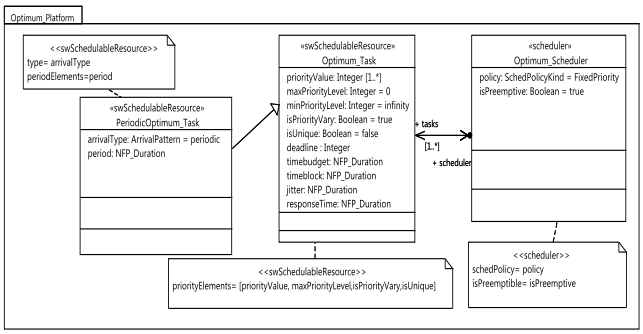

Figure 2. Excerpt of the Optimum platform model

priority value and that two Optimum tasks can have the same level of priority.

Other properties of the *Optimum_Task* concept do not have default values. The property *priorityElements* of *"swSchedulableResource"* corresponds to *priorityValue* attributes of the *Optimum_Task*. *PeriodicOptimum_Task* is a particular type of *Optimum_Task* with a periodic activation pattern. This is illustrated by a default value equal to *"periodic"* of the *arrivalType* property of the *PeriodicOptimum_Task*.

C. RTOS platform model (RTEMS Operating System)

As for abstract platform, we characterize for the concrete platform: tasks, activation and priorities as it is shown in Figure 3. A RTEMS task is represented with *RTEMS_Task* class annotated with *"swSchedulableResource"*. The *priorityValue* property of a task varies between 1 and 255 which corresponds respectively to *maxPriorityLevel* and *minPriorityLevel* properties. A RTEMS task may have more than one priority value, so that the *isPriorityVary* property is set to *true*. Furthermore, two RTEMS tasks can have the same priority value, this is the reason why *isUnique* property default value is set to *false*.

We define also an *RTEMS_Clock* concept in the RTEMS model and we annotate this class with *"swTimerResource"*. A clock is characterized by its *tick* which is represented by an attribute in the *RTEMS_Clock* class. The value of the clock tick is a characteristic of the RTOS and corresponds to *durationElements* property in the stereotype *"swTimerResource"*. Indeed, we can associate to the RTEMS clock an infinite number of timers. A *RTEMS_Timer* concept is identified as an *"alarm"* and has a *period* and an *arrivalType* as well. Although the concept of periodic task does not exist natively in RTEMS, we can define a pattern that describes the realization of the periodic task with the basic concepts of the RTEMS core. Figure 4 shows a possible pattern to realize a periodic task in RTEMS. A class named *RTEMS_PeriodicTask* is created and identified as *"swSchedulableResource"*. This class has three properties: clock, timer and task which are defined respectively as

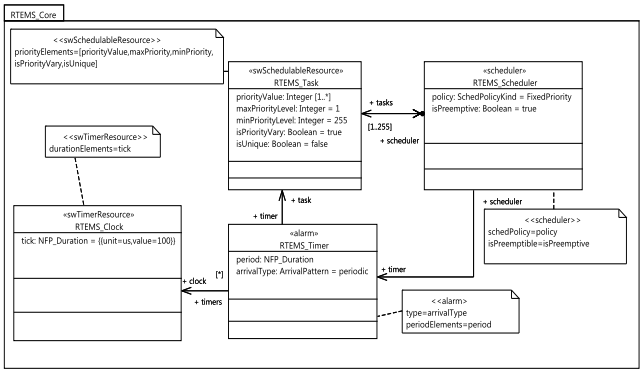

Figure 3. Extract of RTEMS core model

RTEMS_Clock, *RTEMS_Timer* and *RTEMS_Task*. Eventually, the priority of the *RTEMS_PeriodicTask* corresponds to the priority of its task property. Besides, the period of the latter and the type which is *"periodic"* corresponds to the period and type of its timer property. Due to space limitation, we have presented only an excerpt of the RTEMS model.

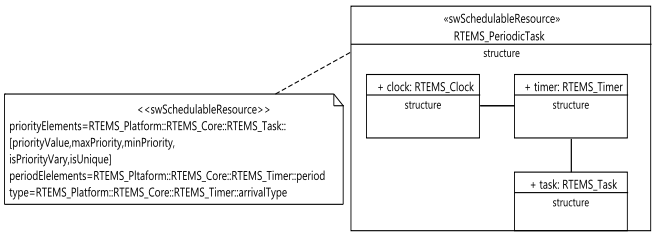

Figure 4. Periodic task pattern for RTEMS

IV. Toward a Valid Implementation Model

A valid implementation model is a model that preserves the properties of the application on the concrete execution platform. This section details our two-steps approach for transformation of the design model toward a valid concrete platform specific implementation model. Figure. 5 gives a schematic overview of the involved models, tests and transformations. *The feasibility tests step* which is responsible for

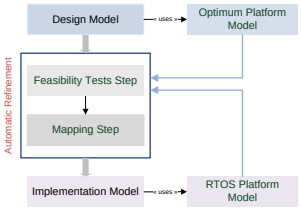

Figure 5. Overview of the proposed approach

checking the feasibility of the design model on the concrete platform. It provides feedbacks to the designer, when the model is non-feasible, to help him in detecting the source of the problem. The second step is the *mapping* that permits to obtain the appropriate implementation model.

A. Feasibility Tests Step

The feasibility tests step uses the abstract and concrete platform models described in the previous section and provides three possible outputs:

- *an error* when the input design model is not implementable on the RTOS. An error informs the designer about the source of the problem. For example, if the number of priority levels used in the design model is higher than the number allowed by the RTOS. In that case, an error is generated with an associated message indicating that the number of priority levels provided by this RTOS is not sufficient to implement the input design model. Algorithm 1 illustrates this test;
- *a warning* when the design model is implementable on the RTOS with adaptation of the resources properties. This adaptation may affect the correctness and the timing performance of the application. Hence it necessitates a fresh schedulability analysis. For example, if at design level, two tasks have the same priority level and at concrete level, the RTOS does not allow same priority level. The input design model has to be refactored by affecting consecutive priority levels to the two tasks and adapting, if necessary, the priority levels of others tasks. Schedulability analysis is re-performed since timing behavior of the two tasks change. In that situation, this step generates a warning to highlight this problem and a design refactoring operation proposal;
- *no problem* when there is no feasibility problems then a mapping step can be performed without any change.

The set of feasibility tests to perform depends on the complexity of the design model : the number of used RTOS concepts. In Table. I, we give the list of the feasibility tests performed under the assumptions made on the design model in this paper. The list is not exhaustive in the general case (usage of communication mechanisms and protection protocol for shared resources).

B. Mapping Step

If the design model is implementable on the target RTOS, we perform the mapping step. The purpose of this step is to generate the implementation model. The latter should preserve the high level specification and should ensure the respect of timing requirements validated at the design level. To this end, this step performs two levels of mapping: the mapping of concepts and the mapping of the properties' values of these concepts. Algorithm 2 illustrates this mapping step. For each instance in the design model which is of type *T* that corresponds to a resource in the Optimum platform, this algorithm generates an instance in the implementation model which is of type *T'* that corresponds to a resource in the RTOS model. This generation is performed if and only

Algorithm 1: Test Number Priority Levels

Input:
M_a: Design Model of the application
M_{PA} : Abstract platform model
M_{PC} : Concrete platform model
Output:
Verdict S = {E, W, NP}; E: Error, W: Warning, NP: No problem
Notations:
Type (i) : the type of the instance *i*
S(c): the stereotype to which the class *c* is conform
Properties(s): properties of the stereotype *s*
$PriorityLevels(M_{PA})$: list of priority levels used in the design model
begin
  $S \longleftarrow NP$
  for $i \in M_a$ ***such as*** *S(Type (i)) ="swSchedulaleResources"* **do**
    $PrioValue(i) \longleftarrow val(p)$ **such as** p = "priorityElements[1]"∈ Properties (S(Type (i)))
    if $PrioValue(i) \notin PriorityLevels(M_{PA})$ **then**
      $add(PrioValue(i), PriorityLevels(M_{PA}))$
  $NbrLevels_{M_{PA}} \longleftarrow$ **sizeof**$(PriorityLevels(M_{PA}))$
  if $it \ni c \in M_{PC}$ ***such as*** *S(c) ="swSchedulaleResources"* **then**
    $MaxPrioLevel \longleftarrow$ val(p) **such as** p = "priorityElements[2]"∈ Properties (S(c))
    $MinPrioLevel \longleftarrow$ val(p) **such as** p = "priorityElements[3]"∈ Properties (S(c))
    $NbrLevels_{M_{PC}} \longleftarrow |MaxPrioLevel - MinPrioLevel|$
  if $NbrLevels_{M_{PA}} > NbrLevels_{M_{PC}}$ **then**
    $S \longleftarrow E$
  return S

Table I
LIST OF FEASIBILITY TESTS

Feasibility Test	Description
Periodic Task	Verify if the concept of periodic task exist or not in the concrete platform
Timer Granularity	Test if the timer granularity of the RTOS is poor compared to the values of timing requirements at the design level
Task Number	Test if the number of application tasks authorized by the RTOS is lower than the number of tasks defined in the design model
Scheduler	Test the adequacy of types between the abstract and the concrete scheduler
Variable Priority Level	Verify if the RTOS support the variation of the priority of a task
Equal Priority Level	Verify if the RTOS support that two tasks have the same priority level

if *T* matches *T'*. Once the instance which is of type *T'* is generated we have to perform the second level of mapping ($Mapping_p$ in Algorithm 2) for the properties that match.

In this paper, the matching is ensured by the use of the pivot language SRM. So we suppose that two types *T* and *T'* match *if and only if*:

- they have the same applied stereotype from SRM;
- the properties of the the applied stereotype that reference *T* attributes, reference also *T'* attributes.

In addition, we assume that two properties match *if and only if*:

- they are referenced by the same property from the applied stereotype;
- they do not have a default value.

In Figure 6, *Optimum_Task* and *RTEMS_Task* match because they have the same applied steryotype *"swSchedulableResource"* and the property *"priorityElements"* of this stereotype references attributes of *Optimum_Task* and *RTEMS_Task*. For the properties, we can see that the only attribute that is referenced by the property *"priorityElements"* of *"swSchedulableResource"* and that has not a default value is *PriorityValue*. Thus an appropriate mapping of this attribute must be considered. We give more details about this mapping through an example in the following section.

Algorithm 2: Mapping Step

Input:
M_a: Design Model of the application
M_{PA} : Abstract platform model
M_{PC} : Concrete platform model
Output:
M_{impl}: Implementation model of the application specific to the concrete platform *PC*
Notations:
Type (i): The concept of the platform that types the instance *i*
Property (c): The set of the properties of the platform concept *c*
begin
  for $i \in M_a$ ***such as*** $Type(i) \in M_{PA}$ **do**
    $Createinstance(i', M_{impl})$ **such as** $Type(i)$ ***matches*** $Type(i') \in M_{PC}$
    for $p \in Property(Type(i))$ **do**
      if $it \ni p' \in Property(Type(i'))$ ***such as*** p' ***matches*** p **then**
        $value_p(i') \longleftarrow Mapping_p(value_p(i))$ For the Concrete Platform *PC*
  return M_{impl}

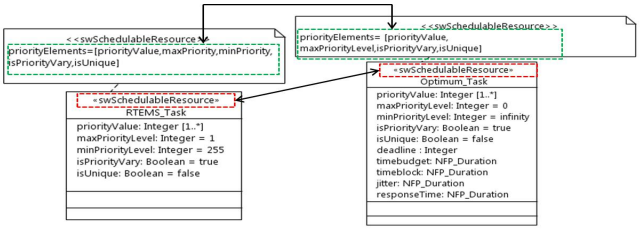

Figure 6. Matching using SRM

V. CASE STUDY

In this section, the proposed approach is illustrated through a case study. Firstly, the design model satisfying timing requirements is presented. Then the transformation of this design model to an implementation model specific to RTEMS is detailed.

A. Design Model

We consider a design model that consists of four periodic tasks. These tasks are instances of *PeriodicOptimum_Task*. Figure 7 depicts the model generated by Optimum (for matter of space, a tabular representation is given instead of the model diagrams).

Optimum gives the highest priority which is 0 to the task *T1*, it assigns the same priority level to tasks *T2* and *T3*

Tasks	timebudget	deadline	period	jitter	Blocking Time	priorityValue	responseTime
T1	10 ms	100 ms	100 ms	0 ms	0 ms	0	10 ms
T2	12 ms	200 ms	200 ms	0 ms	0 ms	1	31 ms
T3	9 ms	200 ms	200 ms	0 ms	0 ms	1	31 ms
T4	10 ms	300 ms	300 ms	0 ms	0 ms	30	41 ms

Figure 7. Example of a design model generated using Optimum

and gives to *T4* a priority value equals to 30. In addition, based on some information such as *timebudget*, *jitter*, etc, Optimum computes the response time for each task in order to verify whether its choices, here priority value of each task, respect the timing requirements. As shown in Figure 7, the response time of the different tasks is lower than their respective deadline. Consequently, the design model generated by Optimum meets its timing constraints in terms of schedulability.

B. Refinement of the design model to an RTEMS implementation model

The refinement of the previous design model to an RTEMS implementation model, following our approach, does not raise feasibility concerns. Consequently we apply the mapping algorithm defined in the previous section to our design model. This algorithm generates four tasks *T1'*, *T2'*, *T3'* et *T4'* which are instances of *RTEMS_PeriodicTask*. *RTEMS_PeriodicTask* is the appropriate matching type since its satisfies the matching conditions explained in section IV-B. For the properties, the mapping algorithm 2 detects that an appropriate mapping of the period and the priority values must be considered.

At the design level, the period values of the different tasks are expressed in *ms*. The by-default tick in the RTEMS model is equal to *100 μs*. To ensure a correct mapping of the period values, we propose a timing unity adaptation detailed in algorithm 3. This algorithm is defined to determine the appropriate value of period for the concrete platform. It considers as input only the concrete platform model (here the RTEMS model) and the instance for which we want to determine the value of period. The concrete platform model gives the clock tick value of the RTOS.

Algorithm 3: $Mapping_{period}(PeriodValue(i_s))$

Input:
M_{PC} : Concrete platform model
Output:
$PeriodValue'(i_s)$: The converted period value for the concrete platform *PC*
Notations:
$unit_{PA}$: The unit of the period of the instance i_s
$unit_{PC}$: The unit of the clock tick of the concrete platform *PC*
α : The coefficient / $unit_{PA} = \alpha\ unit_{PC}$
TickValue: The value of the clock tick of the corresponding concrete platform *PC*
begin
$unit_{PA} \longleftarrow getPeriodUnit(i_s)$
$unit_{PC} \longleftarrow getClockTickUnit(M_{PC})$
$\alpha \longleftarrow computeCoefficient(unit_{PA}, unit_{PC})$
$TickValue \longleftarrow getClockTickValue(M_{PC})$
$periodValue'(i_s) \longleftarrow (periodValue(i_s) * \alpha)/valueTick$
return $PeriodValue'(i_s)$

On the other hand, if the clock tick of the target RTOS is configurable (which is the case for RTEMS for example), our approach detects the minimum timing precision in the design model (period value of *T1* which is equal to 100 *ms* in the example) and adjusts the clock granularity to this value (i.e. RTEMS clock tick will be equal to 100 000 μs). Thus the period value of the generated task *T1'* will be equal, in that case, to 1 *tick*. This can be seen as an optimization for the implementation since the interruption of the ticks management will run at minimum frequency and thus the processor load due to the ticks management is reduced.

In order to generate valid implementation models from a priority point of view, the mapping algorithm of the priority values must ensure that the used priority values at the implementation level are between the maximum and the minimum priority level allowed by the RTOS (maxPriorityLevel and minPriorityLevel of the *RTEMS_Task* attributes in the RTEMS model). Besides, the execution order of the different tasks defined at the design level is preserved at the implementation level. Our approach proposes four strategies to perform the mapping of the priority values. We present briefly 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 available *minPriorityLevel* 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 (from *minPriorityLevel* to *maxPriorityLevel*). 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 (i.e. maxPriorityLevel = infinity).

The selection of the strategy to use depends on the designer intention and the capacity of the strategy to generate valid implementation model i.e. in our example, the *direct mapping* can not be used because the task *T1*, at the design level, has a priority value equal to 0 which is not authorized in RTEMS as 1 is the higher priority level. So if the

designer chooses this strategy of mapping our approach generates an error in order to inform him that the use of this strategy does not guarantee a valid RTEMS model and proposes the other mapping strategies. For reason of space, we present below just the *linear mapping*. Algorithm

Algorithm 4: $Linear\_Mapping_{priority}(PriorityValue(i_s))$

Input:
M_a: Design Model of the application
M_{PA} : Abstract platform model
M_{PC} : Concrete platform model
i_s: Instance in the design model for which we want to determine the corresponding priority value
Output:
$PriorityValue'(i_s)$: The converted priority value for the concrete platform *PC*
Notations:
Type(i): The concept of the platform that types the instance *i*
Higher-levels(i): priority levels higher than the current priority value *PriorityValue*
begin
  $highestPrioPA \longleftarrow getHighestPriorityLevel(M_{PA})$
  $lowestPrioPA \longleftarrow getLowestPriorityLevel(M_{PA})$
  if $(highestPrioPA < lowestPrioPA)$ **then**
    for $i'_s \in M_a / Type(i'_s) = Type(i_s)$ **do**
      if $(PriorityValue(i'_s) < PriorityValue(i_s))$ **then**
        if $(PriorityValue(i'_s) \notin Higher - levels(i_s))$ **then**
          $add(PriorityValue(i'_s), Higher - levels(i_s))$
  else
    for $i'_s \in M_a / Type(i'_s) = Type(i_s)$ **do**
      if $(PriorityValue(i'_s) > PriorityValue(i_s))$ **then**
        if $(PriorityValue(i'_s) \notin Higher - levels(i_s))$ **then**
          $add(PriorityValue(i'_s), Higher - levels(i_s))$
  $highestPrioPC \longleftarrow getHighestPriorityLevel(M_{PC})$
  $lowestPrioPC \longleftarrow getLowestPriorityLevel(M_{PC})$
  if $(highestPrioPC < lowestPrioPC)$ **then**
    $PriorityValue'(i_s) \longleftarrow highestPrioPC + Sizeof(Higher - levels(i_s))$
  else
    $PriorityValue'(i_s) \longleftarrow highestPrioPC - Sizeof(Higher - levels(i_s))$
  return $PriorityValue'(i_s)$

4 considers as inputs the platform models (abstract and concrete) and the design model of the application. For each task instance in the design model, this algorithm computes the adequate priority value for the concrete platform, based on some information presented in the platform models. For example, *maxPriorityLevel* and *minPriorityLevel* are used to determine the order of the priority in the platform and serves to determine the appropriate value.

Figure 8 and Figure 9 represent two examples of RTEMS implementation models which can be obtained by applying our approach. The RTEMS model given in Figure 8 is obtained by applying the mapping of the period values (algorithm 3) and by performing the *linear mapping* (algorithm 4). However, the model given in Figure 9 is obtained by performing the *proportional mapping* and by configuring the RTEMS clock tick for optimisation.

Tasks	period	priorityValue
T1'	1000 tick	1
T2'	2000 tick	2
T3'	2000 tick	2
T4'	3000 tick	3

Figure 8. RTEMS implementation model with linear priority mapping

Tasks	period	priorityValue
T1'	1 tick	1
T2'	2 tick	127
T3'	2 tick	127
T4'	3 tick	255

Figure 9. RTEMS implementation model with proportional priority mapping and period mapping optimization

These two models correspond to an implementation model with properties values (priority and period) equivalent to those in the design model while considering RTEMS platform characteristics. For instance, *T1'* at the implementation level has a period equals to 1000 *tick* which corresponds exactly to 100 *ms* at the design level. In addition, from a priority point of view, these two models are valid. In fact, the priority values in Figure 8 and Figure 9 are between 1 and 255 which correspond respectively to the maximum and minimum priority level allowed by RTEMS. The execution order of the different tasks defined at the design level is also preserved in both models.

VI. Related Work

Platform consideration during the development cycle to transform a system specification to an implementation has frequently been raised by several works. MoPCoM approach [14] is a co-design methodology that represents a refinement of the MDA Y-Chart [15] and is based on OMG standards. This methodology uses UML and MARTE [9] to represent the different models and SysML [16] 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. At each level a convenient granularity of platform model is considered to allow analysis. Even if this work is in a co-design context, we have the same three different levels: one specification level, one validation-oriented level and one implementation level.

In a software development context, the authors in [17] consider a platform as an RTOS and propose a generative process in order to transform an application deployed on one platform to another based on an explicit description of the

latter using SRM. This work makes the assumption that the deployment is always possible. In the same way, the author in [6] proposes a deployment process of an application on a software platform (RTOS). This process considers an explicit description of the RTOS using a Domain Specific language (DSL) called RTEPML and focuses on defining generic transformations to automate the process. Compared to [17], this approach claims the necessity to perform the feasibility tests in particular a test to verify the availability of a concept on the target platform before the deployment. Moreover, these two approaches define generic transformations. No special attention is paid to the characteristics of the different resources of the RTOS and its influence on the validity of the obtained model. In our approach, we define and implement the necessary tests to perform and we add a specific mapping step to integrate specific platform characteristics. In [18], the author proposes an MDA approach for distributed applications. In the paper, the author promotes an explicit definition of an abstract platform to get reference architecture for the design of Platform Independent Model (PIM) for distributed applications. Like in our approach, this work addresses the notion of abstract platform and proposes an explicit representation of the latter. However, it is interested on distributed applications and does not focus on Real-Time Embedded Systems.

VII. Conclusion and Future Work

In this paper, we propose a MDD approach to automate the transition from design model, which is assumed to meet its real-time constraints, to an implementation model specific to a RTOS. This approach aims at providing a generic and automatic guidance framework to the designer in order to guarantee that the implementation model conserves non-functional properties of the design model on the target platform. The proposed approach is based on an explicit definition of two types of platforms: the abstract platform used at the design level to perform a timing analysis, and the concrete platform (RTOS) providing execution services for the application.

As future work, our aim is to consider and exploit the behavior of the application and the services provided by the RTOS to provide a full code generation from the design model while ensuring the respect of the execution semantic of the latter at the implementation level.

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] C. Ferdinand, R. Hechmann, D. Kastner, K. Richter, N. Feiertag, M. Jersak. Integration of Code-Level and System-Level Timing Analysis for early Architecture Exploration and Reliable Timing Verification. In Proceeding of the Embedded Real Time Software and Systems (ERTS2 2010). Toulouse, France.

[3] 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.

[4] M. Woodside, G. Franks, and D. Petriu. The future of software performance engineering. Future of Software Engineer

[5] 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)

[6] 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.

[7] SAE, Architecture Analysis & Design Language (AADL), AS-5506, 2004.

[8] Tivadar Szemethy : Domain-Specific Models, Model Analysis, Model Transformations. Phd Thesis, University of Vanderbilt, Nashville, Tennessee, USA, May 2006.

[9] 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

[10] P. Kukkala, J. Riihimki, M. Hmlinen and K. Kronlf : UML 2.0 Profile for Embedded System Design, Automation and Test in Europe Conference (DATE 2005), pp. 710-715,March 2005.

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

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

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

[14] 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).

[15] OMG: Mda guide version 1.0.1. Technical report, Object Management Group (2003).

[16] OMG, Systems Modeling Language version 1.3, formal/2012-06-01, 2012

[17] 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.

[18] J. Almeida, Model-driven design of distributed applications, in On the Move to Meaningful Internet Systems 2004: OTM 2004 Workshops. Springer, 2004, pp. 854865.