RTOS-Aware Refactoring for Portable Real-Time Design Models

Rania Mzid[a,b,c], Chokri Mraidha[a], Jean-Philippe Babau[b] and Mohamed Abid[c]
[a] 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
[b] Lab-STICC, UBO, UEB, Brest, France
Email: Jean-Philippe.Babau@univ-brest.fr
[c] CES Laboratory, National School of engineers of Sfax, University of Sfax, Tunisia
Email: Mohamed.Abid@enis.rnu.tn

***Abstract*—In a model-driven development context, the refinement of a Real Time Operating System (RTOS) independent design model of a real-time application to a RTOS specific implementation model is a non-trivial task. Indeed, the different design choices made to guarantee the application timing properties are not always implementable on the target RTOS. In this paper, we propose a pattern-based approach to perform the refactoring of the real-time design model when a deployment problem appears. This refactoring guarantees the deployment of the refactored design model and the respect of its timing properties. This paper explains in details two examples of patterns which are the Equal Priority Merge Pattern (EPMP) and the Distinct Priority Merge Pattern (DPMP). The automation of the proposed approach allows showing its applicability on a robotic case study.**

***Index Terms*—Real-Time Embedded Systems, software development, Model-Driven Development, Real-Time Verification, Design Model, RTOS-Specific Model, Patterns, Refactoring.**

I. Introduction

In order to increase the productivity during the development process of Real-Time Embedded Systems (RTES), Model Driven Development (MDD) [1] promotes a rise in level of abstraction by introducing models from the specification to the implementation passing through the design (see Figure 1). Each model focuses on a particular aspect; the specification model gives a functional and behavioral description of the application. At the design level, architectural concerns are introduced to express concurrency and data dependency. The implementation model introduces technological concerns related to the execution platform i.e. the Real-Time Operating System (RTOS).

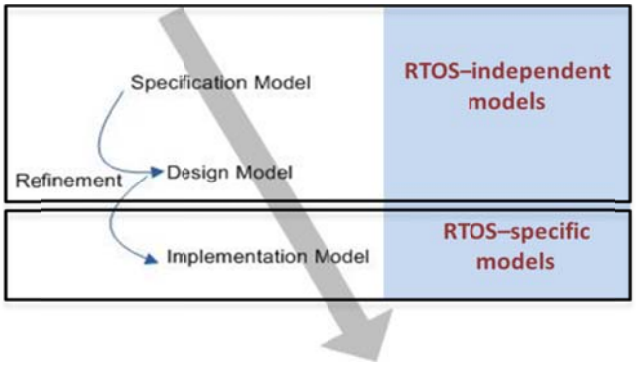


Figure 1. Model-based flow

In a MDD context, Model-Driven Architecture (MDA) [2] standardized by the OMG, recommends system development along the lines of the Y-Chart approach [3]. Thus, the real-time application is described in a design model independent from any particular RTOS. Then, this design model is deployed onto a RTOS to produce the implementation model. Indeed, as shown in Figure 2, the specification model depicts a functional graph defining a set of transactions to capture the system behavior. Each transaction is defined by its period P_i and has a deadline D_i that represents the maximum time value allowed for the associated transaction to be executed. Each function is characterized by its worst-case execution time c_i. From this specification model, the design model identifies the abstract tasks (T_i) implementing the application functions and the software resources (R_i) that can be shared between the tasks. The deployment phase corresponds thus to a mapping between the abstract resources and the running ones (the concrete ones) provided by the considered RTOS.

© 2014 ACADEMY PUBLISHER
doi:10.4304/jsw.9.7.1737-1748

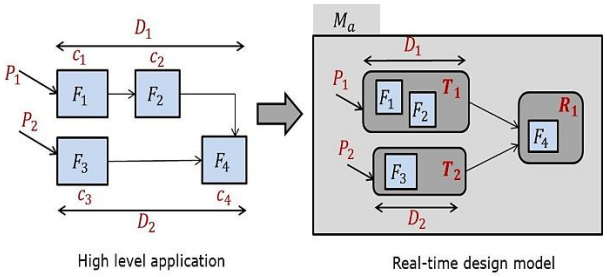


Figure 2. Real-time design model

Timing verification activities, at the design level, aim to verify whether the different tasks complete their executions within the time limit specified by the real-time application i.e. the deadline. This verification requires an abstraction of the underlying platform. To this end, at the design level, execution nodes and communication media are supposed to be known. In this paper, we consider only single-processor systems. In addition, software assumptions related to the target RTOS such as scheduling policy, priority order, etc. are *implicitly* considered in order to keep RTOS-independence at this level. There is indeed a wide variety of RTOSs [4]; some are compliant to a specific standard (e.g. POSIX [5], OSEK [6]), some are commercial or free. These RTOSs share common concepts but with specific features. From these considerations, the different software assumptions considered at the design level for the timing verification may be not supported by the target RTOS. In that case, the design model is said to be not "implementable" on that RTOS, and a new "implementable" RTOS-specific design model shall be found by the designer. This time consuming approach has several drawbacks among which we can mention the portability of the new design model.

To tackle this issue, we focus in this paper on an automatic refactoring of the design model when a deployment problem appears. This refactoring is based on a set of software patterns. Each pattern defines a way to change the original design model with the aim to solve the deployment problem. The resulting model after applying a pattern must guarantee two points: (1) portability i.e. still independent from the target RTOS, (2) the respect of timing properties. In the present paper, we deal with two examples of patterns; *Equal Priority Merge Pattern (EPMP)* and *Distinct Priority Merge Pattern (DPMP).* For each pattern, we explain the corresponding deployment problem and we describe the proposed solution.

The remainder of this paper is organized as follows. Section 2 presents some related works. In section 3, we introduce the timing verification at the design level and we give an overview of a model-driven approach for design refinement (toward implementation); we focus especially on the refactoring phase. Sections 4 and 5 detail respectively Equal Priority merge Pattern (EPMP) and Distinct Priority Merge Pattern (DPMP). In section 6, we show the application of the refactoring phase and the described patterns on a robotic case study. Finally, section 7 concludes the paper and gives some future perspectives.

II. Related Work

In the context of the software development of real-time embedded systems, several works have been proposed with the aim to ensure the respect of timing properties. In [7], the author extends the RT-UML profile to support the creation and validation of OSEK-compliant models .In [8], the authors use OSEK-compliant abstract platforms called SmartOSEK [9] and define a set of transformation rules to create OSEK-compliant models from UML models. In addition, this approach enables the simulation of the resulting OSEK-compliant models and provides the designer with the results to optimize this model at design level. In [10], the authors use RT-UML to annotate UML models describing real-time applications with timing properties. Then they identify the mapping rules between the resulting model and RT-Java as a target platform. The objective of this work is to properly propagate the real-time constraints into the RT-java [11] specific model in order to validate them. These approaches focus especially on real-time aspects without addressing the portability issue which is less convenient when several RTOS are targeted.

To address portability requirement, some exiting approaches follow the lines of the MDA approach [2] by performing timing verification at the design level. Indeed, these approaches aim at verifying a real-time application before its deployment on the RTOS i.e. timing verification of RTOS-independent models. In order to enable such verification, these approaches consider in general some software assumptions related to the underlying RTOS. Indeed, in [12], the authors provide an approach to automatically generate the architectural model from the functional blocks. The focus of this work is to automate this generation and ensure optimized architectural models in terms of timing properties while assuming Earliest Deadline First (EDF) scheduling strategy [13]. In [14] authors propose a MARTE-based [15] methodology by introducing analysis from the functional level to guide the generation of a valid design model in terms of timing requirements. For this achievement, this work assumes that the underlying RTOS provides a fixed-priority scheduling policy [16]. The main objective of these approaches is to produce a description of a real-time application which guarantee the respect of its timing properties and still independent from a particular RTOS. However, for that purpose, they consider in general an ideal RTOS without any attention to the deployment problems which may occur due to the implementation constraints and consequently may affect the timing properties already verified at the design level. Hence, our approach aims at extending the methodology presented in [14] by focusing on the refinement toward implementation of the resulting design model. This work is a step toward providing portability and separation of concerns from one side and early verification of timing properties from the other side during the deployment process of a real-time application on a several RTOSs.

III. Real-time Design Model Refinement toward Implementation

© 2014 ACADEMY PUBLISHER

In this section, we firstly give some real-time concepts related to timing verification at the design level. Then, we introduce the pattern-based refactoring phase by giving an overview of the model-based approach proposed to guide the refinement of the real-time design model to a RTOS-specific implementation model.

A. Timing Verification at the Design Level

The real-time design model consists of m periodic tasks that we denote by $M_a = \{T_1, T_2, \dots, T_m\}$ running in a single-processor system. Each task T_i is defined by a set of parameters deducted from the specification model and the architectural choices enabling thus the timing verification. Indeed, a task T_i is characterized by its worst-case execution time c_i, its activation period P_i and its deadline D_i that represents the time limit in which a task must complete its execution.

The software assumptions considered at the design level to enable timing verification strongly depends on the type of analysis to perform. In this paper, we are interested in the Rate-Monotonic (RM) response time analysis [17]. Thus, we assume that a fixed-priority strategy is used to schedule the different tasks. Consequently, a task T_i is also characterized by its priority p_i. Besides, we suppose that 0 is the lowest priority level and that tasks may share the same priority level.

The architectural model may consist also of a set of software resources $R = \{R_1, R_2, \dots, R_l\}$ that can be shared between one or several tasks (e.g. a mutex to access a critical section). We denote c_{R_i,T_j} the worst-case duration for the task T_j to acquire and release the lock of the resource R_i in case of no contention. Let us remark that c_{R_i,T_j} is considered as an input and that $c_{R_i,T_j} \leq c_j$. Due to the presence of shared resources, a task is also characterized by a blocking time B_i. The blocking time accounts for the time a higher-priority task has to wait, before acquiring the lock, since a lower-priority task owns this lock. The computation of this term depends on the synchronization protocol (e.g. Priority Ceiling Protocol PCP [18], Priority Inheritance Protocol (PIP) [19]) used to implement the access to the shared resource. The choice of which synchronization protocol will be used in the design model corresponds to one of the software assumptions, at this level, and has a direct impact on the timing analysis results. Indeed, if the Priority Ceiling protocol (PCP) [18] is used as a synchronization protocol to avoid unbounded priority inversion and mutual deadlock due to wrong nesting of critical sections, the expression just below will be used to compute the blocking time :

$$B_i = \max_{T_j \in HP_i, R_k \in R} \left\{ C_{R_k,T_j} : p_j < p_i \text{ and } \pi_k \geq p_i \right\} \quad (1)$$

In this protocol each resource R_i is assigned a priority ceiling π_i, which is a priority equal to the highest priority of any task which may lock the resource.

The analysis results correspond to the computation of the processor utilization U and the response time Rep_i of the different tasks in the model. The model satisfies its timing constraints if and only if $U \leq 1$ and $\forall i \in \{1..m\}$, $Rep_i \leq D_i$. The expressions used to compute U and Rep_i are given just below, where HP_i represents the set of tasks with priority higher than T_i .

$$U = \sum_{T_i \in M_a} \frac{c_i}{P_i} \quad (2)$$

$$Rep_i = C_i + B_i + \sum_{j \in HP_i} \lceil R_i / T_j \rceil C_j \quad (3)$$

Figure 3 shows an example of execution of two periodic tasks (T_i and T_j) sharing the resource R implemented using the PCP protocol.

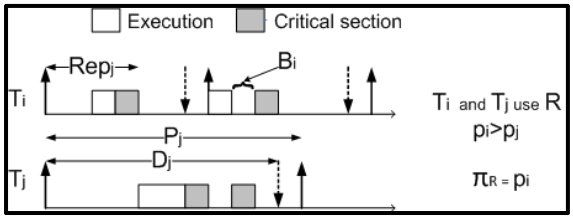


Figure 3. Real-Time Concepts

The priority ceiling of R is equal to the priority of T_i as it is the highest. Up-raising arrows represent the instants of tasks activation, for their part, down-raising arrows determine the deadline for each task activation. The response time to activation is defined as the time between the activation and the completion instants. We also show the blocking time B_i of the task T_i resulting from the utilization of R by T_j. In addition, each task implements a set of functions that we denote by $f \in F$ such as $card(f) \geq 1$ such as F is the set of applicative functions defined in the specification model.

B. Overview of the Proposed Approach

The purpose of the proposed model-based approach is to guide the refinement of the real- time design model to a RTOS-specific model. As depicted on Figure 4, this approach is based on the definition of two types of platforms; the abstract platform used at the design level for the timing validation and the concrete platform which corresponds to the RTOS. Each platform is represented by its model. The abstract platform model explicitly describes the different software assumptions considered to enable the timing verification of the real-time design model. However, the RTOS model describes the software resources provided by the RTOS to enable the execution of the real-time application. One objective of this approach is the generation of implementation models from a RTOS-independent design model satisfying the timing requirements of the application. This generation is performed when no deployment problems are raised. It must also ensures that the resulting model i.e. RTOS-specific implementation (M_{impl}) model still meet the

© 2014 ACADEMY PUBLISHER

timing requirement (i.e.$\forall T_i \in M_{impl}, Rep_i \leq D_i$) while taking into consideration the RTOS characteristics. This objective achievement has been described in details in previous works [20] [21] and is carried out by the vertical path in Figure 4 involving the feasibility evaluation phase, the mapping phase and mapping verification phase.

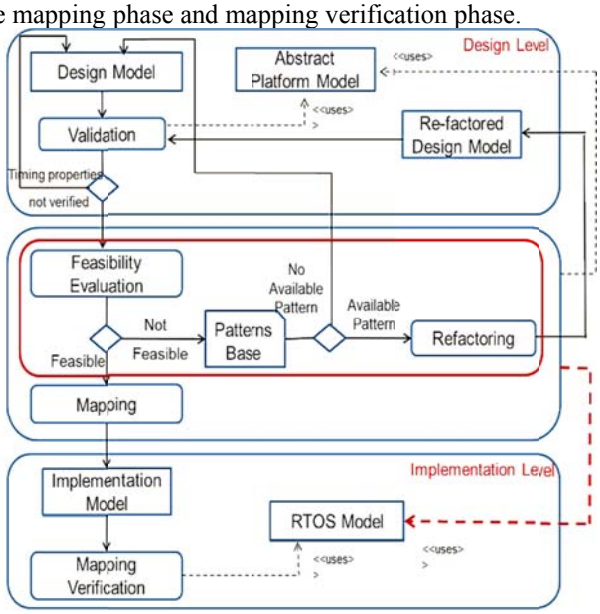


Figure 4. Model-based approach for design refinement

In this paper, we focus on the case where the design model is non-implementable on the target RTOS. The refinement becomes not feasible. In such situation, the refactoring phase proposes solutions to change the original design model with the aim to solve the deployment problem. To this end, this phase is based on a set of predefined patterns. Each pattern corresponds to a deployment problem and is applicable in a particular context (i.e. when some assumptions are fulfilled by the original design model). When the patterns base does not provide any solution for the considered problem (i.e. design model and deployment problem), an error is generated to inform the designer. Otherwise, the refactoring phase applies the appropriate pattern and generates the refactored design model which is at the same level of abstraction as the original design model (i.e. the refactored design model is also an RTOS-independent model). A timing verification of the resulting model is also required to verify whether the timing constraints are still met after the refactoring. When a schedulability issue appears, an error is also generated to inform the designer that the application of the pattern affects the timing properties.

In the following sections, we detail two examples of patterns from the patterns base: Equal Priority Merge Pattern (EPMP) and Distinct Priority Merge Pattern (DPMP).

IV. Equal Priority Merge Pattern: EPMP

In this section, we explain the deployment problem related to EPMP and we describe the proposed solution.

A. Problem Statement

The deployment problem associated to this pattern is described in algorithm 1. Indeed, an error is generated when the original design model M_a defines at least two abstract tasks having the same priority level from one side. From the other side, the target RTOS does not support that the running tasks share the same priority level. One example of this RTOS family is MicroC-OS\II [22].

Algorithm 1: *Equal Priority Levels Detection*

Inputs:
M_a: Original design model
M_{PC} : RTOS model
Output:
Verdict $S = \{E, NP\}$; *E:* Error , *NP*: No Problem
Notations:
$isPriorityShared_{M_a}$: A boolean variable to verify whether the design model defines tasks that share the same priority level
$isPriorityShared_{M_{PC}}$: A boolean variable to mention whether the RTOS supports that tasks share the same priority level.

```
Begin
  isPriorityShared_Ma ← getExistShared( Ma )
  isPriorityShared_MPC ← geSupportShared( MPC )
  if (isPriorityShared_Ma = true) Then
      if (isPriorityShared_MPC = false) Then
          S = E
      else
          S = NP
  Return S
End
```

One key point of this work is platform modeling which has been detailed in previous work [20]. In fact, the RTOS model M_{pc} describes the information required to evaluate the feasibility of the original design model on a given RTOS. The different RTOS models are described using the Unified Modeling Language (UML) enriched with the Software Resource Modeling (SRM) sub-profile of the MARTE standard [15]. Figure 5 shows an excerpt of the MicroC-OS\II [22] focusing on the concept of task. For instance, a MicroC-OS\II task is represented by a class called *MicroC_task* annotated with <<swSchedulableResource>> from SRM to mention that it is a runnable entity. The property isPriorityShared is defined to describe whether the RTOS supports that its tasks share the same priority level. We can see from Figure 5 that this property is valuated to false for MicroC-OS\II.

© 2014 ACADEMY PUBLISHER

Figure 5. Excerpt of the MicroC-OS\II model

B. Solution Description

Let us consider an original design model $M_a = \{T_1, T_2, \ldots, T_m\}$ defining m tasks and n distinct priority levels. In the case where $n < m$, this means that there exist at least two tasks T_i, $T_j \in M_a$ such as $p_i = p_j$. In such situation, this pattern merges the tasks having: (1) the same priority levels, (2) harmonic rates (i.e. two tasks T_i and T_j are harmonic if and only if $P_j \bmod P_i = 0$ and $P_j \geq P_i$). Indeed, this second condition permits to preserve the high level specification (i.e. in terms of functions activation rates). Consequently, the task T'_i resulting from the merge of the tasks $T_i = (p_i, C_i, P_i, D_i, B_i, f_i)$ and $T_j = (p_j, C_j, P_j, D_j, B_j, f_j) \in M_a$, such as $p_i = p_j$ and $(P_j / P_i =$, q is an integer and $P_j \geq P_i)$ is defined as follows:

$$T'_i = (p'_i, C'_i, P'_i, D'_i, B'_i, f'_i)$$

$$p'_i = p_i = p_j$$
$$C'_i = C_i + C_j$$
$$P'_i = min(P_i, P_j)$$
$$D'_i = min(D_i, D_j)$$
$$B'_i = max(B_i, B_j)$$
$$f'_i = \{f_i, f_j\}$$

```
Count = 0;
while(true)
{
if (count mod q = 0) Then
{
f_i
f_j }
else {
f_i}
wait (P'_i)}
```

The consequent model after merging these two tasks consists of m -1 tasks and n distinct priority levels.

C. EPMP Formulation

The formulation of this pattern is described in algorithm 2. This algorithm has as input the design model and generates as outputs an Error or the new design model after refactoring M_{res}. This algorithm generates an error when the rates of the tasks that share the same level are not harmonic. In that case, these tasks cannot be merged because it is not possible to guarantee that the initial functions implemented by the involved tasks will be executed with the same activation rates defined by the application. This algorithm generates also an error when the proposed solution can be applied but the validation generates a negative output. Thus, the resulting design model after refactoring consists of tasks having all distinct priority levels and respecting their timing requirements.

Algorithm 2: *Equal Priority Merge Pattern*

Inputs:
M_a: The design Model
Output:
M_{Res}: Re-factored design model
Error: indicates an error when the pattern is not applicable
Notations:
Type (i): The concept of the platform that types the instance *i*
Ref_Priority: The reference priority level
Ref_Task: A task having a priority equal to *Ref_Priority*
$Current\_priority$: The priority that references the current level
E_Tasks : The list of tasks having priority levels equal to $Current\_priority$

```
Begin
    Ref_Priority ← getHighestPriority(M_a)
    Current_priority ← Ref_Priority
    Ref_Task ← getReferenceTask (M_a, Current_priority )

    While (Ref_Task != null) do
      For each i_s ∈ M_a / Type (i_s) is a periodic task do
        if (isPriorityEqual (i_s, Ref_Task) ) Then
          if (isPeriodHarmonic(i_s, Ref_Task) ) Then
            add (i_s, E_Tasks)

          else Error =true; Return Error

      if (sizeOf (E_Tasks) >1) Then M'_a ← Merge
        (E_Tasks)
        Current_priority ← getHighestPriority
        (M_a, Ref_Priority )
        Ref_Priority ← Current_priority
      Ref_Task ← getReferenceTask
        (M_a, Current_priority )

if ( validate ( M'_a ) = OK ) Then M_Res ← M'_a
    else
      Error =true; Return Error
Return M_Res;
End
```

V. Distinct Priority Merge Pattern : DPMP

In this section, we explain the deployment problem related to DPMP and we describe the proposed solution.

We denote by ***n*** the number of distinct priority levels used in the design model. Besides, ***N*** represents the number of distinct priority levels allowed for the considered application.

A. Problem Statement

The deployment problem associated to this pattern is described in algorithm3.

Algorithm 3: *Distinct priority levels number Detection*

Inputs:
M_a: Original design Model
M_{PC} : RTOS model
Ext_degree: the number of distinct priority levels reserved by the designer for the considered application

© 2014 ACADEMY PUBLISHER

Output:
Verdict $S = \{E, NP\}$; *E:* Error , *NP*: No Problem
Notations:
n Number of distinct priority levels used at the design model
$Nbr\_Levels_{M_{PC}}$: Number of distinct priority levels authorized by the RTOS
N: the number of distinct priority levels allowed for the application

```
Begin
    Nbr_Levels_Ma ← getNumberPriorityLevel (Ma)
    Nbr_Levels_MPC ←getNumberPriorityLevel(MPC)
    N =  Nbr_Levels_MPC
    if ( Ext_degree! = null) Then
        N= Ext_degree
      if  ( (Nbr_Levels_Ma  < N Then
        S = NP
      else
        S = E
    Return S
End
```

This algorithm generates an error when the number of distinct priority levels used in the design model M_a is higher than the number allowed for the considered application (i.e. $n \geq N$). Indeed, this situation may occur in two cases:

(1) large scale applications i.e. n is too large with regard to the number allowed by the RTOS
(2) extensibility requirement i.e. the designer limits the number of distinct priority levels allowed for the considered application (Ext_degree in algorithm 3) to enable the possibility to integrate additional applications (or functions) on the same platform.

The number of distinct priority levels allowed by the RTOS ($Nbr\_Levels_{M_{PC}}$ in algorithm 3) is described in its model. In fact, this number is derived from the maximum and the minimum priority levels allowed by the RTOS. For instance, in Figure 5, these two parameters are represented respectively by *maxPriorityLevel* and *minPriorityLevel* properties of the class *MicroC_Task.*

B. Solution Description

Let us consider an original design model $M_a = \{T_1, T_2, \dots, T_m\}$ defining m tasks and n distinct priority levels ($n \leq m$). In order to reduce n to be equal to N, this pattern merges the tasks having:

- distinct priority levels
- harmonic rates (i.e. two tasks T_i and T_j are harmonic if and only if $P_j \bmod P_i = 0$ and $P_j \geq P_i$ in order to preserve the high level specification (i.e. functions rates)

Let us consider also two tasks $T_i, T_j \in M_a$, each task is defined by a set of parameters; $T_i = (p_i, C_i, P_i, D_i, B_i)$ and $T_j = (p_j, C_j, P_j, D_j, B_j)$ such as $p_i \neq p_j$ and $P_j / P_i = q$; q is an integer. The resulting task from merging these two tasks is denoted $T'_i == (p'_i, C'_i, P'_i, D'_i, B'_i)$ such as $p'_i = \max(p_i, p_j)$, $C'_i = C_i + C_j$, $P'_i = P_i = \min(P_i, P_j)$, $D'_i = \min(D_i, D_j)$ and $B'_i = \max(B_i, B_j)$. Consequently, the model resulting from this merge consists of m-1 tasks and n-1 distinct priority levels. Let us note that we can merge more than two tasks at once.

C. DPMP Formulation

In previous work [23], we have shown that using a heuristic method (i.e. algorithmic description) to describe this pattern is not always effective. Indeed, the problem of merging tasks with the objective to reduce the number of distinct applicative priority levels is a combinatorial problem (i.e. many possible solutions may exist for a given situation). Consequently, we have proposed to formulate this problem using Mixed Integer Linear Programming (MILP) [24] techniques in order to find the best way (in terms of processor utilization U), if any, to merge tasks.

MILP techniques define an objective function which corresponds to a formulation of the considered problem. This formulation is interpretable by a solver that seeks to find a solution for this problem under a set of defined constraints. We give below the objective function and we explain the different constraints defining this pattern.

(i) Objective function

Expression (4) defines the objective function for our problem. We denote by m the number of tasks in the initial model.

$$maximize: \sum_{i,j \in \{1..m\}} Merge_{i,j} - Utilization \qquad (4)$$

Merge is a boolean variable used to mention whether two tasks are merged (if $Merge_{i,j}$ is equal to 1, this means that the tasks T_j is absorbed by the task T_i). Consequently, this objective function aims at maximizing the number of merge while minimizing the processor utilization.

(ii) Merging situations constraints

The objective function aims at maximizing the number of merge, however this function should be aware of some constraints that limit the exploration space and eliminate non meaningful merging situations. These constraints are presented just below:

$$n - \sum_{i,j \in \{1..m\}} Merge_{i,j} = N \qquad (5)$$

$$\forall\, i,j \in \{1..m\}, Merge_{i,j} = 0 \; if \; \left(isHarmonic_{i,j} = 0\right) or \; (p_i = p_j) \qquad (6)$$

$$\forall\, j \in \{1..m\}, \sum_{i \in \{1..m\}} Merge_{i,j} \leq 1; \; \forall\, i,j,k \in \{1..m\} \; Merge_{i,j} + Merge_{j,k} \leq 1 \qquad (7)$$

In constraint (5), *n* and *N* represent two input parameters which are defined previously. This constraint

© 2014 ACADEMY PUBLISHER

means that we have to maximize the number of merged tasks and thus minimize the number of distinct priority levels used in the design model until the number authorized by the RTOS. Indeed, this Equation serves as a bound for the objective function (i.e. the number of merge). Constraint (6) defines a new input parameter which is *isHarmonic*, this parameter is used to mention if two tasks are harmonic. Thus if the value of $isHarmonici_{i,j}$ is equal to 1, then the corresponding tasks T_i and T_j have harmonic rates. Consequently, this constraint avoids the merge of non-harmonic tasks and avoids also the merge of tasks having equal priority levels ($p_i = p_j$).

Finally, the constraint in (7) is used to avoid non-meaningful situations which correspond to the merge of a task already merged.

We define also a new boolean variable that we denote by TASKS and which refers to the resulting task model after merging the different tasks. Therefore, constraint (8) is defined to create the new obtained model. In fact, when $Merge_{i,j}$ is equal to 1, $TASKS_j$ will be equal to 0 and $TASKS_i$ will be equal to 1. This constraint is defined as follows:

$$\forall\ j\ \in \{1..m\},\ TASKS_j = 1 - \sum_{i\in\{1..m\}} Merge_{i,j} \quad (8)$$

(iii) Real-time constraints

The constraints defined in this section are related to real-time requirements. Indeed, the model obtained after applying the merge pattern should satisfy the timing constraints which are expressed in constraints (9) and (10).

$$\forall\ i\ \in \{1..m\}, Rep_i \le D_i \quad (9)$$

$$utilization\ \le Max\_Utilization \quad (10)$$

Constraint (9) ensures that the response times Rep_i of the different tasks in the resulting model are lower or equal than their deadlines. Constraint (10) verifies whether the processor utilization is lower or equal than the maximum authorized utilization. Constraint (11) gives the computation formula of T_i response time while taking into consideration the different decisions of merge.

$$\forall\ i\ \in \{1..m\}, Rep_i = \delta_i + \theta_i + \beta_i \quad (11)$$

The first term of the expression (11) is δ_i which corresponds to the worst case execution time of the task T_i. This term is computed as follows:

$$\forall\ i\ \in \{1..m\}, \delta_i = TASKS_i * C_i + \sum_{j\in\{1..m\}} Merge_{i,j} * C_j \quad (12)$$

The execution time of a deleted task will be equal to 0 since the term $TASKS_i$ is equal to 0 and $\forall\ j \in \{1..m\}, Merge_{i,j} = 0$. However, the execution time of a task resulting from the merge of different tasks will be equal to the sum of the execution times of these tasks.

The second term in the expression is θ_i representing the overhead induced by the interferences of the task T_i with the different tasks in the model having higher priorities. We denote by HP_i the set of these tasks. This variable is defined $\forall\ i\ \in \{1..m\}$ as the sum of two terms φ_i, ϑ_i and it is defined just below:

$$\theta_i = \varphi_i + \vartheta_i \quad (13)$$

$$\varphi_i = TASKS_i * \sum_{j\in HP_i, j\in\{1..m\}} TASKS_j * \left(\left\lceil \frac{Rep_i}{P_j} \right\rceil * C_j\right) \quad (14)$$

$$\vartheta_i = TASKS_i * \left[\sum_{j\in HP_i, j\in\{1..m\}} TASKS_j * \left(\sum_{k\in\{1..m\}} Merge_{j,k} * \left\lceil \frac{Rep_i}{P_j} \right\rceil * C_k\right)\right] \quad (15)$$

The interference term is equal to 0 if the corresponding task is a deleted one ($TASKS_i = 0$). Otherwise, this term computes the overhead resulting from the interferences of tasks $T_j / \ j \in HP_i$. This expression takes into consideration the different situations when higher priority tasks correspond to deleted ones ($TASKS_j$ in the expression) or tasks resulting from merging decision ($Merge_{j,k}$ in the expression). We notice that the expressions (14) and (15) are not linear and thus in order to be interpretable by the solver these expression must be linearized.

For instance, the linearization of the expression (14) is given by the following constraints:

$$\forall\ i,j\ \in \{1..m\}, 0\ \le\ X_{i,j} - \left(\frac{Rep_i}{P_j}\right) < 1 \quad (16)$$

$$\forall\ i,j\ \in \{1..m\}, Y_{i,j} \le X_{i,j};\ Y_{i,j} \le M * TASKS_j;\ X_{i,j} - M * \left(1 - TASKS_j\right) \le Y_{i,j} \quad (17)$$

In order to linearize the expression (14), we define new constraints (16) (17) and 2 additional variables X and Y. The constraint (16) permits to compute the term $\left\lceil \frac{Rep_i}{P_j} \right\rceil$, however the constraints in (17) are defined to determine the value of $TASKS_j * \left(\left\lceil \frac{Rep_i}{P_j} \right\rceil\right)$. Eventually, the constraints in (18) and (19) are used to compute the final value of $\varphi_i\ \forall\ i\ \in \{1..m\}$.

$$\forall\ i\ \in \{1..m\},\ \varphi_i \le \sum_{j\ \in\ HP_i, j\in\{1..m\}} Y_{i,j} * C_j\ ;\ \varphi_i \le M * TASKS_i \quad (18)$$

$$\forall\ i \in \{1..m\}, \left[\sum_{j\ \in\ HP_i, j\in\{1..m\}} Y_{i,j} * C_j\right] - M * (1 - TASKS_i) \le\ \varphi_i \quad (19)$$

Finally the third term in the expression of the response time is β_i which represents the blocking time experienced by a task when lower priority tasks delays the access to a shared resource. This variable is computed as follows:

$$\forall\ i\ \in \{1..m\}, \beta_i = TASKS_i * BT_i \quad (20)$$

This term is equal to 0 if the task corresponds to a deleted task. Otherwise, the blocking time of the considered task is equal to BT_i which is defined as follows:

© 2014 ACADEMY PUBLISHER

$$BT_i = \begin{cases} B_i \;\; if \; \sum_{i,j \in \{1..m\}} Merge_{i,j} = 0 \\ \max_{j \in \{1..m\}} Merge_{i,j} * B_i \;\; Otherwise \end{cases} \quad (21)$$

The term B_i in expression (21) is an input parameter representing the blocking time of the task T_i. Consequently, if the considered task is not merged with other tasks in model ($\sum_{i,j \in \{1..m\}} Merge_{i,j} = 0$), the blocking time is kept the same. Otherwise, the blocking term corresponds to the maximum of the merged task blocking times.

The processor utilization represents an important term in scheduling analysis. In fact, in order to confirm that the design model meets the timing constraints the following constraint must be verified:

$$Utilization \leq 1 \quad (22)$$

We define the Utilization term by the constraints just below:

$$Utilization = \mu_1 + \mu_2 \quad (23)$$

$$\mu_1 = \sum_{i \in \{1..m\}} TASKS_i * \left(\frac{C_i}{P_i}\right) \quad (24)$$

$$\mu_2 = \sum_{i \in \{1..m\}} TASKS_i * \sum_{j \in \{1..m\}} Merge_{i,j} * \left(\frac{C_i}{P_i}\right) \quad (25)$$

Under these constraints, the objective function will seek for the best way to merge tasks (i.e. the optimized solution in terms of utilization) in order to reduce the number of used priority levels while ensuring the respect of timing properties.

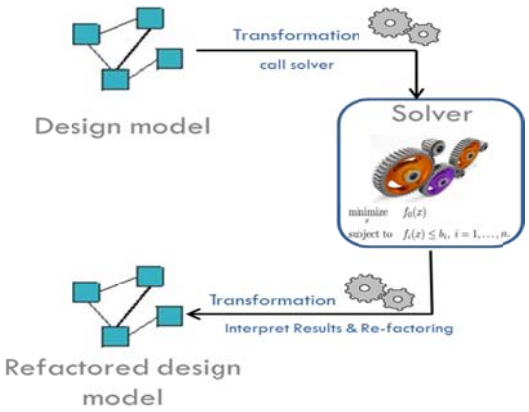


Figure 6. Refactoring by applying DPMP

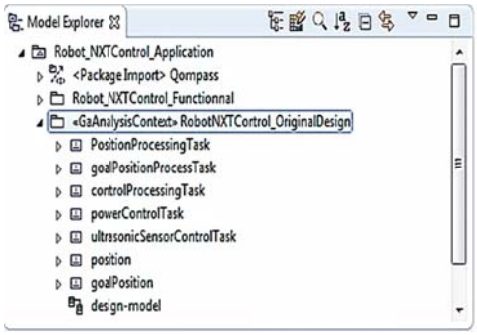
As shown in the Figure 6, from the original design model, a transformation is defined to call the solver. The solver executes the linear program and generates the appropriate output. The solution generated by the linear program will be interpreted in order to perform the refactoring. This refactoring permits to solve the problem of insufficient priority levels for the considered application while ensuring the respect of timing properties.

VI. Experiments

The objective of this section is to evaluate the proposed refactoring phase. Indeed, this phase and the previously detailed patterns (EPMP and DPMP) were integrated in the Qompass-Architect tool. Starting from a functional description of a real-time application, Qompass-Architect automatically generates the corresponding design model satisfying the timing requirements. In order to evaluate the applicability of the proposed patterns, we consider as entry point an original design model describing a classical robotic real-time application. This design model was generated by the Qompass-architect tool (see Figure 7). For sake of clarity, we give a tabular description of the different models. As depicted on Figure 7, this model consists of five periodic tasks; *positionProcessingTask*, *powerControlTask* *ultrasonicSensorContro Task*, *goalPositionProcessTa*sk, *controlProcessingTask* and two shared data resources; *position* and *goalPosition.*

The tasks *positionProcessingTask* and *ultrasonicSensorContro Task* have the higher priority value which is equal to 20. However, *powerControlTask* has the lowest priority equal to 0. This original design model satisfies the timing requirements since the response of the different tasks are lower than their deadlines.

Starting from this design model, the objective is to generate the appropriate RTOS-specific implementation model. One key point of this work is to target several RTOSs. Consequently, as shown in Figure 8, the first step is to select a target RTOS from the model library. If the designer selects MicroC-OSS\II, the feasibility evaluation phase verifies whether the original design model is implementable on the selected one.



	Period	Deadline	Time Budget	priority	Blocking Time	Response Time
positionProcessingTask	20	20	7	20	2	17
ultrasonicSensorControTask	40	40	8	20	0	15
goalPositionProcessTask	100	100	4	15	2	28
controlProcessingTask	100	100	12	10	0	38
powerControlTask	300	300	7	0	0	60

Figure 7. Original design model generated by Qompass-Architect tool

© 2014 ACADEMY PUBLISHER

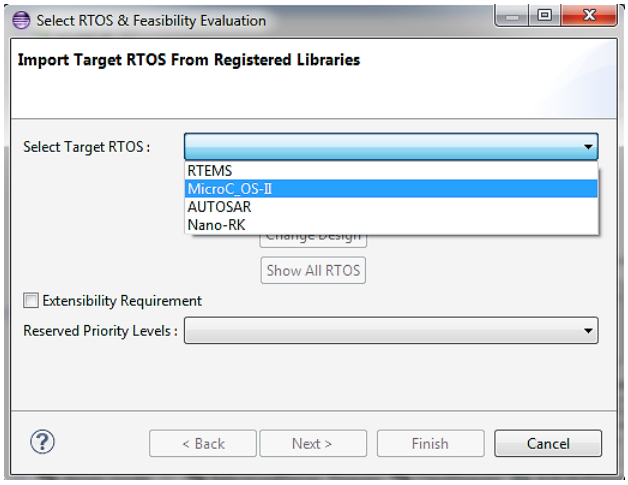


Figure 8. Select target RTOS from the model library

In that case, as shown in Figure 9, the feasibility evaluation phase generates an error status for the equal priority levels test (described in algorithm 1). Indeed, the tasks *positionProcessingTask* and *ultrasonicSensorControTask* in the original design model, given in Figure 7, have the same priority level equals to 20. However, MicroC-OS\II does not allow that tasks share the same priority level. Consequently, the designer looks for a solution to this deployment problem using the button *Find a Pattern* (see Figure 9).

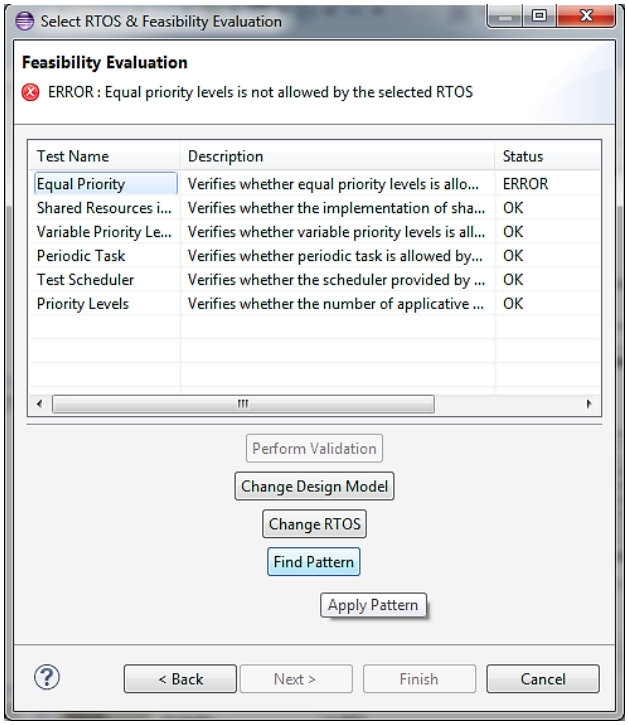


Figure 9. Feasibility evaluation for MicroC-OS\II

In that case, the *patterns base* proposes EPMP pattern as a potential solution (Figure 10). The application of this pattern corresponds to the execution of algorithm 2 (defined in section 4.3).

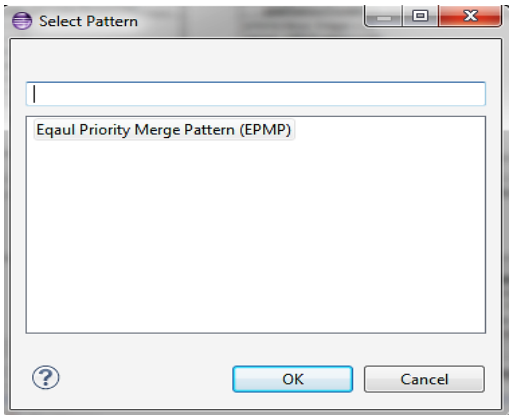


Figure 10. Select a pattern from the patterns Base

Then, the framework generates a *warning* to mention that a re-validation of the resulting model is required (Figure 11).

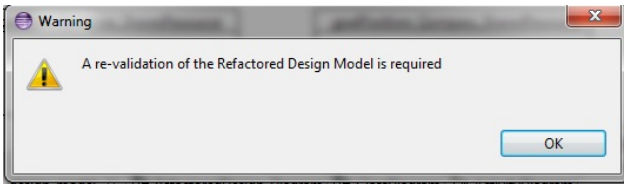
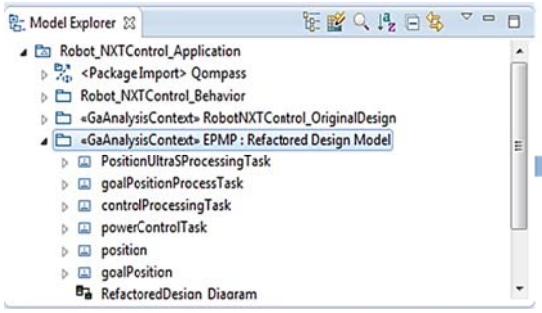


Figure 11. Re-validation request after applying EPMP

The resulting model after applying this pattern is given in Figure 12. This model consists of only four tasks having all distinct priority levels and two shared resources. Indeed, the two tasks *positionProcessingTask*, *ultrasonicSensorControTask* which have the same priority level are merged to a single task called *positionUltraSProcessingTask*. This pattern is applicable since the merged tasks have harmonic rates (i.e. $^{40}/_{20} = 2; q = 2$) and the resulting model satisfies the timing requirement (i.e. the response time of the different tasks in the model are lower than their deadlines).

© 2014 ACADEMY PUBLISHER



	Period	Deadline	Time Budget	priority	Blocking Time	Response Time
positionUltraSProcessingTask	20	20	15	20	2	17
goalPositionProcessTask	100	100	4	15	2	36
controlProcessingTask	100	100	12	10	0	76
powerControlTask	300	300	7	0	0	98

Figure 12. Refactored design model by applying EPMP

To show the applicability of the second pattern, we choose now as a target RTOS RTEMS [25]. In addition, we assume that the designer for extensibility requirements limits the number of distinct priority levels for this application to only 3 (as shown in Figure 13). As depicted on Figure 14, the feasibility evaluation phase, in that case, generates an error status for the distinct priority levels number test (given in algorithm 3). Indeed, the original design model defines four distinct priority levels; however, the designer wants to deploy this application using only three distinct levels.

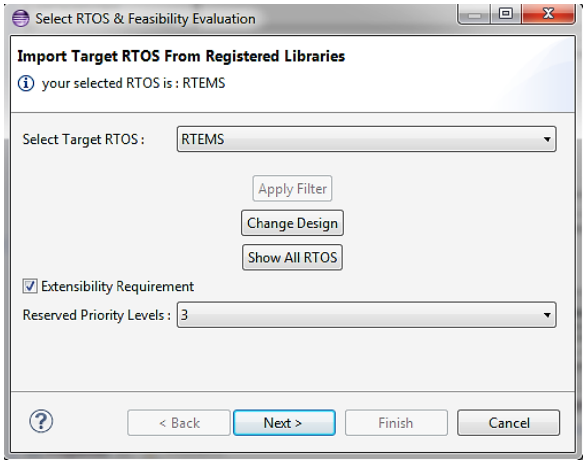


Figure 13. Select RTOS and set extensibility

In that case, the framework proposes the DPMP as a way to solve this problem. The application of this pattern was explained in Figure 6 and corresponds to the execution of the linear program detailed in section 5.3.

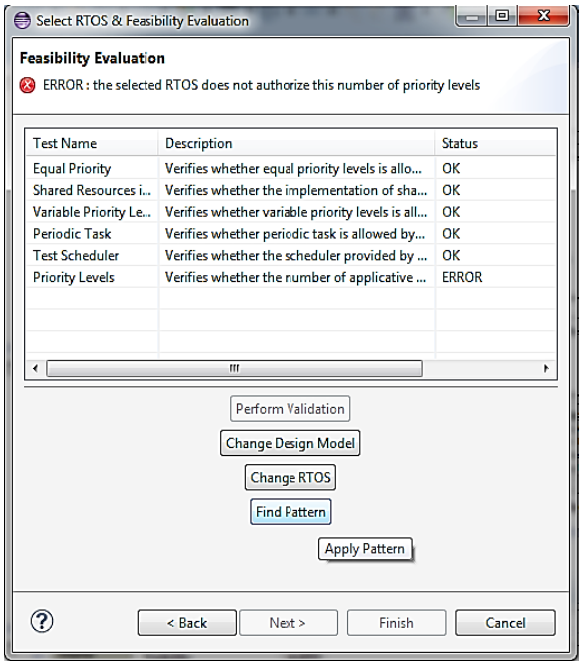


Figure 14. Feasibility evaluation for RTEMS

No.	Column name		Activity	Lower bound	Upper bound
1	TASKS[1]	*	1	0	1
2	TASKS[2]	*	1	0	1
3	TASKS[3]	*	1	0	1
4	TASKS[4]	*	0	0	1
5	TASKS[5]	*	1	0	1
6	Merge[1,1]	*	0	0	1
7	Merge[1,2]	*	0	0	1
8	Merge[1,3]	*	0	0	1
9	Merge[1,4]	*	0	0	1
10	Merge[1,5]	*	0	0	1
11	Merge[2,1]	*	0	0	1
12	Merge[2,2]	*	0	0	1
13	Merge[2,3]	*	0	0	1
14	Merge[2,4]	*	0	0	1
15	Merge[2,5]	*	0	0	1
16	Merge[3,1]	*	0	0	1
17	Merge[3,2]	*	0	0	1
18	Merge[3,3]	*	0	0	1
19	Merge[3,4]	*	1	0	1
20	Merge[3,5]	*	0	0	1
21	Merge[4,1]	*	0	0	1
22	Merge[4,2]	*	0	0	1
23	Merge[4,3]	*	0	0	1
24	Merge[4,4]	*	0	0	1
25	Merge[4,5]	*	0	0	1
26	Merge[5,1]	*	0	0	1
27	Merge[5,2]	*	0	0	1
28	Merge[5,3]	*	0	0	1
29	Merge[5,4]	*	0	0	1
30	Merge[5,5]	*	0	0	1
31	Rep[1]		17	0	
32	Rep[2]		15	0	
33	Rep[3]		40	0	
34	Rep[4]		0	0	
35	Rep[5]		60	0	

Figure 15. Output of the linear program

© 2014 ACADEMY PUBLISHER

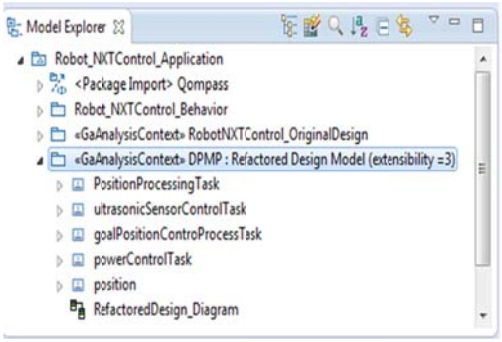
Figure 15 shows an excerpt from the output file of the linear program describing the DPMP pattern. By executing the linear program for the considered problem, the solver confirms that a solution exists. This solution corresponds to the merge of the tasks 3 and 4 (*Merge [3, 4] = 1* in figure 15) which correspond respectively to the tasks *goalPositionProcessTask* and *controlProcessingTask*. Indeed, the *controlProcessingTask* is absorbed by the task *goalPositionProcessTask*. Consequently, the *controlProcessingTask* corresponds to a deleted task (*TASKS [4] =0* in Figure 15) and thus its response time is equal to 0 (*Rep [4] =0* in Figure 15).

This solution was interpreted by the framework to generate the refactored design model given in Figure 16. We can see from this figure that the resulting model consists of four tasks and three distinct priority levels. This model still satisfies the timing requirements since the response times of the different tasks are lower than their deadlines.

VII. Conculsion and Future Work

In this paper, we have shown that timing verification of a real-time design model requires an abstraction of the underlying Real Time operating System (RTOS). In a model-driven context, this abstraction leads to a consideration of software assumptions to keep the RTOS-independence of the design model. This may cause deployment problems when these assumptions are not verified for the target RTOS. In such situation, this work proposes a pattern-based approach when a deployment problem appears with the aim to solve the latter. This approach performs the refactoring of the original design model while maintaining its portability and the respect of timing properties. The automation of the proposed approach permits to evaluate its applicability on a robotic case study.

As perspective of this work, we aim at considering other problems such as timer granularity, shared resources implementation, etc. and proposing for each particular problem a software pattern to enrich our pattern base. In addition, we can extend this work by considering the behavioral aspect and thus other problems must be considered and consequently additional software pattern must be defined. Another possible perspective of this work is to propose software patterns to perform the refactoring of the original design model with the aim to optimize non-functional criteria (other than timing properties) such as memory energy, etc.



	Period	Deadline	Time Budget	priority	Blocking Time	Response Time
positionProcessingTask	20	20	7	20	2	17
ultrasonicSensorControlTask	40	40	8	20	0	15
goalPositionControProcessTask	100	100	16	15	2	40
powerControlTask	300	300	7	0	0	60

Figure 16. Refactored design model by applying DPMP

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] http://www.omg.org/.

[3] B. Kienhuis, E. Deprettere, P. Van Der Wolf, and K. Vissers. A methodology to design programmable embedded systems. In Embedded processor design challenges, pages 321–324. Springer, 2002.

[4] R. Yemhalli. Real-time operating systems: An ongoing review. In Work-ln-Progress Sessions of the 21th IEEE Real-lime System Symposium (RTSSWIPOO), Orlando, Florida, November 2000.

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

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

[7] A. Moore. Extending the RT-profile le to support the OSEK infrastructure. In Proceedings of the 5th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, pages 341347, Washington, DC, Apr. 2002.

[8] G. Yang, M. Zhao, L. Wang, and Z. Wu, "Model-based Design and Verification of Automotive Electronics Compliant with OSEK/VDX," presented at The Second International Conference on Embedded Software and System (ICESS), Xi'an, 2005.

[9] M. Zhao, Z. Wu, G. Yang, L. Wang, and W. Chen,"SmartOSEK: A Dependable Platform for Automobile Electronics," The First International Conference on Embedded Software and System, vol. Springer-Verlag GmbH ISSN: 0302-9743, pp. 437, 2004.

[10] L.B. Becker,; R. Holtz,.; C.E. Pereira, On Mapping RT-UML Specifications to RT-Java API: Bridging the Gap. In the 5th IEEE International Symposium on Object- Oriented Real-Time Distributed Computing, Washington, USA, 2002. p. 348-355

[11] The Real-Time for Java Experts Group, The Real Time Specification for Java, Version 0.8.1, 27 Sept. 1999; http://www.rtj.org/rtj.pdf

[12] C. Bartolini, G. Lipari, and M. D. Natale, From functional blocks to the synthesis of the architectural model in

© 2014 ACADEMY PUBLISHER

embedded real-time applications, in Proc. IEEE Real Time and Embedded Technology and Applications Symposium (RTAS), 2005, pp. 458467.
[13] F. Zhang and A. Burns, "Schedulability analysis for real-time systems with edf scheduling," IEEE Transactions on Computers, vol. 58, no. 9, pp. 1250–1258, 2009
[14] C. Mraidha, S. Tucci-Piergiovanni, S. Gerard. Optimum: A MARTE-based methodology for Schedulability Analysis at Early Design Stages. In proceeding of the Third IEEE international workshop UML and Formal Methods (UML&FM 2010). Shanghai, China
[15] 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
[16] J. E. Lehoczky,1990. Fixed priority scheduling of periodic task sets with arbitrary deadlines. Proceedings in ll the IEEE Real-Time Systems Symposium, 5-7 December 1990, pp. 201-209.
[17] M. H. Klein, T. Ralya, B. Pollak, R. Obenza, and M. G. Harbour. A practitioners handbook for real-time analysis. Kluwer Academic Publishers, 1993.
[18] J. B. Goodenough and L. Sha. The priority ceiling protocol: A method for minimizing the blocking of high priority Ada tasks, volume 8.ACM, 1988
[19] L. L. Sha, R. Rajkumar, and J. Lehoczky, "Priority Inheritance Protocols: An Approach to Real-Time Synchronization," IEEE Trans. Computers, Sept. 1990, pp. 1175-1185.
[20] 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 (SEAA12), Cesme, Izmir, Turkey, September 2012.
[21] R. Mzid, Ch. Mraidha, J-P. Babau, M. Abid. Real Time Design Models to RTOS- Specific Models Refinement Verification. The 5th International Workshop on Model Based Architecting and construction of Embedded Systems ACES-MB, 2012, Innsbruck, Austria, September 2012.
[22] Jean J. Labrosse. MicroC/OS-II The Real-Time Kernel
[23] R. Mzid, Ch. Mraidha, A. Mehiaoui, S. Tucci-Piergiovanni, J.P Babau, and M. Abid. DPMP: A Software Pattern for Real-Time Tasks Merge. The 9th European Conference on Modeling Foundations and Applications (ECMFA'13). Montpellier, France, July 2013.
[24] M. Guignard-Spielberg, K. Spielberg, Integer programming: State of the art and recent advances, Annals of Operations Research 139, 2005.
[25] RTEMS C Users Guide. Edition 4.6.5, for RTEMS 4.6.5. August 2003.

Rania Mzid received a master degree in new technologies of computer dedicated systems in 2010 from the National School of Engineers of Sfax (Tunisia). Currently, she is a PhD student at the Laboratory of Model Driven Engineering for Embedded Systems of the CEA LIST institute in France (in cooperation with the Lab-STICC laboratory at the university of Bretagne (UBO) in France and CES laboratory at the National School of Engineers of Sfax in Tunisia). Her research interests include the software development of real-time and embedded systems based on OMG standards such as MDA (Model Driven Architecture) and MARTE (a UML profile for designing real-Time embedded systems).

Chokri Mraidha is a researcher at the Laboratory of Model Driven Engineering for Embedded Systems of the CEA LIST institute in France. He got a master degree in distributed computing in 2001 and a PhD in Computer Science in 2005. His research and development interests include real-time and embedded systems model driven development, correct-by-construction and optimization-oriented real-time systems design. He is involved in UML-based OMG standards for design of real-time systems like SysML and MARTE and the AUTOSAR standard for automotive. He is working in European and French research projects developing model-based approaches for design and verification of architectures for critical real-time systems for automotive, railway, aerospace and nuclear power plants.

Jean-Philippe Babau is Professor in Computer Science at the University of Brest (UBO) in France. He received a PhD in computer science from the University of Poitiers in 1996. The thesis defined a method to perform a real-time validation for complex multitasking systems. He was researcher at the CITI laboratory in Lyon until 2008, working on the use of both formal (timed automata, SDL) and modeling (DSL, UML) languages for architecture description to build complex (open and dynamic) Communicating Real-Time Embedded Systems. Now, he works on QoS and architecture modeling, real-time code generation and data integration for complex systems, including marine and ubiquitous systems.

Mohamed Abid received the PhD. degree from the National Institute of Applied Sciences, Toulouse (France) in 1989 and the "thèse d'état" degree from the National School of Engineering of Tunis (Tunisia) in 2000 in the area of Computer Engineering & Microelectronics. His current research interests include: hardware-software codesign, System on Chip, Reconfigurable System, and Embedded System, etc. He has also been investigating the design and implementation issues of FPGA embedded systems. Actually, he occupies the post of director of doctoral school "Sciences & Technologies", University of Sfax. He is member and Head of the research laboratory «Computer Embedded System » CES-ENIS, since 2006 (http://www.ceslab.org). He was member of "System on Chip at Computer, Electronic and Smart engineering system" Laboratory at ENIS 2001-2005. He was also responsible of «Hardware-Software co-design» research Group at EμM-Lab-FSM, Monastir-Tunisia, 1991-2000. He is member of scientific committee at ENIS, since 2008, member of quality committee at ENIS, 2006-2007 and member of national committee of engineering pedagogy, since 2006. He was member and responsible of doctoral degree «computer system engineering» at ENIS, 2003-2010. He served in national or international conference organization and program committees at different organizational levels. He was also Joint Editor of Specific Issues in two International Journals and Joint editor of many conference's articles nationals and internationals. Dr. Abid is joint coordinator or an active member of several International Research and Innovation projects. He is Author or co-author of more than 30 publications in Journals and author or co-author of more than 180 papers in international conferences. He is also author or coauthor of many guest's papers, Joint author of many book's chapters. Dr. Abid has served also as Guest professor at several international universities and as a Consultant to research & development in Telnet Incorporation.

© 2014 ACADEMY PUBLISHER