MDE benefits for Real Time Operating Systems modeling

Yessine Hadj kacem[1], Adel Mahfoudhi[1, 2], Mohamed Abid[1]
[1] National Engineering School of Sfax Road Soukra km 3,5
Computer & Embedded Systems Laboratory (CES)
B.P. : w -- 3038 Sfax TUNISIA
[2] Department of Computer Science, Science Faculty of Sfax
Road Soukra km 3,5 BP : 1171 -- 3000 Sfax TUNISIA
adel.mahfoudhi@fss.rnu.tn
mohamed.abid@enis.rnu.tn

***Abstract*—A large part of real time embedded systems RTES has to satisfy real time constraints and it usually employs Real Time Operating System RTOS. In order to decrease the design complexity of such systems, they need methods and tools based on high abstraction layers.**

Currently, UML profiles are found to be an effective solution for the automatic RTES design. Unfortunately, they are poor in integrating RTOS modeling.

In the present work, a methodology based on model driven engineering MDE for RTOS design is introduced. The proposed approach aims at defining a platform independent model of RTOS. It suggests the implementation of statecharts relating to the process states, whose real time constraints can be checked by defining their semantic variants. The ultimate goal is the automatic generation of the code related to the scheduler.

***Keywords*— RTOS modeling, semantic variants, statecharts implementation, MDE, automatic code generation**

I. INTRODUCTION

Designing ERTS has always been a challenge. Indeed, standards to facilitate the checking of system properties at a preliminary stage are progressing well based on different abstraction layers. With regard to the bottom layers, there exists many synthetic tools; the only problem relates to the CAD (Computer Aiding Design) of the highest level which is the concern of this work.

At present, UML (Unified Modeling Language) is having a growing interest in the software and hardware development. It represents a viable solution to decrease the complexity of ERTS design via UML profile. The ERTS complexity depends on the architecture deployment and requires runtime guarantees. Although ERTS requires a real time operating system, the existing profiles do not integrate the notion of RTOS modeling during the ERTS design, since they just focus on architecture and application modeling.

This paper presents our contribution to the RTOS modeling, based on model driven engineering. In fact, the mains goals are the definition of a platform independent model of RTOS and the automatic generation of the code related to the scheduler. It suggests the implementation of statecharts relating to process states.

For the suggested models, the structure of the RTOS is described through a class diagram which includes the definition of operational semantics. Then, the behaviour of a task which constitutes the core of the RTOS is defined in order to ensure coherence between various diagrams UML. The temporal and transitional semantics of the statecharts relative to the various states of a real time process is defined in order to reach the model of task scheduling. These two independent platform models are integrated in MDE process to generate the code related to the scheduler.

This paper starts by providing a brief discussion about some related work in section two. Then, the proposed design methodology is described in section three, in which the models of the RTOS structure and the scheduler are introduced through the implementation of statecharts. The experimental results of an application example are provided in section four. At the end, some final conclusions with some future work are given.

II. RELATED WORK

After examining the specificity of each UML profile such as SPT (Scheduling, Performance and Timing) [15, 12], QoS/FT (Quality of Services & Faults Tolerance) [15] and MARTES (Modeling and Analysis of Real-Time and Embedded systems) [13], it is concluded that the focus was on the description of the material architecture and the application. These profiles are founded on an abstraction level higher than other approaches like ROOM, SDL, ADL, Petri Net. They also aim at the applications to data flow predominance rather than those to control. Even though these works briefly tackle the temporal aspect, they cannot cover the RTOS modeling. They are criticized for the lack of temporal and transitional semantics common to the models as well as the absence of tools which support them. In reality, these works have not enabled us to guarantee the reliability of the system yet; i.e., its determinism aspect. These models do not support the integration of real time characteristics sufficiently and therefore they do not consider the RTOS related to a specific architecture and application. The simulation approaches need a simulation time long enough to give a relatively reliable

sight of operation.

According to [16], RTOS modeling is based on two independent class diagrams: a diagram describing the structure and another describing the scheduler. They suffer from major limitations, namely the coherence between the used diagrams and the lack of temporal semantics definition. In fact, because it can not cover the temporal behaviour of the RTOS, the diagram used to characterize the scheduler is a static one. It must also be complementary to the structure model via a good expression of the follow-up of the real time process evolution. Therefore, a methodology assuring the coherence between the used diagrams and the support of scheduling model is important.

In [14], the author's work consists in developing a middleware in order to implement scheduling algorithms on a real platform. It just focuses on mapping and ordering tasks dynamically for platforms using a middleware layer between the application and the RTOS.

In [10], a model driven approach aims at proposing generic RTOS APIs (Application Programming Interfaces) and generating a fully-functional code by transforming generic RTOS APIs into RTOS specific APIs. This proposition can describe most of typical RTOS services but does not support real time task scheduling.

III. RTOS MODELING

The proposed methodology presents a step ensuring coherence between the various used UML diagrams and covering the behavioral aspect of the system as real time constraints. First, the model of the RTOS structure is defined. Then, a statechart diagram related to the state of a real time task is specified. After that, the temporal semantics presented by the statecharts [1] is given. While defining the semantic variation points of the statecharts, some techniques such as the reification and enumeration of the states and the events are applied. The model related to the RTOS structure corresponds to the source model during the stage of model transformation. During this stage, the scheduling model represents the target model.

After defining the RTOS models, temporal constraints such as deadline, duration, etc are specified using Object Constraint Language (OCL).

A. RTOS structure model

A class diagram is proposed for the description of the RTOS structure. It describes the major components of the RTOS.

The class diagram which is presented by Figure 1 is characterized by the following entities:

- Task: It is the most important component of the RTOS. A task must acquire a great number of information in order to manage their scheduling
- Event: It causes the change of a task state
- ISR: Interrupt Server Routine: It is the routine in charge of the interruption processing. In this context, it makes the relay between the material interruption mechanism and the software one
- Alarm: Based on a meter, an alarm could activate a task, impose an event or activate an alarmCallBack
- Counter: It presents a software/ hardware source for an alarm. It is an object intended for the recording of "ticks" coming from a timer
- Resource: This entity is used to coordinate the concurrent accesses to shared resources. It is similar to semaphores. It is also used for explaining resource management.
- MeanOfCommunication: It is an abstract interface which manages data between active objects [4]. The class ProtectedVar, which implements this interface, associates a mechanism of data protection (semaphore). In addition, LettreBox uses a file of messages. Besides, the read and write methods of this interface can update or get the value of the protectedVar class. This entity ensures data protection
- Watchdog: The ISR contains one or more watchdog timers. The watchdog could possibly provide debugging information
- Precedes: It illustrates the dependence of a task on another. It includes the definition of operational semantics [2].

Before presenting the scheduling model, it should be born in mind that each state of a running task on RTOS can take only one of the following values: {Waiting, Running, Ready, Suspended, Created}. As for the event, it has these values :{ terminate, activate, start, wait, preempt, release, create}.

B. RTOS scheduling model based on statecharts implantation

The structure of the statecharts diagram is, nonetheless, given a precise specification [17]. It can not easily be understood. So, UML 2.0 Statecharts present some semantic variation points. These variation points [9] concern, principally, three aspects: the time management, event selection policy, and transition selection policy.

A set of approaches [8, 11] was proposed in the literature in order to define these semantics and implement the statecharts. The present work adopts the approach proposed by [6], whose technique is based on the enumeration and reification

The reification consists in the transformation of states into specific class hierarchy through the application of the design patterns.

A solution to separate the behaviour related to a state in an object, is to reify states through the use of the state pattern [7]. To reify and select the right transition events, the command pattern [7] is applied to the entity Task (See Figure 2). In order to ensure the progression of the automat, it is necessary to focus on the deterministic aspect of the system. It is also essential to determine the state running of the automat and the behavior to be adopted according to the event which has occurred.

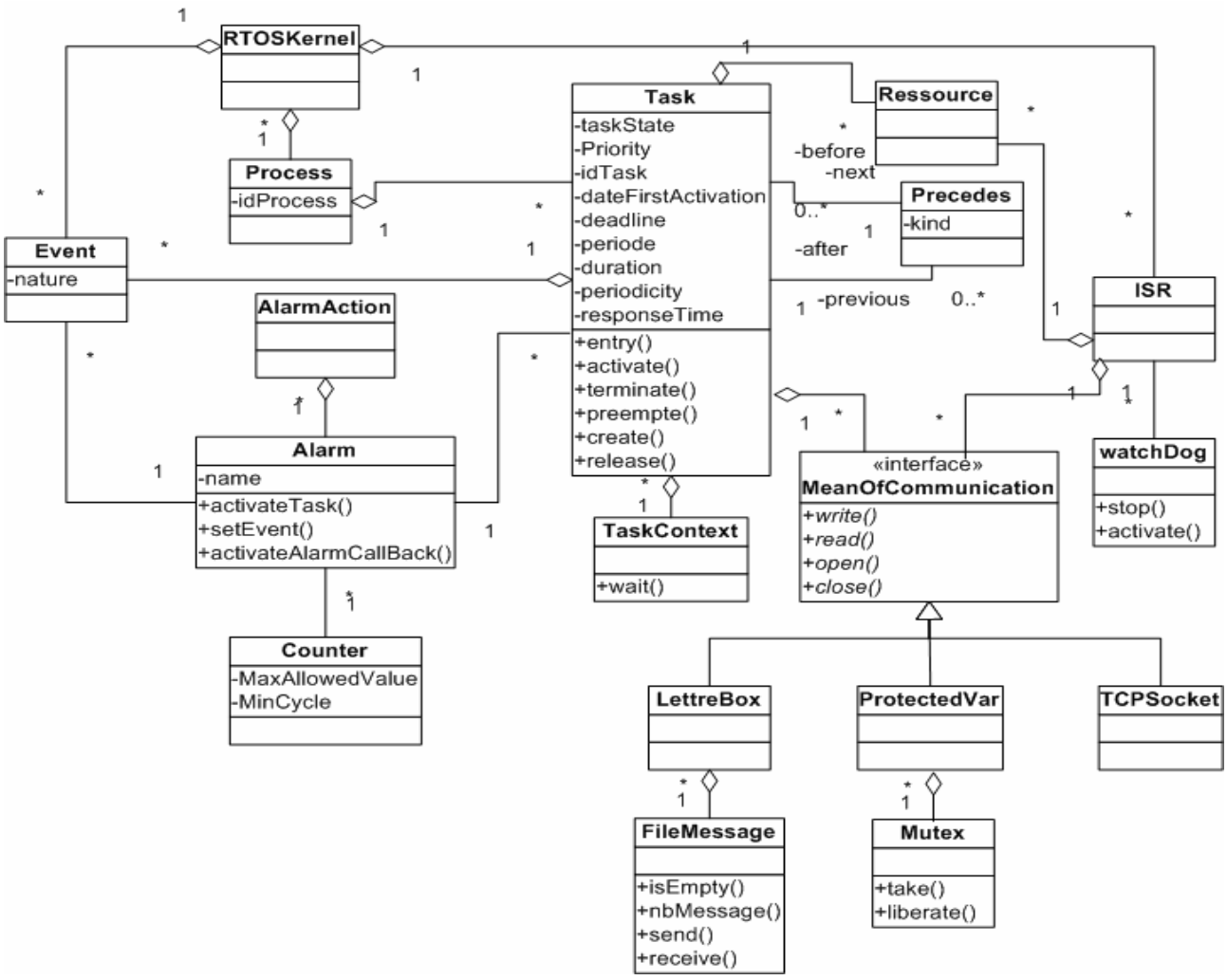

Figure 1: Static Model of the RTOS Structure

Regarding the enumeration of the states and the events, the code reacting the progression of the automat is localized in the method processEvent(). As for the enumeration of the states and the reification of the events, the code will be set out again between the method processEvent() and execute() of each class. As far as the reification of the states and the enumeration of the events are concerned, the code will be distributed between the method processEvent() and the method processEventPlay() of each class state. Finally, when the states and the events are reified, the code is distributed between the method processEvent() principal class, the processEvent() methods of the state class and the execute() methods of the class called event.

The last solutions based on enumeration and reification do not allow the representation of the concept of file messages related to the automat progression. Time is not taken into account. To overcome this problem, the use of the Active-Object pattern is, therefore, essential since it is effective for the achievement of the various policies of parallelism as shown in Figure 2.

Following the reification application of the states and events, as well as the evolution illustration of the automat, the final model represented in Figure 2 will be considered as the target model during the transformation process.

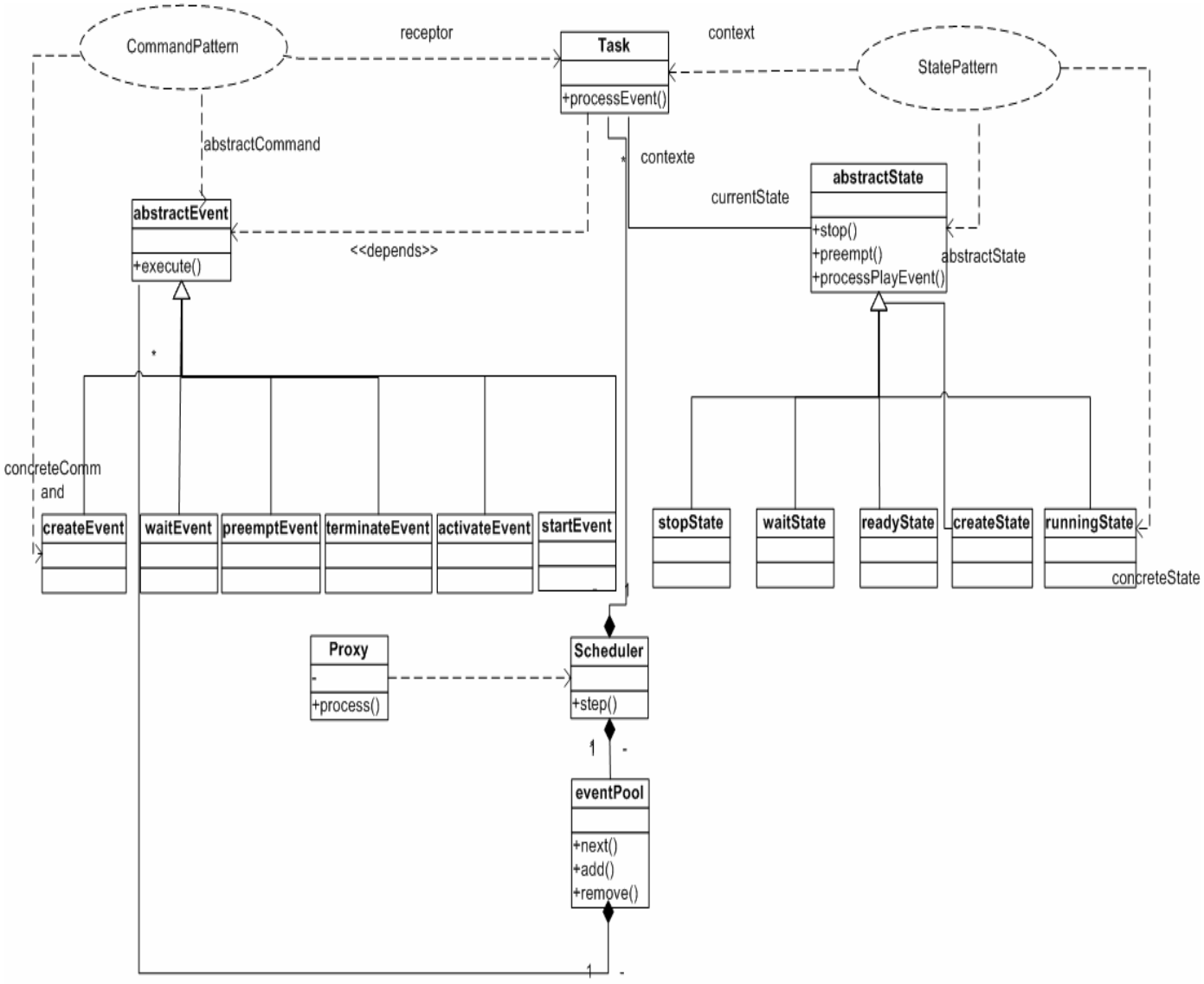

Figure 2: RTOS scheduler model

C. Specifying temporal constraints

In order to focus on software correction quality during model transformation, OCL is used to explain temporal constraints such as deadline and duration. Let us take the example of the attribute progress of the Task entity, whose value must always be lower than the deadline. This constraint is translated into OCL as shown in Figure 3.

```
context Task ::activate()
inv : periode = deadline
and Ci < deadline
and progress <= duration
pre: state = 'ready'
post: progress > progress@pre
and state = 'running'
```

Figure 3: Example of Temporal constraints specified with OCL

IV. CASE STUDY

The objective of this step consists in transforming an XML (Extensible Markup Language) source model obtained automatically from an UML source model to an XML target model. The model transformation is based on ATL (ATLAS Transformation Language) [5]. To describe the model transformation, the KM3 (Kernel MetaMetaModel) language is used. It makes it possible to define models according to meta-model MOF (Meta Object Facility) in a textual form.

The source model transformed corresponds to the diagram of class presented by Figure 1. The code corresponding to XMI (XML Metadata Interchange) based on XML offers a tree structure to our model by presenting the classes and the attributes in textual format.

To do these transformations, four tasks are taken with various characteristics. The scheduling of these tasks is made according to the scheduling algorithm Rate Monotonic [3].

The four tasks are specified in ecore format as shown in Figure 4.

```
<Task idTask="T1" " taskState="Create" dateFirstActivation="0"
periode="6"  Ci="1"
<Task idTask="T2" taskState="Create" dateFirstActivation="0"
periode="10" Ci="3" >
<Task idTask="T3" taskState="Create" dateFirstActivation="0"
periode="15" Ci="2">
<Task idTask="T4" taskState="Create" dateFirstActivation="0"
periode="20"  Ci="3">
```

Figure 4: Tasks example

The model transformation consists of the following three major stages:

- Stage 1 (Scheduling analysis): It checks the schedulability conditions
- Stage 2 (Initialisation): It puts all task instances in the event pool and initializes the tasks and events states
- Stage 3 (Scheduling): It ensures tasks scheduling through time progressing

A. Scheduling Analysis

The basic schedulability conditions for Rate Monotonic were derived from a set of n independent periodic tasks with a fixed priority. A set of tasks is schedulable by the RM algorithm if (1) is verified where Ti is the task period, n is the number of tasks and Ci is the worst case execution time.

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq n(2^{\frac{1}{n}} - 1) \quad (1)$$

Using ATL, Equation (1) is specified as shown in Figure 5:

```
helper context RTOSStructure!Task1 def: Somme():
String=
 RTOSStructure!Task1.allInstances()-
>collect(e|e.Ci.toInteger()/e.periode.toInteger())-
>sum()->toString();
helper      context      RTOSStructure!Task1      def:
ContraineRM(x:String):String=
 x.toInteger()*(2.exp(1/x.toInteger()-
1)).toString();
helper      context      RTOSStructure!Task1      def:
ConditionRM(x:String):Boolean=
 if                          self.Somme().toReal()<=
self.ContrainteRM(x).toReal()
 then true
 else false
       endif;
```

Figure 5: Schedulability conditions explained with ATL

B. Initialization

After checking schedulability condition, it is important to take instances of all classes of target model. The rule called Task2Task aims to transform Task source model elements to Task target model elements. It is described by Figure 6:

```
rule Task2Task{
 from
       s : RTOSStructure!Task1
 to
       w : RTOSSchudeler!Task (
             idTask <- s.idTask,
             taskState  <- s.taskState ,
             priority <- s.priority,
             dateFirstActivation                    <-
s.dateFirstActivation,
             deadline <- s.deadline,
             duration <- s.duration,
             periode <- s.periode,
             Ci <- s.Ci,
             Tsi <- s.Tsi,
             progress <- '0')
}
```

Figure 6: Tasks initialization

After transforming all instances of the task source model into the target model one, it is important to use a helper named isCreate which selects all tasks with created state. In the same way, four other helpers are implemented in order to cover the other task states. IsCreate helper is represented by Figure 7.

```
helper      context      RTOSStructure!Task1      def:
isWait:Boolean=
 if self.taskState='Wait'
 then true
 else false
 endif;
```

Figure 7: isCreate Helper

The previous helper is used via a rule called Task2CreateState. The rules «task2WaitState», «task2RunningState », « task2StopState », «task2ReadyState » are implemented with the same manner as shown in Figure 8.

```
rule task2WaitState{
from
      s : RTOSStructure!Task1 (s.isWait)
 to
        w : RTOSSchudeler!WaitState
        ( idTask<-s.idTask

              )
}
```

Figure 8: Task2WaitState helper

After initializing all the tasks and putting them in the eventPool, the four tasks must be scheduled using Rate Monotonic algorithm with a fixed priority. These tasks are scheduled according to each time unit. To do that, it is important to follow the following steps:

For each moment, the file message Eventpool gives an idea about the existing tasks, their current attributes, specially the progress and the state ones.

C. Scheduling

According to Rate Monotonic, the task with the highest priority is extracted from the event pool. To find the highest priority, all the instances of this class are collected in a

sequence. These instances are sorted using the instruction asSet(). This operation returns a set containing the elements of the self collection. Order is lost from a sequence or an ordered set. The helper that defines the highest priority is shown by Figure 9.

```
helper context RTOSStructure!Task1 def :Maxtasks
:String=
RTOSSchudeler!EventPool.allInstances()-
>collect(e|e.priority.toInteger()).asSet()->select(e
| e >= 0)->last()->toString();
helper context RTOSStructure!Task1 def : MaxTask
:String=
(if
self.priority.toInteger()=self.Maxtasks.toInteger()
       then self.idTask
       else false
       endif);
```

Figure 9: Extracting task with highest priority using ATL

During the last step, the task with the highest priority is running while the other ones are waiting or blocked. So, the class RunningState of the target model is instanced. Thus, after selecting the highest priority, the concerned task is defined via a helper called check2 and it is used in the following rule and represented by Figure 10.

Finally, after executing all the necessary rules, the target model is written in ecore format. It contains the four tasks scheduled according to Rate Monotonic algorithm. This file can be translated to any specific platform, i.e., to any computer programming language.

```
rule RTOSModelingt{
 from
     s : RTOSStructure!Task1(s.check2)
 to
 w : RTOSSchudeler!shudeler(
       idTaskm<-s.idTask,
       priority<-s.priority,
       dateFirstActivation<-s.dateFirstActivation,
       deadline<-s.deadline,
        progress<-'2'
       ),
       c: RTOSSchudeler!RunnigState(
                   idTask <- w.idTaskm,
                   priority<-s.priority,
                dateFirstActivation<-
s.dateFirstActivation,
                periode <- s.periode,
                duration<-s.duration,
                deadline<-s.deadline)
}
helper context RTOSStructure!Task1 def :check2
:Boolean=
       if self.priority =self.MaxTasks
       then true
       else false
       endif;
```

Figure 10: Instantiation of RunningState class

V. Conclusion

MDE is a well-known technique that has been successfully applied in ERTS design, especially for hardware and application modeling. In this paper, the mentioned technique is used for integrating RTOS modeling in high abstraction level. At this level, concepts undergo abstraction and are independent of realisation and specific platform execution. The major contributions are the definition of independent platform RTOS models and the production of the code that ensures tasks scheduling.

MDA pattern based on statecharts implementation technique is very effective since it leads to the creation of scheduling model. During model transformation, scheduling model is considered as the target model while the structure model is the source one. The proposed approach is validated through a case study on Rate Monotonic algorithm.

Future work includes the focus on annotating used UML diagrams with reliability attributes to identify and recover system from failures.

References

[1] A. Cuccuru, C. Mraidha, F. Terrier, S. Gérard. Templatable Metamodels for Semantic Variation Points. ECMDA-FA 2007: 68-82

[2] B. Combemale, S.Rougemaille, X. Crégut, F. Migeon, M. Pantel, C. Maurel. Expérience pour décrire la sémantique en Ingénierie des modèles. IDM6 LILE 26 28 juin 2006

[3] C. L. Liu , James W. Layland, Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment, Journal of the ACM (JACM), v.20 n.1, p.46-61, Jan. 1973

[4] D. Thomas, C. Baron, B. Tondu. Ingénierie dirigée par les modèles appliquée à la conception d'un contrôleur de robot de service. IDM6 LILE 26 28 juin 2006.

[5] Frédéric Jouault, Ivan Kurtev: Transforming Models with ATL. MoDELS Satellite Events 2005: 128-138

[6] F. Chauvel and J. Jézéquel. Code generation from UML models with semantic variation points. In S. Kent L. Briand, editor, Proceedings of MODELS/UML'2005, volume 3713 of LNCS, pages --, Montego Bay, Jamaica, October 2005. Springer

[7] Gamma, Erich, Helm, Richard, Johnson, Ralph, and Vlissides, John. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Longman Publishing Co., Inc., 1995.

[8] G. Pinter and I. Majzik. Impact of Statechart Implementation Techniques on the Effectiveness of Fault Detection Mechanisms, Proceedings of the 30th EUROMICRO Conference (EUROMICRO'04). 1089-6503/04 IEEE

[9] Harel, David and Naamad, Amnon. The STATEMATE Semantics of Statecharts. ACM Transactions on Software Engineering and Methodology, 5(4):293–333, October 1996.

[10] Ji Chan Maeng, Dongjin Na, Yongsoon Lee, Minsoo Ryu: Model-Driven Development of RTOS-Based Embedded Software. ISCIS 2006: 687-696

[11] L. Gomes, A. Costa. From Use Cases to System Implementation: Statechart Based Co-design, Proceedings of the First ACM and IEEE International Conference on Formal Methods and Models for Co-Design (MEMOCODE'03). ISBN 0-7695-1923-7/03 2003 IEEE.

[12] M. Cruz Valiente, G. Genova, J. Carretero. UML 2.0 Notation for Modeling Real Time Task Scheduling. Carlos III University of Madrid JOURNAL

[13] OMG Document Number: ptc/07-08-04. A UML Profile for MARTE, Beta 1 OMG Adopted Specification, August 2007

[14] Peng Yang, Francky Catthoor: Dynamic Mapping and Ordering Tasks of Embedded Real-Time Systems on Multiprocessor Platforms. SCOPES 2004: 167-181

[15] S. Bernardi and D. Petriu. Comparing UML Profiles for Non-functional. Requirement Annotations: the SPT and QoS Profiles, SVERTS 2004

[16] Shourong Lu; Halang, W.A.; Gumzej, R. Towards platform independent models of real time operating systems. Industrial Informatics, 2004. INDIN '04. 2004 2nd IEEE International Conference on Date: 26-26 June 2004, Pages: 249 - 254

[17] S. Kholgade, J.White, H. Reza: Comparing the Specification of a Near-Real Time Commanding System Using Statecharts and AADL. ITNG 2007: 355-360