Towards the automatic generation of real time operating systems applying UML/MDA

Yessine Hadj kacem[1], Adel Mahfoudhi[1, 2], Hedi Tmar[1], 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
adel.mahfoudhi@fss.rnu.tn
[2] Department of Computer Science, Science Faculty of Sfax
Road Soukra km 3,5 BP : 802 -- 3018 Sfax TUNISIA
mohamed.abid@enis.rnu.tn

***Abstract*— This paper presents our contributions to the specification and the design of Real time embedded systems, which require runtime guarantees from their underlying environment. It is not sufficient to reach these guarantees; performance and timing constraints but it is desirable to employ Real time operating system RTOS. With the model driven approach MDA, and specially, with a UML (Unified Modeling language) profile, software designers can focus on their business logic. That is to say MDA enables them to specify the functions and the properties of the RTOS with a platform independent model.**

This work is one step of the RTOS modeling, resting on MDA Model driven architecture). The model driven engineering MDE based solution proposes to model the structure of a RTOS. It suggests the implementation of statecharts relating to the state of a process. Using this approach, real time constraints can be translated by defining the semantics variants of statecharts. The main goal of the suggested proposition is to generate the code automatically.

***Key words*— RTOS modeling, statecharts semantics, statecharts implementation, UML/MDA, automatic code generation**

I. INTRODUCTION

THE correctness of the computed results in real time embedded systems depends not only on the right results but also in the time during which they are provided. Their bad function can have serious effects (economic, legal, human, etc) because of the overload or the deadline expiry for some services. These systems require runtime guarantees from their underlying environment. To reach these guarantees, performance and timing constraints, these systems must be provided by software called RTOS.

So, several constraints, namely, real time ones, are imposed during their design phase. The checking of the system properties at a preliminary stage could reduce the problem impact. In fact, the real time design passes through different abstraction layers in order to automate the transition between them. In regard to the bottom layers, there exists many synthetic tools; the only problem concerns the CAD (Computer Aiding Design) of the highest level and it is here where our work lies.

Currently, oriented objects modeling supported by UML standard brings effective solution to the problems related to the real time systems design. Its realisation is possible through the extension and/or the restriction of this standard via UML profile. However, the capacities of real time behaviour specification of a given application have not been completely satisfactory yet. Indeed, these methods, recently industrialized, provide solutions in terms of concurrent application, but they remain insufficient especially, for the expression of the no functional properties and the integration of the RTOS modeling.

This paper proposes an approach that supports the RTOS modeling starting from high level design, and ending up with the implementation code which can be used in different platforms.

This position paper starts with a brief discussion of some related works. The proposed approach comes into in the next section. In this section, the models of the RTOS structure and the scheduler are introduced through the implementation of statecharts. Section four presents a case study. The paper closes with some final conclusions and an outlook on future work.

II. RELATED WORKS

When the specificity of each UML profile such as SPT [11], QoS/FT [11] and MARTES [10] are examined, it is concluded that the focus is on to 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 of control. Even, if 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 fact, 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 therfore 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.

In [12], the authors' work is based on two independent class diagrams: a diagram describing the structure and another describing the scheduler. These models, related to the structure and the scheduler, explicitly separated, suffer from major limitations; namely the coherence between diagrams and the definition of temporal semantics. In fact, the diagram used to characterise the scheduler is a static one. Thus, it can't cover the temporal behaviour of the RTOS, it must also have to be complementary to the structure model via a good expression of the follow-up of the real time process evolution. For this, a methodology assuring the coherence between used diagrams and the support of scheduling model is important.

Based on a real time library VxWorks written in C, DAV et al. [3] carry out the transformations necessary to lead to a UML diagram. This downward transformation leads to some entities specifying the components of a real time system. The bond between them is left with the load of the designer. This approach is restricted with a static description. Thus, the behaviour can be dealt with introducing attributes describing the state of a task state progression into time or by defining a reflexive precedence relation or by adding attribute showing time evolution. This technique is called the definition of operational semantics [2].

According to [9], a scheduling algorithm can be modeled using the sequence diagram and some stereotypes provided by the SPT profile. This proposal handicaps resides at the existence of a great number of scheduling algorithms, and consequently the designer will be opposite to a several scheduling algorithms using a succession of sequence diagrams and he will be vis-à-vis the problem of integration of the whole of these diagrams in MDA process.

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. To lead to the model of scheduling, the temporal and transitional semantic of the statecharts relative to the various states of a real time process is defined.

III. PROPOSED APPROACH

A. Over View

To overcome the limitation of the previously mentioned works, this proposed approach presents a step ensuring coherence between various used UML diagrams and covering the behaviour aspect of the system like real time constraints. First, we define the model of the RTOS structure. Then, a statecharts diagram related to the entity Task presents the temporal behaviour of a real time task. This diagram is annotated with OCL constraints. After that, we define the temporal semantic presented by the statecharts [1]. While defining the variant semantic points of the statecharts, some techniques such as the reification and the enumeration of the states and the events are applied. The integration of design patterns is chosen for the re-use of existing and testing software components, rather than to recreate new models for the implementation of the statecharts. The final model corresponds to the target model during the stage of model transformation. As a final stage, the code is generated automatically.

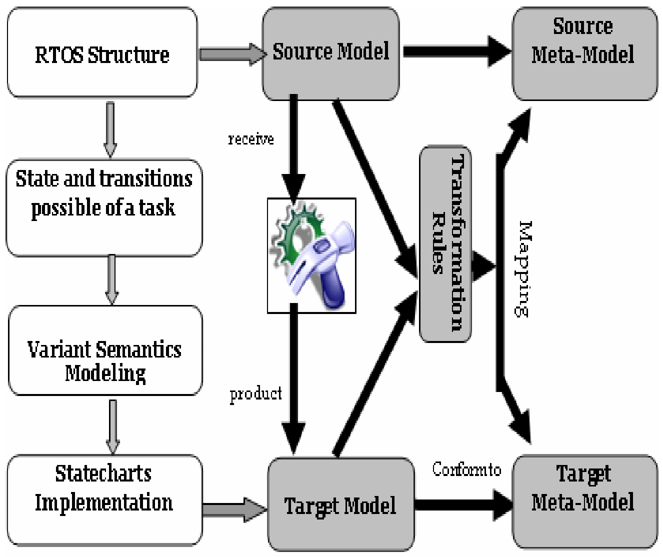


Figure 1: Proposed Approach

B. RTOS structure

Two diagrams are proposed for the description of the RTOS structure, a class diagram describing the major components of the RTOS, and a statecharts diagram modeling the behavioural aspect of a real time Task. To guarantee the correction quality of the system, the statecharts diagram is annotated by some OCL rules.

The class diagram which is presented by figure 2 is considered as the source model during the stage of model transformation that has an important role in Model Driven Engineering; it is represented by the following entities:

- Task: It is the most important component of the RTOS core. 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. It makes, in this context, 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 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

- MeanOfCommunication: It is an abstract interface which manages data between active objects. The class ProtectedVar which implements this interface, associates a mechanism of data protection (semaphore). In addition LettreBox uses a file of messages.
- Watchdog: The ISR contains one or more watchdog timers. The watchdog could possibly provide debugging information
- Precedes: It illustrates the dependence of a task with another one.

.

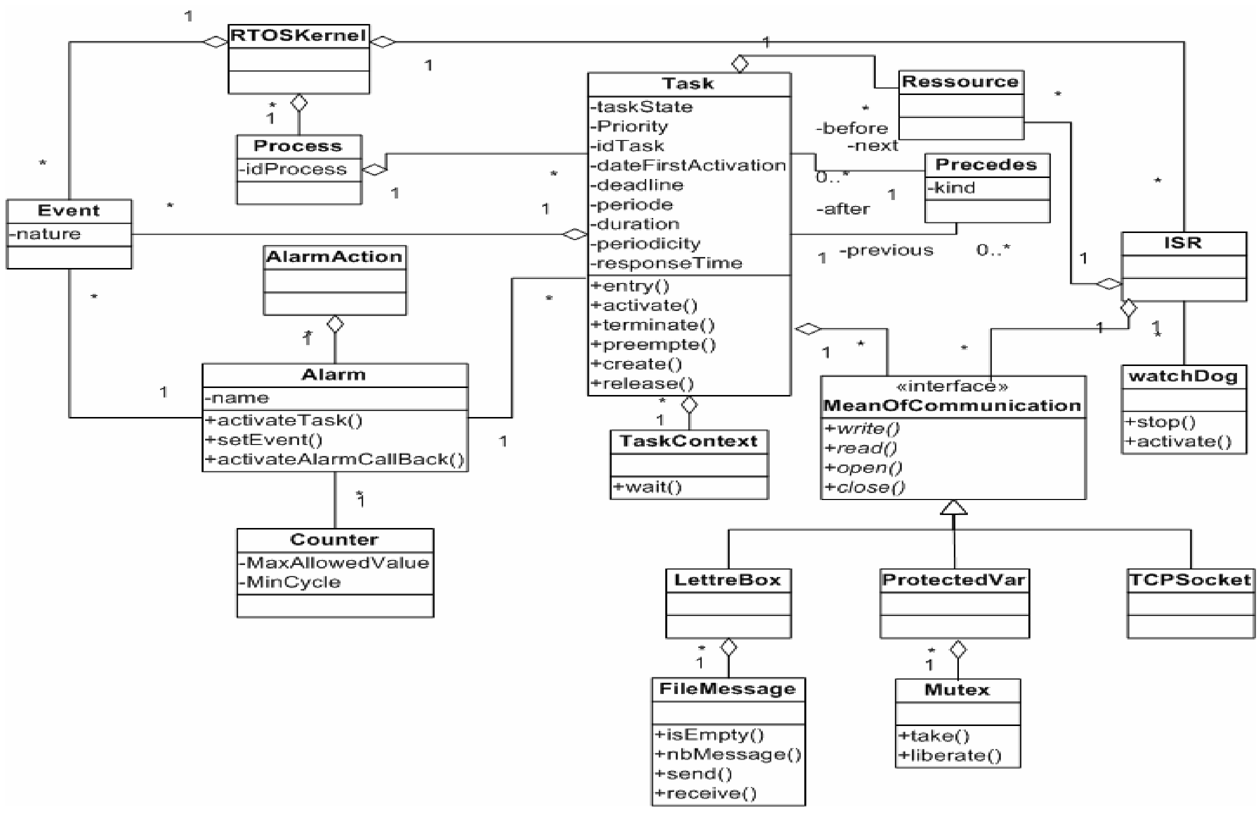


Figure 2: Static Model of the RTOS Structure

For the dynamic model of the RTOS structure, it is described by the statecharts diagram. Before presenting the appropriate diagram, let us remind that each state of a task running on RTOS can take only one of the following values:

- Waiting: waiting for synchronization;
- Running: running on the processor;
- Ready: waiting to be selected by the RTOS to enter the Running state
- Suspended: task finished or stopped by the scheduler
- Created: new task

The structure of the statecharts diagram is nevertheless given a precise specification [13], which is required for tool interoperability. It can not easily be understood. So UML 2.0 Statecharts present some semantic point variation. The definition of this semantic will be detailed in the next section. It corresponds to the target model during the model transformation.

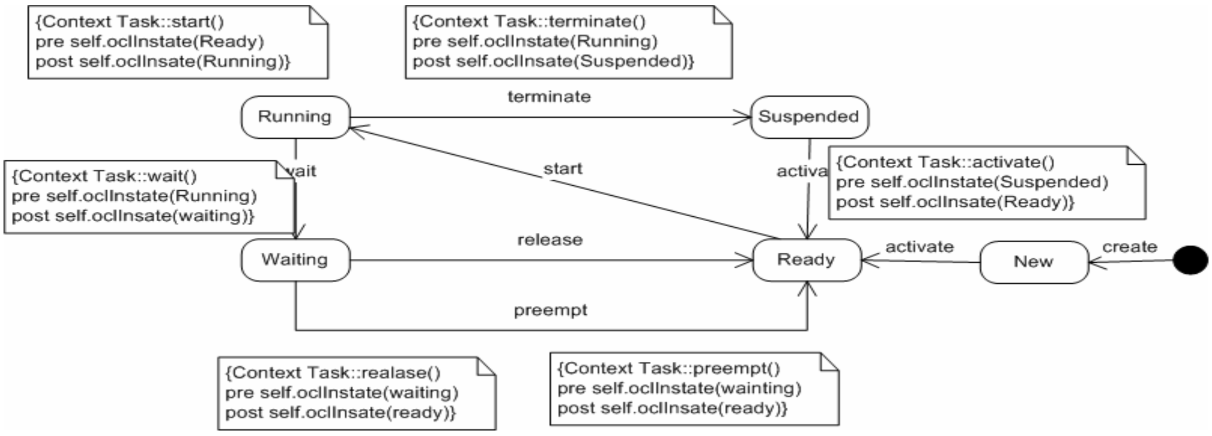
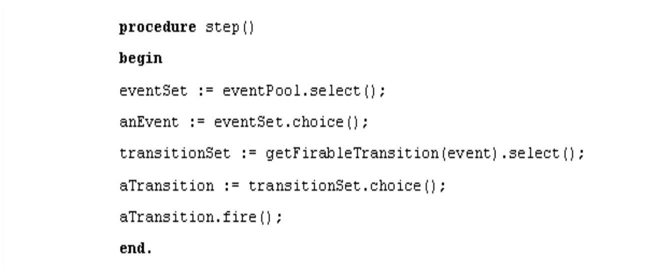

{Context Task::start()
pre self.oclInstate(Ready)
post self.oclInsate(Running)}
{Context Task::terminate()
pre self.oclInstate(Running)
post self.oclInsate(Suspended)}
Running
terminate
Suspended
wait
start
activa
{Context Task::wait()
pre self.oclInstate(Running)
post self.oclInsate(waiting)}
{Context Task::activate()
pre self.oclInstate(Suspended)
post self.oclInsate(Ready)}
Waiting
release
Ready
activate
New
create
preempt
{Context Task::realase()
pre self.oclInstate(waiting)
post self.oclInsate(ready)}
{Context Task::preempt()
pre self.oclInstate(wainting)
post self.oclInsate(ready)}

Figure 3: Dynamic model for the RTOS Structure



```
procedure step()
begin
eventSet := eventPool.select();
anEvent := eventSet.choice();
transitionSet := getFirableTransition(event).select();
aTransition := transitionSet.choice();
aTransition.fire();
end.
```

Figure 4: The run-to-completion procedure

A. Scheduling model

Statecharts have been adapted with an informally or undefined semantics. The semantic variation points principally concern 3 aspects: time management (synchronous vs. asynchronous), the event selection policy, and the transition selection policy.

Harel [7] represents the semantics of the statecharts based on the description of a run-to-completion step as illustrated in figure 4.

A set of approaches [6, 8] was proposed in the literature in order to define this semantics and implement the statecharts. For our work, we choose the approach proposed by [4]. This technique is based on the enumeration and the reification.

The state of the Task entity can take the following values :{ created, new, waiting, ready, running, stopped}. As for, an event has these values :{ terminate, activate, start, wait, preempt, release, create}.

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 utilisation of the state pattern [5].

To reify and select the right transition events, the command pattern [5] is applied to the entity Task. (see figure 5).

In the light of the solutions given previously and in order to ensure the progression of the automat, it is necessary to focus on the deterministic aspect of the system, it is essential to determine the state running of the automat and the behavior to be adopted according to the event which has occurred.

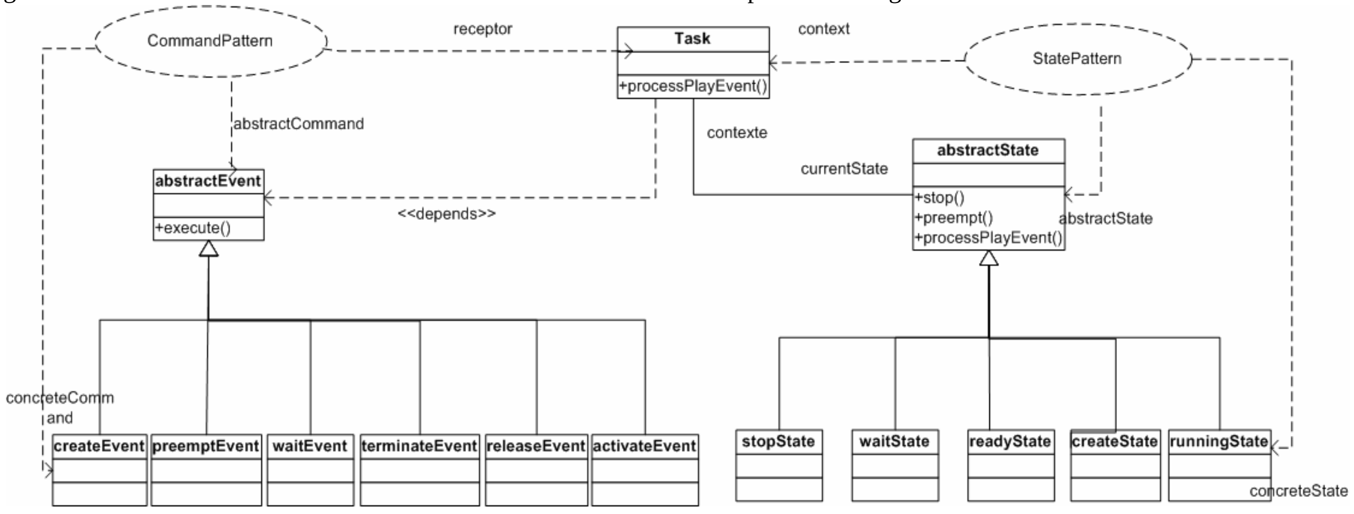


Figure 5: Application of the state and command pattern to the Task entity

When it acts of the enumeration of the states and the events, the code reacting the progression of the automat is localised in the method processEvent(). As for the enumeration of the states and the reification of the events, the code will have set out again between the method processEvent() and execute() of each class. Concerning the reification of the states and the enumeration of the events, the code will be distributed between the method processEvent() and the method processEventPlay() of each class state. Finally, when we reify the states and the events, the code is distributed between the method processEvent() principal class, the methods processEvent() of the classes states and the methods execute() of the classes events.

The last solutions based on enumeration and reification do not allow representing the concept of file messages related to the automat progression. Time is not taken into account. To overcome this problem, the use of the pattern Active-Object [5] is therefore essential. This owner is thus effective for the achievement of the various policies of parallelism as it is shown in figure 6.

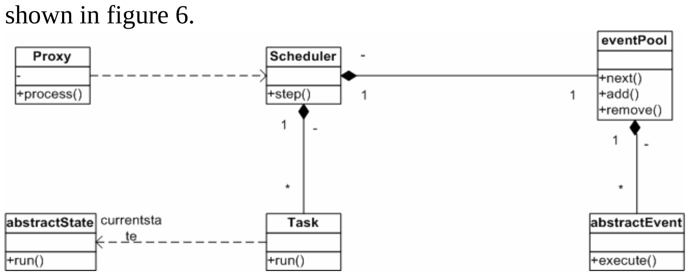


Figure 6: Active-Object applied to Task entity

Following the application of the reification of the states and the events, as well as the illustration of the evolution of the automat, the final model corresponding to the target model during the models transformation is represented by the model below.

A. *Code generation*

The objective of this work consists in transforming a source model XML (Extensible Markup Language) obtained automatically starting from an UML source model, in a target model XML. To carry out the transformations, we are based on a model transformation using ATL language. To describe the model transformed, the KM3 (Kernel MetaMetaModel) language is used. It makes it possible to define models according to meta-model MOF in a textual form.

The source model transformed corresponds to the diagram of class presented by figure 2. 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 form.

The target model is the model that we want to obtain after the execution of the transformations applied to the source model. It was presented above by figure 7.

IV. CASE STUDY

It should be noted that the example used at the time of the transformation is taken adequately since the objective of our work is to show right the feasibility of the use of the MDE for the integration of RTOS modeling for embedded system design. So transformations are focused just on ensuring tasks scheduling.

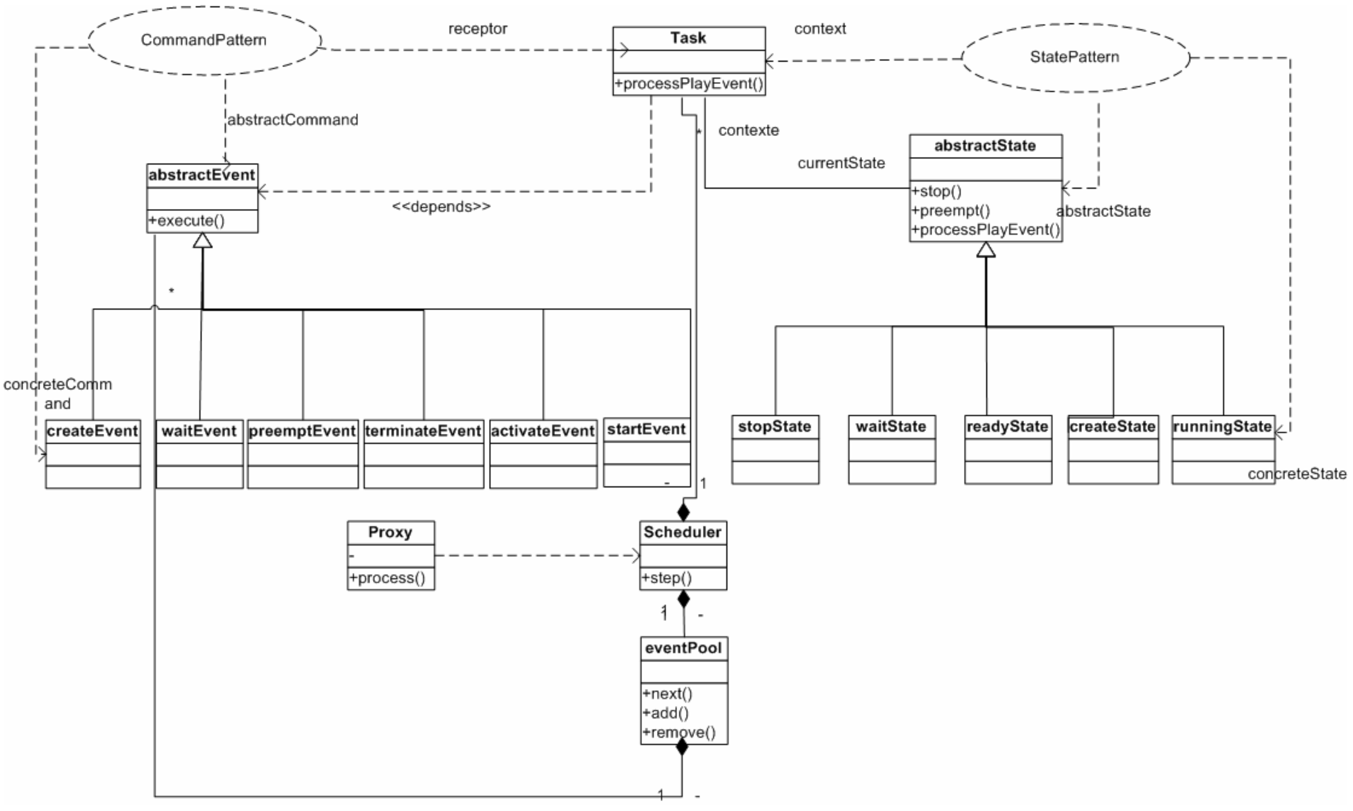
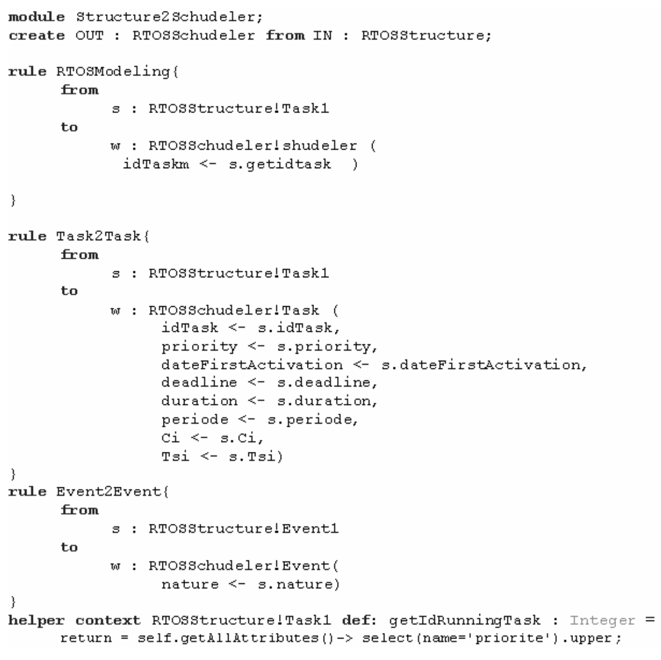


Figure 7: RTOS scheduler model

In order to do that, four tasks are taken with various characteristics. During the writing of the transformation rules, the scheduling of these tasks is made according to the scheduling algorithm Rate Monotonic as shone in figure 8.



```
module Structure2Schudeler;
create OUT : RTOSSchudeler from IN : RTOSStructure;

rule RTOSModeling{
      from
            s : RTOSStructure!Task1
      to
            w : RTOSSchudeler!shudeler (
             idTaskm <- s.getidtask  )

}

rule Task2Task{
      from
            s : RTOSStructure!Task1
      to
            w : RTOSSchudeler!Task (
                  idTask <- s.idTask,
                  priority <- s.priority,
                  dateFirstActivation <- s.dateFirstActivation,
                  deadline <- s.deadline,
                  duration <- s.duration,
                  periode <- s.periode,
                  Ci <- s.Ci,
                  Tsi <- s.Tsi)
}
rule Event2Event{
      from
            s : RTOSStructure!Event1
      to
            w : RTOSSchudeler!Event(
                  nature <- s.nature)
}
helper context RTOSStructure!Task1 def: getIdRunningTask : Integer =
      return = self.getAllAttributes()-> select(name='priorite').upper;
```

Figure 8: Transformation rules

I. Conclusion

The present paper demonstrates that the RTOS can be modeled in high level design. The challenge consists of the use of existing UML profiles in order to integrate the RTOS modeling by the definition of the transition and the temporal semantics.

At the level of the integration of the RTOS modeling in MDE approach, concepts endure abstract and are independent from realisation and specific platform execution.

The implementation of the variant semantic points offered by UML statecharts provides an efficient way to specify task management and real time scheduling.

Future work includes the focus on the transformation rules to generate the code.

References

[1] Arnaud Cuccuru, Chokri Mraidha, François Terrier, Sébastien Gérard. Templatable Metamodels for Semantic Variation Points. ECMDA-FA 2007: 68-82

[2] Benoit Combemale Sylvain Rougemaille, Xavier Crégut, Fedéric Migeon Marc Pantel Christine Maurel. Expérience pour décrire la sémantique en Ingénierie des modèles. IDM6 LILE 26 28 juin 2006

[3] Dave Thomas Claude Baron Bertrannd Tondu. Ingénierie dirigée par les modèles appliquée à la conception d'un contrôleur de robot de service. IDM6 LILE 26 28 juin 2006.

[4] Franck Chauvel and Jean-Marc 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

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

[6] Gergely Pinter and Istvan 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

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

[8] Luís Gomes, Anikó 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.

[9] Maria Cruz Valiente, Gonzalo Genova, Jesus Carretero. UML 2.0 Notation for Modeling Real Time Task Scheduling. Carlos III University of Madrid JOURNAL

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

[11] Simona Bernardi and Dorina Petriu. Comparing UML Profiles for Non-functional. Requirement Annotations: the SPT and QoS Profiles, SVERTS 2004

[12] Shourong Lu Wolfgmg A. Halang Roman Gumzej. Towards Platform Independent Models of Real Time Operating Systems. 0-7803-8513-6/04/ Q2004 IEE

[13] Sudhanwa Kholgade, Jamie White, Hassan Reza: Comparing the Specification of a Near-Real Time Commanding System Using Statecharts and AADL. ITNG 2007: 355-360