Workshop on “MODEL DRIVEN ARCHITECTURE in the
SPECIFICATION, IMPLEMENTATION and VALIDATION of
OBJECT-ORIENTED EMBEDDED SYSTEMS” (SIVOEES)

UML’2003, October 2003, San-Francisco (US)

MDA for embedded systems dedicated to process control

Jean-Louis Houberdon, Jean-Philippe Babau
CITI: Centre d’Innovation en Télécommunication et Intégration de service
INSA Lyon, F69621 Villeurbanne Cédex - France
jean-louis.houberdon@insa-lyon.fr, jpbabau@if.insa-lyon.fr

1 Introduction

The main interest of the Object Oriented (OO) approach is its ability to improve the quality of software, mainly due to modularity, extensibility, and reusability. This approach is widely acknowledged by the industrial world and reinforced by the emergence of a standard (Unified Modeling Language [UML01]). Moreover, the MDA approach (Model Driven Architecture [MDA03]) has been defined by the OMG (Object Management Group) in order to improve the development of systems. This approach is based on models and model transformations.

In a classical software life cycle (V shaped, waterfall, incremental, ...), development is defined as a set of consecutive phases. So, according to the MDA approach, each phase produces a model. Each model is used as starting point for the next phase. In MDA, a phase corresponds to a model transformation.

MDA is recent and is not widely used for the development of embedded real-time systems. The domain of this paper deals with applications of process control. In such applications, a physical event (interrupt, physical expiration) triggers an action which produces a command, with some associated real-time constraints like deadlines.

This paper aims to propose a modeling approach for development of embedded real-time process control systems while following the MDA approach. This approach aims to conform to the MDA approach. In the first section, the MDA approach and the associated modeling approaches are described. Then in a second section, after the context has been described, we identify and describe the set of the needed models and the set of the associated transformations. Finally we conclude with some applications and perspectives of this work.

2 MDA

2.1 MDA approach

2.1.1 Introduction

The MDA approach “starts with the well-known and long established idea of separating the specification of the operation of a system from the details of the way that system uses the capabilities of its platform” [MDA03]. The MDA approach is based on two essential concepts: the PIM (Platform Independent Model) and the PSM (Platform Specific Model). The PIM is a specification of the application independently of the implementation specificities. This model describes the application in term of structure and behaviour. This view abstracts implantation details (user interface, implementation language, ...).

The PSM is a model of the application which integrates all platform constraints. In a development process, several PSM’s may be defined. It is useful for example to deploy the same PIM specification on

several physical sites with many different technologies. In addition, in case of a change of technology, the MDA approach can increase portability and reduce costs. The platform has to be remodeled. With the PIM already done and the constraints of the new platform, a new PSM is then produced.

2.1.2 Principles of MDA for the development

During system development, using informal models and non explicit transformations introduces errors. Moreover, due to a lack of formalisation, a change in the requirements or in the platform specifications is hard to integrate in a new version of the application. So a methodology must formalise models and associated model transformations. This is the challenge of MDA. In top-down development (chosen here) of softwares, several kinds of model transformations exist: refinement, abstraction, mapping and optimisation.

The refinement produces a new model from a previous one by the adjunction of pieces of information to the initial model. For example, in an incremental development, the refinement permits to add properties, called *applications* in MDA, to the initial model.

The transformation called abstraction produces a model, called *view* in MDA, which contains a subset of the properties of the initial model. This is often used in order to prove properties on the initial model. For example, it is possible to produce a model which contains only time properties in order to check time performances of the abstracted model.

The mapping is a model transformation which realises the fusion of a PIM with a PM (Platform Model) in order to produce a PSM. In MDA, this fundamental operation can be applied with several strategies. It is possible to mark the PIM in order to prepare the production of a PSM. It is also possible to add additional information to the PIM in order to produce a PSM.

Finally, it is always possible to optimise a model. The optimised model keeps the same functional and behavioural properties than the initial one. Only the structure is modified. This transformation makes the model smaller for the user or is used to improve the QoS (Quality of Service) of the system.

In a development conforming to MDA approach, a first PIM is established. This PIM is then refined in order to represent the set of *applications* of the system. Next, one or more PSM's are created with this PIM. This PSM's are the results of the mapping of several PM's. Each of this possible PM depends on one specific technology. The PIM's and PSM's are optimised in order to reduce their complexity. The last PSM produced is used to generate code. This code generation should be automatic.

2.1.3 Modeling language

The using of MDA leads to choose a modeling language. As suggested by the OMG, UML (Unified Modeling Language [UML01]) is a good candidate. UML is a standard for OO modeling. This language is extensible, so it can be adapted easily to specific constraints of a given domain. The extension in UML is done via stereotypes (elements whose semantic is extended) grouped into profiles. Moreover, UML profiles have been defined in some domains, such as "Scheduling, Performance and Time" (SPT) [SPT02] for real-time. This profile will be used to model some specific aspects of the aimed domain. Once the language is chosen, it is necessary to define the modeling strategy.

2.2 Modeling approaches

2.2.1 Aspects, components and patterns

Aspect Programming (AOP) [KLM+97] considers that each functionality of the system is an aspect. This mechanism is very flexible and improves the modularity of crosscutting concerns. Aspects are not coded in the main code, but in separate modules. Component programming is closed to object concept. A component is described with its interface, like an object. This interface shows the services that it offers.

But, the component also has an interface which shows the services it needs. So this approach permits to model the interdependencies between components.

Several works ([TNHN02], [GBNG02], [GNS+02]) use aspects and components in order to introduce QoS constraints into the models and in order to apply the principles of MDA. But, these papers always consider the using of middlewares such as CORBA^{TM} [COR] or J2EE^{TM} [J2E]. These middlewares need a lot of resources (memory, processor). This is not appropriate in real-time systems development. The systems we consider are dedicated and embedded. It is necessary for them to be efficient. Moreover, in such systems, the operating system is a dedicated real-time operating system (RTOS) which does not support such middlewares.

On the other hand, MDA proposes to use types, patterns and meta models for the modeling. A design pattern is "a description of communicating classes and objects which are appropriate to solve a general problem of conception in a given context" [GHJV95]. Design patterns are very useful to formalise typical programming schemes of the domain of embedded systems [Dou02]. But, there are no design patterns for the modeling of real-time systems dedicated to process control. In consequence we propose to establish patterns for the modeling (according to MDA) of the real-time systems dedicated to process control.

2.2.2 Meta models for the design patterns

The main problem caused by the using of design patterns is that pattern description is based on one or more examples. The constraints which are applied on the model are expressed within the examples with specific words and names. Unfortunately, the semantic of the model can not be constrained via naming rules. For example, making a *read()* method write something is unfortunately possible.

A solution is to use meta models which describe in a more precise manner the pattern. The approach [KFGS02] is based on the notion of roles. A role can be played by a class or an unspecified model element according to the considered role. The role induces structural and behavioural constraints on the element playing the role. For example, it is possible to limit the number of elements playing a given role. The main interest of this approach is the ability to specify constraints in the meta model with OCL (Object Constraint Language) [UML01]. These constraints help the user of the roles while constraining him. Thus he is led to do fewer errors in his models. In addition, some studies are led actually to check automatically the constraints in OCL on a model [CS02].

2.3 Conclusion

In this first section we have defined model transformations needed for development. We have chosen a modeling language, UML. Finally we have chosen a strategy based on patterns expressed with meta models based on role for the development of real-time systems dedicated to process control. We present now the proposed methodology.

3 Domain context

3.1 Control command systems

The aimed systems in this study are control command systems dedicated to process control. These systems realise operations of command synthesis. The command synthesis corresponds to the production of specific outputs to actuators in order to follow a given command. This command is produced following measured datas on one or more sensors. The measure activity is a set of signal processing. Then the result of the measure activity is computed (taking into account the process state) in order to produce the corresponding command. The command synthesis is done either in open-loop mode or in closed-loop mode (cf. figure 1).

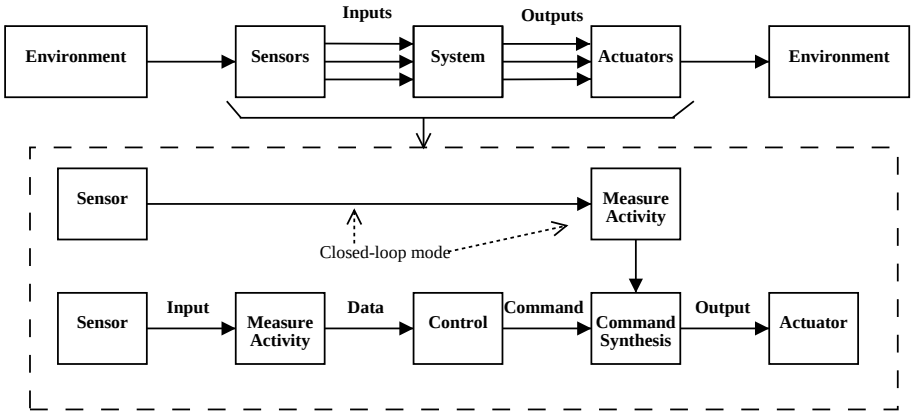

Figure 1: Generic architecture of a control command system

3.2 Real-time

In real-time, three kinds of constraints are usually considered: activation constraints, age of a data and deadlines. These constraints are part of the constraints described in the UML profile SPT (Scheduling Performance and Time [SPT02]).

An activation constraint is applied on an input event and indicates its arrival law which can be either periodic, aperiodic, sporadic or following a probabilistic law.

The age of a data is the maximum time between the last update of the data and all retrieving operations of the data which follows the update. If the age of a data is superior to a given threshold, the data is considered invalid and prevent the normal functioning of the system.

The deadline expresses the maximum time between an input event and the end of a corresponding operation.

3.3 Architecture

In this paper we limit to a monoprocessor architecture managed by a RTOS (Real-Time Operating System). The interactions between physical elements (sensors and actuators) and the software part of the system is done through the Input Output System (IOS) associated to the RTOS. The IOS manages device drivers dedicated to each sensor and actuator. So each device driver abstracts the physical element it is associated with. If a device driver is associated with a sensor in input mode, a *read()* operation will permit to read the value it contains. On the contrary if the driver is associated with an actuator in output mode, a *write()* operation will permit so send orders to the device.

Now, the domain is defined. In the next section we will identify the models and the transformations.

4 Meta models and transformations

4.1 Methodology

Conforming to the MDA approach, the first model is a PIM (cf. figure 2). In this model, the constraints of the application are described. In real-time systems, time constraints are expressed in the requirements. In consequence, they are introduced at the PIM level.

Then the PSM is produced integrating the platform constraints. In the aimed domain, two kinds of platform constraints exist: the constraints related to the communication with the process and the constraints related to the schedule of actions (task model). The first constraints describe the interactions between the system and its environment. The other constraints describe the way the model must be

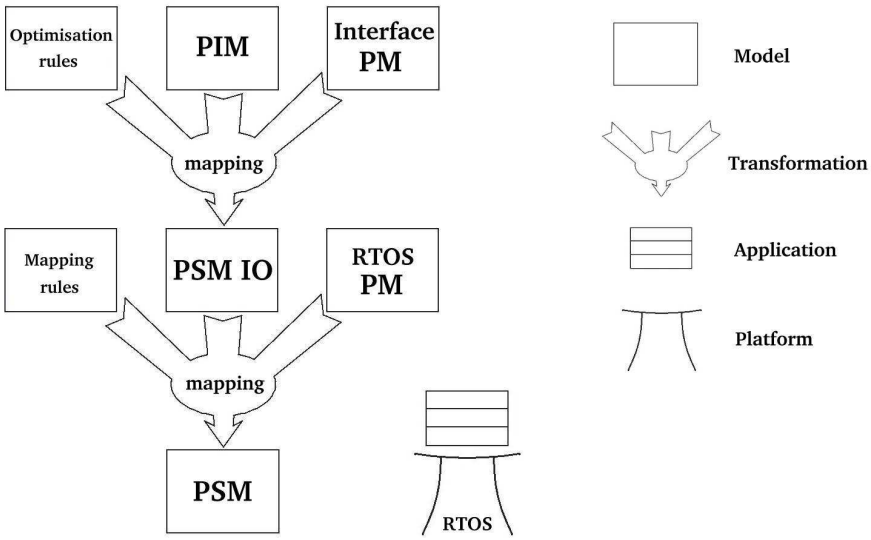

Figure 2: The set of models to be used for the development of Control Command systems

executed during execution time. We have chosen to divide the mapping in two phases: first we introduce the constraints related to the interface with the environment, called Input/Output (IO) constraints and then the other ones. The reason is that the communication with the environment introduces new elements and new tasks in the model. So scheduling aspects are to be considered afterward.

For IO constraints, the communication with the hardware is based on device drivers. So at this level, the PM (called Interface PM) describes driver services. Indeed, in embedded systems, the memory is limited and the size of the code must be small. As we will see, the approach produces models with many classes. So after the mapping of the Interface PM which produces the first PSM (called PSM IO), an optimisation phase is necessary to reduce the number of objects and method calls.

Then we introduce scheduling constraints with a new mapping. This mapping is done with a PM which is a task model of the services offered by the RTOS. This work is based on a generic task model which is used as a generic Platform Model. The last obtained model is the input of an adequate code generator and the code is generated. During code generation, optimisations on the PSM are necessary in order to improve its performances.

In the next subsections of the study, we present the principles of the models we have seen and some problems caused by the successive transformations like the constraints derivation and the model optimisation.

4.2 First step: the PIM

The aim of the PIM is to model the properties which are independent of the platform. In these constraints we have *applications* (“What must the system do ?”) and non-functional constraints (for instance real-time properties). A classical control command system is divided in three layers: Input, Computing, Output. So the PIM is also divided in three layers conforming to the classical control command system architecture.

The first layer describes system inputs. These inputs are called datas (cf. *Data* role in figure 3). At the PIM level, inputs are not dependent of the platform. Datas represent a perfect view of the environment. Thus, for example, in order to make an obstacle avoiding application, an input class *Map* plays the *Data* role and represents the map of the environment. This class can not represent the values of the sensors,

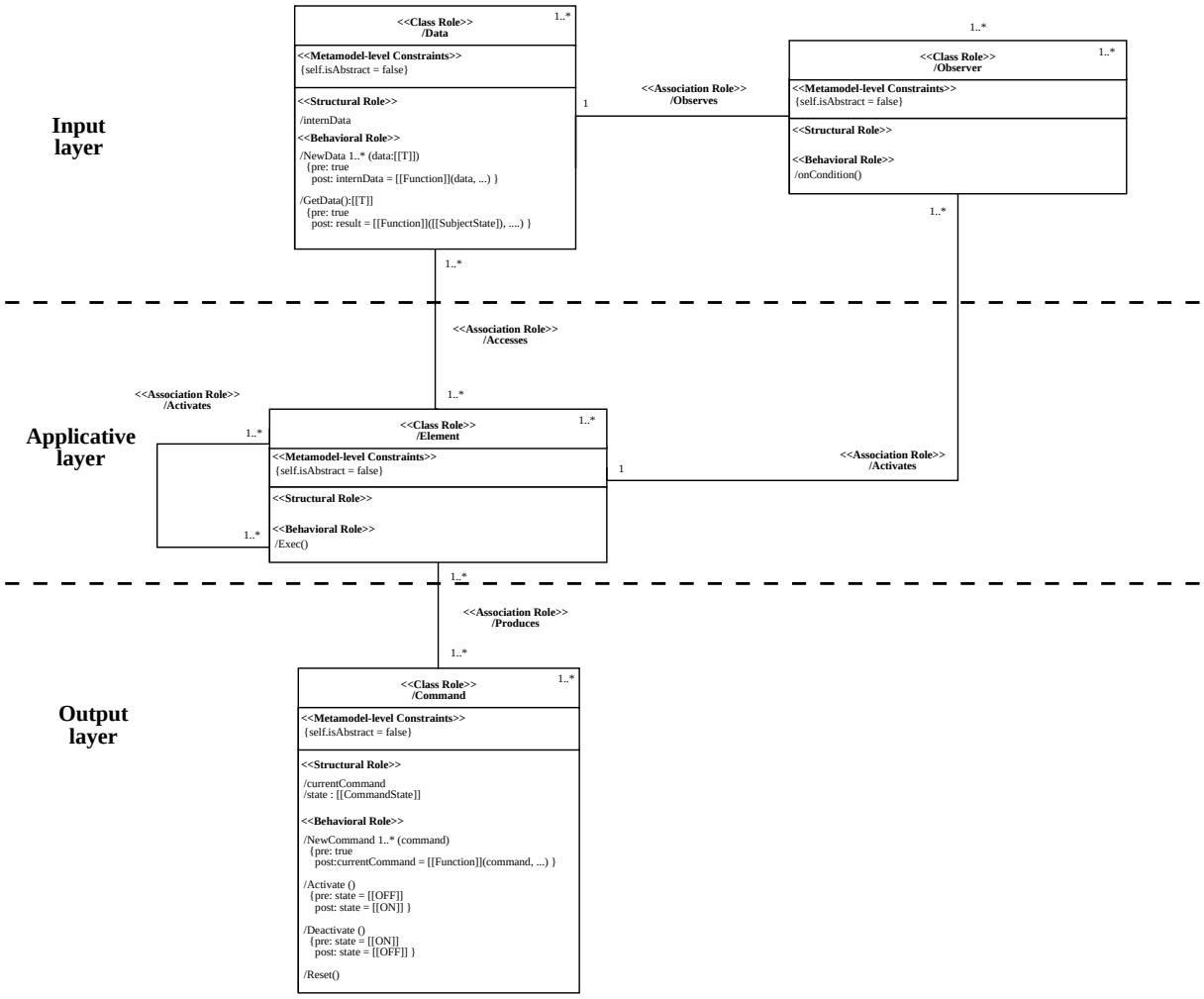

Figure 3: The PIM meta model

because at this level, we do not have information about the way the map will be constructed. According to the proposed meta model, in the PIM, the *Data* role is played by a class which has at least two methods: one to retrieve (*GetData*) the content of the stored data and one to modify (*NewData*) this content. We can have more than one method to modify the content of the data (*Increment(), Decrement(),* ...).

In a reactive system, each action is activated by an event. An event is either a data which exceeds a threshold or the modification of a data or an event coming directly from the environment. In order to separate a data from the events it can generate when it is modified, we introduce an *Observer* role. This role is similar to the design pattern *Observer* [GHJV95]. But the whole pattern is not adapted to the aimed domain: Real-time applications must be predictive. *Attach()* and *Detach()* operations introduce dynamic behaviours that we can not consider, so these operations are suppressed. The observers act as event generators for the system: they make it reactive.

The next layer is the applicative layer. It models the applications of the system i.e. transforming inputs into high level commands. In this layer, the *Element* role is played by a class which offers services in order to produce commands.

Finally the output layer is constituted with classes playing an *Command* role. This generic *Command* can be modified (*NewCommand(command)*), activated (*Activate()*), deactivated (*Deactivate()*) or reset (*Reset()*). Activating and deactivating change the state of the *Command*. The *Command* is applied or not according to its state.

This modeling is based on the generic architecture of control command systems. In order to simulate this model, we have to model its environment and the process to control and we have to consider abstract and perfect sensors and actuators which produce datas and consume commands.

Expressing time constraints is the core of real-time systems. It is necessary to be able to model them at the PIM level. We present here just an example which is the end-to-end deadline. An end-to-end deadline is the maximum value of the time interval between an input event from the process and the moment the corresponding command is sent to the process. The input event occurs at this level when the data is updated. The output is bound to an command that sends a command to the environment. So the constraint is bound to the *NewData()* role of the *Data* role and the *NewCommand()* role of the *Command* role (cf. figure 4). In order to express the semantic of an end-to-end deadline, we use a new stereotype called «EndToEndDeadline» based on the SPT.

```
<<EndToEndDeadline>>:
   Command.NewCommand.end() - Data.NewValue.begin() < ('50', 'ms')
```

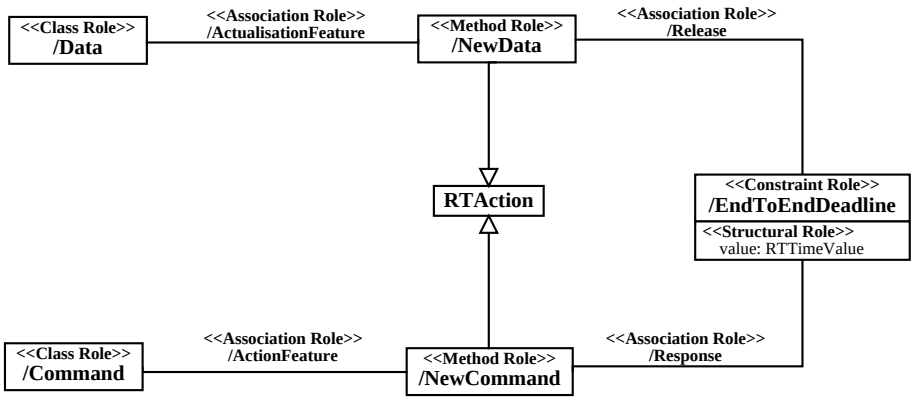

Figure 4: End-to-end deadline modeling

During modeling, one of the encountered problems is taking care of the constraint context. In UML sequence diagrams indicate object interactions according to the uses (scenarios) of the system. They are adapted to express specific constraints of a given scenario.

4.3 PSM

4.3.1 PSM IO

Principles The aim of the PSM IO is to integrate the constraints related to the sensors and the actuators. These constraints are modeled in a PM called Interface PM. The Interface PM is divided in two layers. The first layer is the interface which represents the services of device drivers. It encapsulates the communication between software and hardware (sensors and actuators).

The second layer offers the high level services a PIM may use. These services are platform independent. But, the implementation of this layer is strongly linked to the device driver layer. This layer manages the interactions between the application and the device drivers services and belongs to the PM. In the sensor side, the input of this layer is constituted of the raw data coming from the driver. Then the layer produces the information the classes playing *Data* roles need. In the actuator side, this layer calculates the raw orders that must be given to complete the commands. So this new layer has two functionalities: data processing, communication (i.e. protocol) between the application and the drivers.

The mapping consists on manually associating the classes of this layer with the *Data*/*Command* classes they must be associated with.

In conclusion, the PSM IO is constituted of seven layers : interface with the sensor drivers, sensorInterface/application communication, application datas, application, application commands, actuatorInterface/application communication and interface with the actuator drivers.

Constraints derivation An essential point of this transformation is the mapping of the real-time constraints. The real-time constraints expressed at the PIM level refer to the borders of the system (*Data* and *Command* roles). But the borders have changed at the PSM IO level. So, at this level the constraints must be applied on the methods of the classes which communicates with the real environment. This operation is called constraints derivation.

For example, a *Data* class called *DataDoor* represents the door of a car, modeled with two states: *opened* and *closed*. We consider the opening of the door activates an action which must be executed in a given time (according to an end-to-end deadline). At the PIM level, the time constraint beginning corresponds to the moment the state of the door changes from *opened* to *closed* and is expressed on the *CarDoor* class (`CarDoor.Open.begin()`). At the PSM level, this class is an internal state of a system data. The time constraint has to be moved on the class communicating with the hardware. The class communicating with the hardware effectively detects the true moment of the opening of the door.

This operation may be complex. Especially when several sources in the PM are used to construct a PIM data. In this case, we must find the true temporal source of the constraint in order to know where it must be modeled. We prefer in this case to consider that the first event in their order of arriving is the release event of the constraint.

Optimisation In order to keep encapsulation, flexibility and modularity, the approach introduces a lot of classes and methods in the models. But this leads to complex architectures. For example, the activation of an action caused by the press on a button produces seven method calls where only one call should be enough. Because of the layered architecture, seven objects are necessary to respect it.

In the aimed domain, size constraints are strong. So the proposed architecture has to be simplified. By analogy with some optimisation approaches used on hardware [FGM01], it seemed interesting to optimise the application before it is implemented. Indeed, some optimisations must be done according to the global structure of the application. Moreover, the same optimisation may be common to several implementations.

The aimed applications and the proposed models lead us to propose several kinds of optimisations. A first proposed optimisation is the regrouping of the object playing a *Data* role, its communication interface with the driver and the driver. Another proposed optimisation is related to the observers. The observers acts as event generators. We may regroup the observed data and the observer : the code of the observer is added into the method which updates the data. These optimisations strongly reduce the number of classes and method calls.

4.3.2 PSM

The PSM refines the PSM IO in order to prepare the code generation. It introduces the sequencing of the actions via the use of Operating System tasks. In the methodology, this model is independent of a specific RTOS. This phase is based on a generic PM representing RTOS services. The task model is constituted of : Periodic Tasks, Polling Servers, Aperiodic Tasks and Software Tasks. Each object is implemented with some primitive objects of the RTOS's. Each object can be modeled with the SPT [SPT02] by the using of the appropriate stereotypes.

The mapping from the PSM IO to the PSM consists in associating objects to tasks using Active Objects (AO) [BS00]. An AO is a server that executes object methods: when an AO receives a message, it executes the methods for the concerning objects (calls Passive Objects). Grouping objects can produce several implementations: one AO for each layer, one AO for each event, one AO for each object, ... The mapping method is crucial in the aimed domain. It can lead to a large amount of tasks (one AO/object) or

create inappropriate blocking times according to the real-time constraints (one AO per the whole system). Finally, the constraints modeled in the PSM IO must be derived in the PSM. Activating constraints become activating constraints of tasks. Deadlines and data age constraints must be checked with the appropriate methods (MAST [PHD01]).

The obtained PSM is used to generate the code. The code generator may optimise the model in order to make the code faster and smaller.

We can notice that the PSM meta model does not use roles. Roles were used in the first phase essentially in order to provide architecture constraints, especially the PIM. Afterwards, if the structuring rules have been respected at the PIM level, the models produced at the other levels respect these structuring rules. So, considering the transformations are correct, meta model constraints are not useful.

5 Application

The proposed approach has been tested on several projects (Josefil [JOS], Télémaque). The aim of the Josefil challenge is to realise the control system of an autonomous robot of exploration which carries out missions of temperature measurements. The goal of Josefil [JOS] is to test the MDA approaches in case of specification changes (at PIM and PSM level). The PIM modeling was firstly done in UML. The PIM expressed the applications of the system. The PSM IO was also done in UML and then has been translated into SDL (Specification Description Language [SDL96]). Finally the SDL model has been translated directly into executable code. The generated application has a layered architecture and a big part of the conception can be reused. Refinement, optimisations and mappings have been done by hand because no appropriate tools have been yet developed.

Applying the approach demonstrated that the modeling of the PIM and the Interface PM are crucial. A bad choice at the PIM level can make the system unreusable. To illustrate this we give two examples based on the Josefil challenge. First, the mission could be given either by a file containing a list of points or by several mouse clicks on the screen, i.e. different PM's. If no *Data* (called *Mission*) class exists at the PIM level, it is not possible to reuse the code if the file is replaced by the mouse. Second, a LED must light when the robot goes forward. This *application* given in the requirements must be done by an *Element*, i.e. by the applicative layer. The PM package given by the Josefil Challenge directly codes this *application*. The reason seems that it was easy to code it directly in the PM part. But, the code as in the previous example is not reusable at all because platform concerns and application concerns are mixed.

On the contrary the approach produces a lot of classes. For example, for the speed control of a robot with a joystick, the approach leads us to a model with about twenty classes. Although the behaviour is simple, the number of class is too large. Optimisations are crucial if we want to produce "real" applications in the aimed domain.

Finally it has appeared to us that the approach is complex for a simple user. Meta models and all transformations are not easy to understand. Editing tools must be developed in order to help the user to apply the MDA approach.

6 Conclusion

This paper introduces a methodology for the development of real-time systems dedicated to process control following the MDA approach. Although it is based on meta models, the methodology remains mainly informal due to the lack of tools. It is necessary to provide PIM editors, automatic tools of mapping (at least semi automatic tools), optimisation and code generation. The tools of mapping must be parameterized with PSM generation rules and according to the aimed platform. A work is in progress in order to provide an editor tool.

A complete development tool based on the approach would help the designer to manage all the models of the approach. It would also ensure their interconnection.

Finally the approach must be extended to "soft" real-time systems and to distributed systems.

References

[BS00] J.-P. Babau and J.-L. Sourrouille. "Multi-tasking implementation for object-oriented design". In *Workshop SIVOOES-2000*, Cannes, France, 2000.

[COR] `http://www.corba.org/`.

[CS02] G. Caplat and J.-L. Sourouille. "Model mapping in MDA". In *Workshop WiSME UML'2002*, Dresden, Germany, 2002.

[Dou02] B. P. Douglass. "Real-Time Design Patterns: Robuts Scalable Architecture for Real-Time Systems". *Addison-Wesley*, 2002.

[FGM01] A. Fraboulet, K. Godary, and A. Mignotte. "Loop fusion for memory space optimization". In *IEEE International Symposium on System Synthesis*, Montréal, Canada, Oct 2001. IEEE Press.

[GBNG02] J. Gray, T. Bapty, S. Neema, and A. Gokhale. "Generating aspect code from models". In *OOPSLA Workshop on Generative Techniques for Model-Driven Architecture*, Seattle, WA, Nov 2002.

[GHJV95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. "Design Patterns". *Addison-Wesley*, 1995.

[GNS+02] A. Gokhale, B. Natarjan, D. C. Schmidt, A. Nechypurenko, N. Wang, J. Gray, S. Neema, T. Bapty, and J. Parsons. "CoSMIC: An MDA generative tool for distributed real-time and embedded component middleware and applications". In *Proceedings of the OOPSLA 2002 Workshop on Generative Techniques in the Context of Model Driven Architecture*, Seattle, WA, Nov 2002.

[J2E] `http://www.sun.com/j2ee/`.

[JOS] `http://www.ensieta.fr/mda/JOSEFIL/`.

[KFGS02] D.-K. Kim, R. France, S. Ghosh, and E. Song. "Using role-based modeling language (RBML) as precise characterizations of model families". In *Proceedings of The 8th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 2002)*, Greenbelt, MD, Dec 2002.

[KLM+97] G. Kiczales, J. Lamping, A. Menhdhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin. "Aspect-oriented programming". In *European Conference on Object-Oriented Programming*, Jyvaskyla, Finland, 1997. Springer-Verlag, LNCS 1241.

[MDA03] OMG Architecture Board MDA Drafting Team. "Model Driven Architecture Guide Version 1.0", 2003. `http://www.omg.org/cgi-bin/apps/doc?omg/03-05-01.pdf`.

[PHD01] J.-L. Medina Pasaje, M. Gonzales Harbour, and J.-M. Drake. "MAST real-time view: A graphic uml tool for modeling object-oriented real-time systems". In *IEEE Real-Time Symposium, London*, 2001.

[SDL96] ITU-T. "Recommendation Z.100, Specification and Design Language (SDL)". COM X-R 17-E, Geneva, 1996.

[SPT02] OMG. "UML Profile for Schedulability, Performance and Time Specification", Mar 2002. `http://www.omg.org/cgi-bin/doc?ptc/02-03-02.pdf`.

[TNHN02] A. Tesanovic, D. Nyström, J. Hansson, and C. Norström. "Integrating symbolic worst-case execution time analysis with aspect-oriented system development". In *OOPSLA 2002 Workshop on Tools for Aspect-Oriented Software Development*, Seattle, WA, Nov 2002.

[UML01] OMG. "OMG Unified Modeling Language Specification", Sep 2001. UML 1.4 Version.