Model driven engineering for embedded real-time systems

Y. Tanguy$^1$, S. Gérard$^1$, A. Radermacher$^1$, F. Terrier$^1$

1: CEA-List, CEA Saclay DTSI/SOL/LLSP F91191 GIF-SUR-YVETTE

Abstract: In order to keep competition at bay, industries have no other choice than ever increasing the innovative part in their products at a lower cost and within a shorter time-to-market. Obviously, this tends to make products drastically more and more complex and hard to realize.

The Accord|UML research project, driven for several years by the CEA-List, aims to provide solutions to manage complexity in the development of real-time embedded systems (RTES). Accord|UML consists in a methodology and a set of dedicated tools. The methodology is based on a generative model-centered process that covers the whole RTES development cycle from the very early requirement analysis steps to the generation of an executable prototype.

This paper will analyze how model driven engineering may be valued in a RTES development context. Especially, referring to the Accord|UML project, it will illustrate abstraction and refinement through the presentation of minimal language extensions for real-time concerns, code generation and validation of embedded real-time features.

Keywords: UML, MDE, RTES, Accord|UML, Methodology.

1. Introduction

Due to growing competition among industries, engineers are forced to spend more and more time on their very core business domain to constantly improve the functionalities offered by the products they put on the market. The problem is that such a race for technological innovation tends to generate more complexity, higher costs and longer development cycles for every new generation of products! In parallel, the part of software embedded in these products keeps increasing, making them ever more difficult to design, test, and maintain. Moreover, embedded systems are real-time systems by nature due to their direct relation with their environment or their users. Such embedded real-time software must implement special features as they have to respect specific non-functional constraints and requirements, like responsiveness or periodicity which also tends to make them more complex.

Considering these issues, it appears that new development techniques should be adopted rapidly to relieve engineers of software development concerns allowing them to concentrate their efforts on their initial domain. Actually, when correctly supported by tools, the model paradigm support the use of multiple graphical views dedicated to explicit different aspects of the system, or dedicated to give the reader the most comprehensive abstraction level needed. Associated to efficient code generation tools, a part of the tedious and error prone coding process is relieved from developers giving them more time to spend on the functional part of their products. The model paradigm is hence likely to address these needs and could become quite an efficient way to get complexity under control, improving product quality, and reducing time-to-market delays in the same time. To sum up, an emerging solution may be to ask developers to describe models rather than to write code.

Reliance and efficiency are two inherent requirements of any real-time embedded system (RTES) and this cannot be obtained unless the concerned software is built on top of precise specification and analysis. Model driven engineering (MDE) can be summarized in two major basic principles of software development which are: abstraction and refinement. Considering both aspects, model engineering can perfectly be derived and used in the development of RTES. This paper will discuss issues related to the two following model engineering aspects when mapped and valued in a RTES development context:

- Abstraction – the complete specification of a RTES requires the description of a set of complementary and consistent views dedicated to various aspects of the system like its functional analysis, real-time constraints or fault tolerance behavior. Furthermore, the embedded and real-time nature of such systems leads to specification where generic product features and implementation specific features are tightly stuck together and hard to distinguish. As a result, developers need one (or more) suitable language(s), with a sufficient level of richness, completeness and precision, to manage the various abstraction levels required in RTES development. Among RTES model properties, completeness, determinism and executability
appear as essential features that a model needs to satisfy.

- Refinement – model transformation stands for a large part of the growing interest in model driven engineering as it automates various tedious and error-prone activities during the modeling process. Applied to the RTES development context, model transformation is a perfect means to support refinements going from first specification up to the implementation related modeling steps.

The Model Driven Architecture approach, promoted by the OMG for a few years, puts the model paradigm in the center of the development process. In this approach, the process activities rely on an intensive use of model transformations and tries as much as possible to separate concerns that are platform specific and platform independent. The CEA-List drives experiment and research in the field of RTES development for several years. In this scope, a lot of work has been done to apply the founding concepts of MDA [1] (and model driven oriented approaches in general) in a development process dedicated to real time and embedded systems. This work resulted in the Accord|UML toolkit which consists of a complete UML based methodology and related supporting tools. This methodology covers the whole development cycle of RTES from very early specification activities to a first executable prototype.

The paper will analyze how model driven engineering can provide a significant contribution to RTES development through practical experiments conducted by the CEA-List group in the Accord|UML research project. This paper is divided into three main parts. In the first one we give general considerations regarding MDE approaches applied to real-time and embedded development. Then in a second part MDE modeling aspects are discussed paying a particular attention to the modeling language and the methodology aspects used to build efficient models. The last part of this document is focused on model transformations and how they can be used to improve and ease a RTES modeling process.

2. Adapting MDE for real-time embedded systems

Accord|UML [2-5] is both a modeling framework and a methodology which aims to assist RTES development. One of the main founding ideas behind Accord|UML is the fact that most of the real-time implementation concepts are masked and managed by the framework so that the developers can concentrate on the domain related aspects of the system (functional and non-functional aspects). An expected result is that developers are not asked to be real-time experts. Most of the real-time implementation difficulties are transparently managed by the framework. To achieve this goal, Accord|UML provides a complete model driven engineering development process and makes an extensive use of UML abstraction and viewpoint facilities. Moreover, automated model transformations are provided to facilitate refinements or model completions and to ensure that the developer correctly follows the methodological steps defined in Accord|UML methodology. Providing automated transformations each time it is possible can drastically reduce modeling errors, and alleviate developers from painful modeling tasks.

As described on Figure 1, the method relies on the four well-known software-engineering tasks applied to modeling: analysis, design, prototyping and validation.

![Figure 1 The Accord|UML process](image)

The developer goes from one stage to another using precise model refinements used in a seamless iterative process.

The first modeling stage directly comes from a refactoring of requirements documents written for the system to be modeled. It mainly consists in rewriting existing requirements documents using UML as the modeling language, and following Accord|UML specification rules. At this stage, the system is seen as a black-box, which means that the developer focuses on its basic functions and its relationships with the environment without giving any realization information. The basic functions and external interactions are modeled as high level scenarii with the help of UML sequence and use case diagrams. Accord|UML offers the possibility to add high level non-functional requirements such as expected temporal constraints (deadline, ready time, etc...) in this first modeling stage. The model built during this stage is the preliminary analysis model (PAM).

This initial model is then refined in order to detail the system specification. The developer is supposed to build the static model which describes the system
structure, and the dynamic models which describe the system behavior. For this latter, Accord|UML uses both sequence and state diagrams. Sequence diagrams are used to provide detailed scenarii which directly comes from a refinement of high level scenarii modeled in the previous phase. These diagrams give a detailed representation of inter-object interactions and clarify expected temporal constraints. State diagrams are quite useful to describe object internal behavior. Obviously, the system structure is detailed through UML class diagrams. This second step produces a detailed analysis model (DAM). Such model remains independent from any implementation language and target execution platform.

This model may be enriched to produce a validation model. This kind of model allow temporal or performance constraints validation. This model may also be used as the initial model to generate test cases. Moreover, the detailed analysis model may also be refined and completed with platform specific information (or constraints) to obtain a prototype model. In that case, the model is necessarily mapped to an implementation language and contains the implementation for every real-time concept shadowed in previous models by the Accord|UML framework. Using this prototype model, code can then be generated and compiled against a given execution platform.

3. A model-centered approach

In the context of real-time and embedded systems, industrials are more and more interested in object oriented or component based approaches. Indeed, such approaches are well-known to offer abstraction and complexity management capacities. This allows various actors, with different understanding levels to efficiently work on the same model by using different model viewpoints or abstraction levels. Formerly used for documentation purpose, UML tends to take a growing importance in industry as a complete development tool per se.

UML is now considered as the de facto standard language for modeling. Two complementary UML profiles have been specified and adopted as new OMG standard that address RTES development specific issues: the « scheduling, performance and time » (SPT) profile [6] and the « QoS and fault tolerance» profile [7]. These profiles do not fully meet RTES development needs and they should be considered as a generic basis to construct a specific modeling language to support RTES modeling.

3.1. High-level concepts

Developers need to be able to express real-time features in their models as soon as possible during the development process. To ensure this, it is absolutely necessary to provide expressive high level modeling concepts that are completely independent from any implementation consideration. In this scope, Accord|UML intends to let developers add non-functional features directly inside their own domain specific models. Specific MDE artifacts have been developed inside the Accord|UML toolkit for the description of non-functional features such as real-time, embedded or distributed modeling features.

In Accord|UML, two main concepts are dedicated to express real-time features [8]:

- The « RealTimeObject » stereotype extends UML with the possibility to model concurrent entities and which encapsulates concurrency control and object life-cycle management.
- The « RealTimeFeature » is the other key concept. It provides a solution to express real-time temporal constraints in UML models. This concept directly inherits from the quality of service as defined in the SPT specification document.

3.2. Dedicated models for real-time

RTES modeling not only requires dedicated modeling concepts, but also well-formed models. To be considered as well-formed in a real-time context, models need at least to own the following properties: determinism, completeness and executability.

3.2.1 Determinism

System determinism means that considering a given initial state, the system always reacts to the same given set of stimuli by producing exactly the same behavior. Considering RTES, this definition may be extended saying that the expected behavior is supposed to appear at a precise date. Such property is very important for RTES. Determinism is critical in real-time modeling as such system may:

- Integrate control.
- Run for extended periods.
- Operate autonomously.

Indeed, real-time applications differ from general-purpose applications in the temporal correctness of the results produced. Moreover, temporal correctness often needs to be proved or validated. Such proof requires the use of various behavior analysis techniques like schedulability analysis or deadlock detection analysis which is only possible in a determinist model.

As a generic modeling language the UML specification has a generic semantic description. But UML also offers a set of specialization facilities that allow anyone to adapt UML for his particular needs. Considering the semantic, this is achieved through the notion of semantic variation point. It consists of
specific points of the specification where the UML semantic is voluntarily kept open. In fact any UML modeling tool or methodologies are supposed to clearly and precisely explicit the variation points with their own semantic. Unless semantic variation points are completed, it is not possible to state on the meaning of a UML model, in such a case it would be definitely impossible to build determinist models.

As described on the picture below, an example of variation point is the dispatching policy for messages stored in UML state machine message queue. Accord|UML specifies that the messages are managed with a message box and that the dispatching policy associated to this is an « Earliest Deadline First » (EDF) algorithm.

In other word, a real-time feature is added to each message sent. The message that owns the closest deadline is extracted and treated in priority.

3.2.2 Completeness

Completeness is the next important property required for real-time models. A complete model can be intuitively defined as a model which contains all the necessary information required to represent the modeled system. This concept is obviously hard to be formally and precisely defined. In Accord|UML, a model is considered as complete if it contains every sub-models described in the methodology. For example, at the detailed analysis modeling stage, the complete model is supposed to describe the whole structure, behavior and interaction of the system. In Accord|UML the whole system behavior is here supposed to be completely described in an implementation-independent way. Building models independent from a target platform and implementation language enhances model reusability but requires a dedicated language to model any possible kind of actions.

Since its version 1.5, UML defines the clear and unambiguous abstract syntax of an action modeling language. Accord|UML proposes a concrete syntax [9] compliant with UML action language abstract syntax. This language perfectly fits real-time needs, and due to its compliancy with UML 2.0 abstract syntax, it will also be compliant with any tool based on UML 2.0 specification such as a model execution engine.

3.2.3 Executability

A model is executable when it can be operated on a computer. There is mainly two ways to make a model executable.

The first solution requires a model execution engine, which is a kind of model interpreter that can read a model and directly operates on a computer (or any hardware execution unit).

The other solution consists in defining a transformation from the model to an existing executable platform and formalism. This is usually achieved through a code generation process.

Obviously, in both these two cases, the model to be executed needs to be complete and determinist (unambiguous at least). To achieve the executability property of models, Accord|UML provides a dedicated code generation facility and a real-time execution framework (the Accord kernel). This framework provides an operating support for high level real-time concepts defined in the Accord|UML profiles that extend the UML language. Hence, the Accord kernel gives sense to features defined in Accord|UML such as the “RealTimeObject” concept, by providing execution support to realize communication and message management associated to this kind of objects.

3.3. Consistency and complementarities

To manage the complexity of systems, a widespread software engineering technique is the separation of concerns. It consists in modeling the various aspects in the system separately with different views (structure, behavior …).

One advantage of this technique is that developers can focus on a particular view to solve a problem without being disturbed by the difficulty of managing the whole system. The issue inherently raised by the separation of concerns is the problem of the consistency between views. Indeed, it is important to keep in mind the fact that views are local points of view which all relates to the very same and unique model of the system. As a result, these views are not completely independent one from each other; the same model element often appears in several different views. While working on the different views separately, one can easily introduce inconsistencies or incoherencies in the model. A possible solution consists in defining a set of consistency rules that need to be verified by the whole model to ensure its global consistency. Such rules may be described using the object constraint language [10] (OCL). If the modeling tool offers an OCL expression evaluation service, the consistency rules may even be automatically checked during modeling.
As shown on Figure 3, modeling real-time applications using Accord|UML relies on the previously described concept of separated views grouped as a consistent set. In the detailed specification stage, Accord|UML identifies three partial, complementary and coherent views of the system:

- The **structural view** defines the architecture of the application. This model is mainly based on UML class diagrams, and describes the whole application classes including their property and relationships. Interactions between classes are not treated here, relationships just model the fact that classes need or may need to have communication support.

- The **behavioral view** focuses on the local behavior of the classes defined in the structural view. The goal of this view is to clarify both class and operation behavior. Class behavior relates to control aspects, while operation behavior is the algorithmic description of operations. This point will be further discussed in the paragraph 3.4.

- The **interaction view** concerns the “instance layer” of the application model. It connects the various instances used in the application and describes explicitly how messages are exchanged at runtime between a set of instances to realize the tasks that the application is required to realize.

### 3.4. Modeling the system behavior

The UML state machine is the usual way to express behavior in models. This kind of diagram is added to system classes to describe their internal behavior. A possible way to proceed is to use a single state machine diagram to model both control and algorithmic aspects of class behavior.

However, merging these aspects in the same diagram presents two major drawbacks. Firstly, diagrams rapidly become hard to maintain as it produces very large state machine diagrams even for relatively simple behaviors, and hard to read due to the mix between control and algorithmic aspects. The second drawback is a reusability issue. As the algorithmic aspects are interleaved with control aspects in the state machine diagram transitions instead of method body, most of useful object oriented language properties like inheritance, overriding, or polymorphism are no more usable. They need to be redefined and clarified to take into account state machine specifications. Moreover, any modifications in the algorithm may impact the whole behavior as a single diagram is used.

The separation of concerns applied to behavior modeling is an efficient way to avoid the drawbacks mentioned before. It mainly consists in a clear separation between control related aspect and algorithms. This modeling principle has been promoted for a long time by synchronous approaches in the context of reactive systems development. This principle is also promoted as a major modeling concept in Accord|UML (see Figure 4).

The control aspect of class behavior refers to the class logic control also known as the object lifecycle. This part of the modeling process uses a specialization of UML standard state machines specification: the protocol state machines. Protocol state machines have two main particularities:

- Actions are attached to transition (no state actions).
- Transitions can only trigger class method, not an action sequence.

Indeed, UML state machine can be either Moore or Mealey automaton, which means that actions can be respectively associated either with states or transitions. For readability reasons it is preferable to strictly restrict the use of state diagram either to Moore or Mealy style. In Accord|UML, state machines are supposed to be Mealy automaton. Using state actions in UML state machine may be tedious as the semantic is not intuitive and may even lead to concurrency issues in a real-time context.
Transitions in a protocol state machine always refer to an operation which is defined in the context of the classifier that owns the transition. Such transition never triggers any actions or any set of actions, but only the execution of the method that implements the operation it refers to. With this kind of state machine, the user can describe which class methods are available for execution at a given instant and within a given state.

To describe how a method implements a given operation, Accord|UML proposes to use activity diagrams. These diagrams are quite well adapted to describe algorithms. It consists of action sequence and classical algorithmic control nodes (if … then … else …, loop …). Accord|UML action language [9] is used to define actions in this kind of diagrams. An open source editor for this action language will soon be realized and made available for download on the Eclipse MDDi project website.

Figure 5 Protocol state machine transitions

4. Refinement with model transformations

Model transformations hold an important place in any MDE process. This part gives examples in which transformations can greatly improve and speed up developer's work.

4.1. Modeling assistance

An obvious possible use of model transformation is to provide modeling assistance to developers. The potential objectives are twofold.

First, transformation may be used in a given methodological context to ensure that the developer strictly follows a particular development process. For example, the user may be forced to use transformations to go from one modeling step to another. This requires that a methodology expert has prepared every model transformation used by developers in their development process.

The other possibility is to use (semi-) automated transformations to manage error prone or tedious modifications in models. For example, a simple naming correction in the model may impact a large set of elements to ensure traceability consistency. This is typically a case where models can be automatically refactored. Another example is the application of patterns in the model.

Figure 6 Signal implementation pattern

Accord|UML provides such modeling assistance:

- Between the main modeling stages (cf. Figure 1).
- To check model consistency, and in particular to ensure that state machines are valid and consistent with the system structure.
- To refine and add implementation patterns to each real-time concepts introduced by Accord|UML.

The Figure 6 presents such implementation pattern applied to a signal defined in the model.

4.2. Validation

In addition to modeling assistance Accord|UML provides bridges between UML and other formalisms. Such links are provided between UML and existing tools issued from formal technologies for validation purpose. Two experiments have been realized in this field:

- Automatic test case generation.
- Schedulability analysis.

In both cases, model transformation rules were defined to transform Accord|UML models in a new formalism provided as input to the behavioral analysis tool: AGATHA [11]. The formal technological space is well-known to provide good validation tools. The problem is that such tools in general suffer from their inherent complexity that requires developers to be mathematics experts. The goal here is to hide such complexity with the help of UML models and related transformations. Formal tools are used and their results are retrieved and displayed in an appropriate form directly within the initial UML modeling tool.

4.3. Application synthesis

Real-time and embedded systems also differ from general-purpose systems because of the
It is possible to add platform dependant optimization by binary shifts operation. "divide/multiply by 2" operation that can be replaced equivalent instructions (a well-known example is consists in replacing some instructions by faster generated code. Another possible optimization generation strategy consists in reducing calculations. To manage real-time implementation constraints, the code generation process (C/C++ implementation).

In Accord/UML, the application is produced by a code generation process (C/C++ implementation). To manage real-time implementation constraints, the code generation can be configured to give priority to memory consumption, or other criteria. For example, to produce a code optimized to use less memory and minimized energy consumption, a code generation strategy consists in reducing calculations. Redundant and useless calculations can be removed or replaced by constant expressions in the generated code. Another possible optimization consists in replacing some instructions by faster equivalent instructions (a well-known example is "divide/multiply by 2" operation that can be replaced by binary shifts operation).

It is possible to add platform dependant optimization rules to the general strategies previously mentioned. The same way model transformations were used to reach formal validation tools; existing hardware/software optimization tools may be used to optimize the application for a given platform. This has been successfully experimented in the ACOTRIS project. ACOTRIS [12] was an RNTL 2000 labeled project which goal was to bridge the gap between a UML modeling process for real-time embedded applications and an implementation realized with a synchronous implementation language SIGNAL [13] and a synchronous-based optimization tool SynDEx [14].

![Figure 7 Overall view of ACOTRIS transformations](image)

The picture above gives a simple overall description of the code generation transformation realized in the ACOTRIS project to reach the synchronous approaches technological space from Accord/UML models [15-17]. As formal approaches, synchronous languages and tools are recognized for their optimization and validation capacities but they require a high level expertise from developers to be used efficiently. Using the abstraction facilities offered by UML models to masked synchronous implementation specificities, ACOTRIS results potentially bring such tools to a much wider audience. As a result, once a synchronous code is generated, the SynDEx tool may be used to provide an optimized placement of the software components on each available execution unit.

5. Conclusions

The experiments conducted by the CEA-List around the Accord/UML methodology and tools shows that MDE artifacts can cover needs of real-time embedded system development:

- **RTES development requires the ability to work on several views describing the same model.** MDE language and UML in particular offers such facilities of local point of view description. Moreover, MDE artifacts may be built to ensure the coherence, consistency, and more generally well-formed modeling rules that are supposed to be verified. Such rules are crucial to be able to work independently with multiple views without risk to enforce the well-formed rules defined in the methodology, and introduce inconsistency issues in the model.

- **The targeted execution platform may differ.** In a MDE process, a large part of the models may be described independently from any final implementation consideration (language or target platform). The platform itself may even be modeled and directly linked to the application developed. As a result, industrials may gain a better control on the application maintenance and reusability. It is now possible to capitalize not only on the source code which is firmly linked to an implementation language and an execution platform but directly on the MDE artifacts and models from which the code is generated.

- **RTES performances should be managed.** Heuristics and code generation techniques can perfectly handle this point. With efficient code generation, it is possible to construct high level models and, at the same time, to produce a code that is very close to the best hand tailored code.

- **Test and validation are critical activities in RTES development.** As MDE artifacts, such as model transformations, can bridge the gap between UML and formal technological spaces, existing and efficient formal tools are available to test and validate UML models.

The definition of a large set of model driven engineering artifacts dedicated to development, reusability, validation and maintenance of real-time embedded applications is one of the main concerns of CARROLL common research program settled by...
6. Glossary

- **UML** Unified Modeling Language
- **MDE** Model Driven Engineering
- **RTES** Real-Time Embedded System
- **PAM** Preliminary Analysis Model
- **DAM** Detailed Analysis Model
- **EDF** Earliest Deadline First
- **OCL** Object Constraint Language
- **MDA** Model Driven Architecture
- **PIM** Platform Independent Model
- **PSM** Platform Specific Model
- **QoS** Quality of Service
- **SPT** Scheduling Performance and Time
- **MDDi** Model Driven Development integration

7. References