MDE BENEFITS FOR DISTRIBUTED, REAL TIME AND EMBEDDED SYSTEMS

François Terrier and Sébastien Gérard
CEA-List, {francois.terrier, sebastien.gerard}@cea.f

Abstract: Embedded systems envelopment are currently being challenged to provide global solutions that reconcile three conflicting agendas: enrichment/refinement of system functionalities, reduction of time-to-market and production costs, and compliance with nonfunctional requirements. Model-Driven Engineering (MDE) can help development master these complexities by both separating concerns and systematically automating the production, integration and validation process. This paper draws on the Accord|$_{UML}$ research project to illustrate the benefits of model-driven engineering for embedded real time systems development.

Keywords: Model-Driven Engineering, UML profile, Embedded System, Real Time

1. INTRODUCTION

On today's sharply competitive industrial market, engineers must focus on their core competencies to produce ever more innovative products, while also reducing development times and costs. This has further heightened the complexity of the development process. At the same time, industrial systems, and specifically embedded systems, have become increasingly software-intensive. New software development approaches and methods must therefore be found to free engineers from the technical constraints of development and allow them to concentrate on their core specialties. One possible solution is to provide them with development models adapted to these specialties instead of asking them to write code.

The purpose of this paper is to describe the advantages expected from MDE for embedded real time systems. The paper begins with an overview

of development problems, and then describes the various stages in MDE and their advantages, using the example of a CEA-List experiment conducted for the Accord|$_{UML}$ research project.

2. MODEL MANIPULATION

Model-driven engineering relies on mastery of two essential mechanisms: abstraction and refinement. Both mechanisms partially satisfy the need for top-down and bottom-up approaches, while providing the different points of view required for system development (e.g. design and validation).

2.1 MODEL ABSTRACTION

The concept of "abstraction" is intrinsic to that of modeling, which by definition consists of representing a real world object in simplified form. This involves two possible types of abstraction – vertical and horizontal.

- *Vertical abstraction* – it follows development process flow, producing models that focus on the pertinent level of detail. There is a recurrent need, in system development, for models of standardized software (RTOS and/or middleware) and of hardware implementation platforms (e.g. POSIX, OSEK) that identify dependencies between application models and implementation choices/constraints.
- *Horizontal abstraction* – it takes place at a same level of definition and emphasizes certain system facets or complementary viewpoints. Examples include task models for RT analysis, architectural models centering on system functions and scenarios models for system testing.

For refinement purposes, the goal is to master and automate the process of building one specialized model from another. This typically means producing executable applications for example by model compilation, formalization, use of design patterns for the domain.

2.2 EXECUTABLE MODELS

Fast prototyping, evaluation and validation are vital to the development of real time embedded (RT/E) systems. To this effect, engineers need "executable" models, i.e. models whose behavior (dynamics) can be executed or simulated. More specifically, they must also be:

- *Deterministic behavior models* – RT/E applications must always behave in the same way in a given, same initial context. The semantics of the models used, and therefore, of the underlying modeling language, must be precisely defined and above all unambiguous.

- *Complete models* – A complete model contains all the data required to analyze its behavior and to generate an executable view of this data.

A model is executable if it exhibits both the above properties: *determinism and completeness*. It can be executed in different ways, e.g. via automatic code generation and simulation of the resulting code, or through formal model analysis tools that trace system operation and verify behavior.

2.3 UML FORMALISM

This section provides a brief description of the UML standard from an RT/E perspective. It then identifies the reasons why extensions to this standard, in the form of language-dedicated MDE artifacts, are necessary. Following this quick overview, the high level abstractions required to model RT concerns, i.e. concurrency and RT constraints, are presented.

2.3.1 UML PROFILE FOR EMBEDDED SYSTEMS

The Object Management Group has standard profiles (e.g. for CORBA), to model "schedulability, performance and time" (SPT [1]) and "quality of service and fault tolerance" (QoS&FT [2]). However, these profiles cannot fully support the needs of the real time domain. OMG has therefore launched a new profile, definition for this domain (MARTE, [3]), which includes the previous SPT profile and affords:

- generic concepts required to model real time aspects in both qualitative and quantitative terms (concurrency, resources, time);
- concepts required for schedulability or performance analysis on a model;
- a complete set of modeling elements to build specification and design models of embedded systems, and support the various (asynchronous and synchronous) computation models used in the RT domain;
- extensive models of standard platforms (POSIX, Arinc, OSEK, etc.).

2.3.2 OPEN POINTS

The UML2 standard deliberately leaves open a number of issues for which solutions differ, depending on domain and software design approach. These points need to be clarified to obtain complete and unambiguous models [4]. The three most important of them are:

- *Concurrency models* – the basic concept of UML is the "active object", whose only given characteristic is the ability to autonomously process certain messages, thereby introducing parallel execution.
- *Message management policy for state machines* – the basic management scheme calls for messages to be placed in a queue, then selected one at a

time to trigger a transition and undergo processing. The next message is only taken once the first has been processed. Neither the message selection criterion nor the exact system behavior on receipt of an unexpected message is defined here.

- *Description of algorithms* – UML2 proposes a conceptual framework for describing all actions and their sequences. It does not, however, define a standardized notation that is common to and directly usable for them.

Practical implementation of UML for embedded systems also requires well-defined semantic variation points and a modeling method adapted to each development stage, to ensure selection of the right modeling level, usable concept subset and writing rules.

3. BACKGROUND ON MDE RESEARCH

The CEA-List conducts studies on new methods and techniques to facilitate development of distributed real-time embedded systems (or DREs). Through regular collaboration with major industrial firms (PSA, Thales, EdF, EADS, CS-SI, etc.). Among the needs expressed there is a strong demand for assistance in dealing with the increasing complexity of systems, a wide variety of implementation options, constantly evolving functionalities and shorter times-to-market. Strong attention is also being paid to "capturing" and generalizing company development knowhow to afford and enhance reusability, not only in the last stages of implementation but also throughout the system development cycle. DREs are therefore an ideal case for studying possible automation of development steps or model analysis:

1. They intrinsically involve various points of view (e.g. functional, real-time, security, fault-tolerance), with the drawback that separation of product line generic requirements from those of particular applications is difficult (due to *a priori* implementation choices or constraints or frequent expression of requirements based on specific real time values).
2. Target implementation options vary widely: they may use different execution models for a same specification, (e.g. multitask models with RTOS, synchronous models, loop programming); they may be mapped on various platforms (single/multiple processors, shared/distributed memory…). In addition, these options use largely proprietary and ad hoc solutions and benefit only from a few viable standards.
3. Performance is often a "sensitive" issue, which cannot be dealt with practically by conventional software encapsulation techniques (multi-level interfaces may be inefficient). This can lead to strong interleaving of real time and functional codes, a critical factor where component

models for DREs must be defined without knowing the intricacies of real time and functional code operation inside the component.

4. They are often critical to testing or validation and require complete and accurate specifications and intensive use of system analysis techniques.
5. They generally require very skilled developers (specification, design, implementation, validation, integration), with a high level of expertise.

All of these requirements are an incentive for intensive use of MDE techniques throughout the system development and validation process. This means use of complementary MDE artifacts tested, implemented and evaluated in the Accord$_{|UML}$ methodology tool kit as discussed below.

4. PREVIOUS EXPERIENCE WITH MDE

Accord$_{|UML}$ [5-13] is both a conceptual framework and a method whose purpose is to assist in developing RTE applications, whereby specific design and implementation aspects are abstracted as much as possible, so that:

- Developers can concentrate on the specialist aspects of these systems (e.g. functionalities and performance constraints).
- Porting of applications is made easier by use of a first modeling level that is independent from implementation platforms.

To achieve this, Accord$_{|UML}$ is adapted to an MDE context. Accord$_{|UML}$ is based on use of UML models for abstraction and automatic and/or assisted transformation for refinement. Another of its objectives is to provide the method with modeling guidance tools. This method involves four main modeling phases: preliminary analysis, detailed analysis, validation/testing and prototyping. Progression from one to another of these phases is achieved by a continuous and iterative process of refining UML models.

The preliminary analysis phase thus consists essentially of rewriting system specifications in unambiguous form [8, 11]. Requirements are translated into use cases that are themselves broken down into scenarios depicted by sequence diagrams; these scenarios model interactions between the system, considered as a black box, and its environment. This requires very few concepts: the modeling rules (e.g. choice of diagrams and consistencies) are formalized and incorporated into the development process model as an initial profile, identified as the "Preliminary Analysis Rules".

The second phase consists of switching from a black box to a white box view in which emphasis is then placed on system content. The initial version of the resulting detailed analysis model is obtained by refining the preliminary analysis model. This model affords precise modeling of system behavior using interaction diagrams and state diagrams [6, 7, 9]. It can be

transformed into a dedicated RT validation model and annotated to perform either performance analysis or scheduling analysis [14].

The prototyping model is then obtained on the same basis, by automatic generation from the previous model. This operation is performed specifically through systematic application of design patterns to key elements of the embedded system model (real time objects, automata management, real time constraints, asynchronous or synchronous communications, etc.). The result is a multitask model that can be implemented on the Accord platform [11, 12]. This platform is a "framework" for implementing the high level concepts (such as "real time object") that are manipulated via the application models. It enables complete separation of the specialty models from the implementation constraints of the target platforms.

To assist users in this modeling process, the methodology is backed up by a set of Eclipse modules (modeling and model transformation profiles), which were integrated for experimentation purposes into the IBM-Rationale modeling tool "Rationale Software Architect" (RSA).

4.1 HIGH-LEVEL MODELING CONCEPTS

To facilitate the expression of RT requirements from the onset of the embedded system modeling process, modeling concepts must be suitably detailed and independent from implementation techniques. Two types of abstractions are especially vital to RT/E modeling: parallelism and RT characteristics. To integrate these abstractions, Accord$|_{UML}$ proposes [11]:

- The "RealTimeObject" stereotype: Model elements that have this stereotype are in fact considered as concurrent entities that encapsulate concurrency and state controls for the messages received for processing. This is an extension of the UML *active object* concept. Such an approach views the real time object as a task server whose RT characteristics (deadline, periodicity, etc.) are defined by the received messages triggering the tasks. The operation calls received by the object are processed by a concurrent activity.

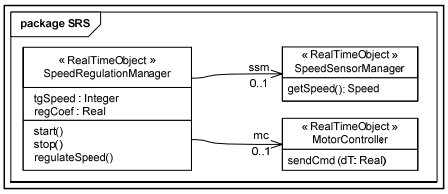

Figure 1. Typical use of RealTimeObject.

- The "RealTimeFeature" stereotype reifies the quality of service concept for modeling the RT constraints of an application. The different tag

definitions associated with this concept allow modeling of qualitative RT characteristics such as deadlines, periods, or ready times. Figure 2 describes the behavior of the *SpeedRegulationManager* class, using an executable-protocol-type state machine (standard specialization of a common state machine). Addition of the "RTF" stereotype with the tagged value *period = 100 Hz* on the cyclic transition of the *On* state with itself causes this cyclic treatment to take place periodically.

Figure 3 illustrates a *startRegulating* interaction**.** It has the «RTF» stereotype that specifies an end-to-end deadline for said scenario, which here is *100 ms*. This means that all the activities executed in the system on receipt of the *startRegulating* message must end no later than 100 ms after the message has been received.

All of the UML extensions proposed by Accord$_{|UML}$ are thus grouped together in a language-dedicated MDE artifact – the Accord$_{|UML}$ profile.

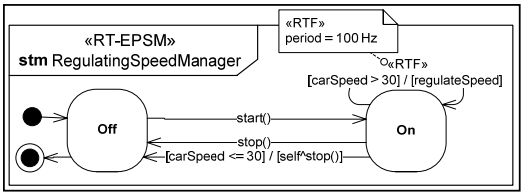

Figure 2. Periodic RTF applied to a state machine.

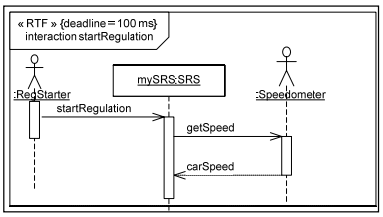

Figure 3. End-to-end Deadline.

4.2 EXECUTABLE MODELS

4.2.1 DETERMINISTIC MODELS

To enable construction of a deterministic model, the underlying modeling language must have both well-defined grammar (syntax) and unambiguous, deterministic semantics. The following paragraphs use three examples based on the Accord$_{|UML}$ approach to illustrate how the indeterminisms remaining in the standard are corrected by specializing the latter in a dedicated profile.

One open variation point in UML semantics is the policy set for dispatching the messages present in a state machine queue. In Accord$_{|UML}$, this variability has been eliminated, by assigning each message an RT constraint, i.e. by considering that each state machine stores received events in a mailbox and that messages are selected by comparing their RT constraints (e.g. the first message selected is the one with the earliest deadline, which is tantamount to "earliest deadline first" scheduling).

4.2.2 COMPLETE MODELS

Once model determinism has been confirmed, the second requirement for RT/E-dedicated models – completeness – must likewise be met. In our case, a complete model not only describes system structure and communications but also fully models behavior, control mechanisms and data processing actions. Our approach proposes to separate control (life cycle) aspects from data processing aspects. Control mechanisms are modeled using state machines as described above. Data processing actions are modeled using UML activity diagrams supplemented by various basic actions. Mathematical actions are modeled using MathML language syntax [15].

To meet the needs of all users, Accord|$_{AL}$ – the action language associated with Accord|$_{UML}$, proposes two formalisms to describe data processing actions, textually or graphically [13] (Figure 4). They are equivalent, and the user can switch back and forth between them according to his needs or working habits. Each action is defined as follows: semantics, textual notation (in EBNF), graphic equivalent and examples.

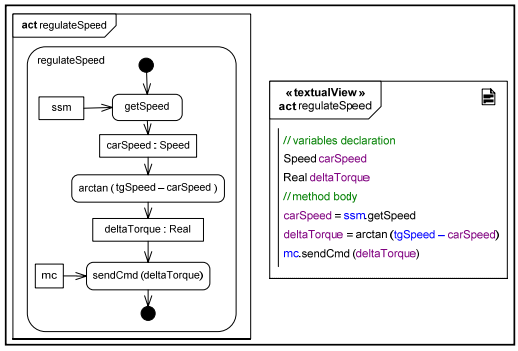

Figure 4. Graphic and textual views of algorithm modeling.

4.3 MODEL EXECUTION

An executable model is a model that can be ran on a computer. There are two main techniques for making a model "executable":

- By a machine that incorporates model elements and can run them on a computer (this is known as "model interpretation") [14].
- By transformation, i.e. conversion of the model into a formalism that is itself executable (i.e. code generation).

5. HOW COULD MDA HELP DREs?

Five requirements have been defined for DRES development. Experience with Accord|$_{UML}$ shows that use of model-driven engineering techniques and development more than cover these requirements, as shown for each here:

- *DREs intrinsically require various points of view*: process definitions and tooling associated with UML extensions that provide high-level concepts can formalize the content of the models as well as their interdependency. It is then possible, on a given model, to ensure visibility of the requirements corresponding to implementation concerns and to easily extract or modify them without changing the rest of the model.
- *DREs implementation choices vary widely*: definition of Computation Description Models and Platform Description Models allow separation of these elements from the rest of the application model. Final implementation can then be done though dedicated transformations.
- *Performance is often a "sensitive" issue*: Code generation heuristics have shown that it is possible to both manipulate high-level concepts in the model and ensure effective implementation.
- *DREs are often critical to testing or validation*: Definition of UML extensions eliminates ambiguities and provides full semantics, thus enabling, by example, use of formal analysis tools to derive test sequences or determine the feasibility of scheduling.
- *DREs generally require highly skilled developers*: generic implementation patterns and architectures are widely used for such systems. Introduced into the development process as reusable elements, they allow easy reuse of developer know-how.

It should be noted that transition from code-oriented to model-oriented development will not alleviate the need for answers to the usual problems of traceability, configuration and version management, etc. For MDE to be successful in industry, solutions to these issues will have to be available for all tools claiming to be MDE-compliant!

Definition and development of a set of MDE artifacts lies at the core of new large-scale joint research programs as for example:

- The *CARROLL* program involving CEA, INRIA and Thales (www.carroll-research.org), whose goal is to provide tools for model-driven engineering and component-based middleware.
- The *Software Factory* project of the System@tic competitiveness cluster (www.systematic-paris-region.org) federates more than 40 partners in research on MDE, validation and execution. This project will provide an open source platform supporting MDE for embedded systems.

REFERENCES

[1] OMG, "UML Profile for Schedulability, Performance, and Time, v1.1," formal/05-01-02, 2005.

[2] OMG, "UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics & Mechanisms," ptc/04-09-01, 2004.

[3] OMG, "UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE) RFP", realtime/05-02-06.

[4] S. Gérard and F. Terrier, "UML for Real-Time," chapter in "UML for Real: Design of Embedded Real-Time Systems," L. Lavagno, G. Martin, and B. Selic, editors, Kluwer Academic Publishers, Boston, 2003, p. 369.

[5] F. Terrier and S. Gérard, "Real Time System Modeling with UML: Current Status and Some Prospects," in 2nd Workshop on SDL and MSC, 2000, Grenoble, France.

[6] A. Lanusse, S. Gérard, and F. Terrier, "Real-Time Modeling with UML : The ACCORD Approach," in "UML'98 : Beyond the Notation," 1998, Mulhouse, France, J. Bezivin et P.A. Muller eds.

[7] S. Gérard, N. S. Voros, C. Koulamas, and F. Terrier, "Efficient System Modeling of Complex Real-time Industrial Networks Using The *ACCORD/UML* Methodology," presented at Architecture and Design of Distributed Embedded Systems (DIPES 2000), B. Kleinjohann, Kluwer Academic Publishers, p. 10, Paderborn University, Germany, October 18-19 2000.

[8] S. Gérard, F. Terrier, and Y. Tanguy, "Using the Model Paradigm for Real-Time Systems Development: ACCORD/UML," presented at OOIS'02-MDSD, J.-M. Bruel and Z. Bellahsène eds., Springer, pp 260-269, Montpellier, September 2002.

[9] C. Mraidha, S. Gérard, F. Terrier, and J. Benzakki, "A Two-Aspect Approach for a Clearer Behavior Model," presented at the 6th IEEE International Symposium on Object-Oriented Real-time Distributed Computing (ISORC'2003), T. N. P. Puschner, A. Ghafoor eds., IEEE Computer Society, ISBN 0-7695-1928-8, pp 213-220, Hakodate, Hokkaido, Japan, 14-16 May 2003.

[10] P. Tessier, S. Gérard, C. Mraidha, F. Terrier, and J.-M. Geib, "A Component-Based Methodology for Embedded System Prototyping," presented at 14th IEEE International Workshop on Rapid System Prototyping (RSP'03), IEEE Computer Society, ISBN 0-7695-1943-1, pp 9-15, San Diego, USA, 9-11 June 2003.

[11] S. Gérard, C. Mraidha, F. Terrier, and B. Baudry, "A UML-Based Concept for High Concurrency: the Real-Time Object," presented at The 7th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'2004), T. A. a. I. L. J. Gustafsson, IEEE Computer Society, ISBN 0-7695-2124-X, pp 64-67, Vienna, Austria, 12-14 May 2004.

[12] N. Guelfi, A. Schoos, S. Gérard, and F. Terrier, "EUDEMES: Component-Based Development Methods for Small-Size Embedded Systems," ERCIM NEWS, 2003, vol. 52 (Embedded Systems), p. 64

[13] C. Mraidha, S. Gérard, Y. Tanguy, H. Dubois, and R. Schneckenburger, "Action Language Notation for Accord/UML," CEA DTSI/SOL/LLSP/04-163/HD, 2004.

[14] D. Lugato, N. Rapin, and J.-P. Gallois, "Verification and tests generation for SDL industrial specifications with the AGATHA," presented at Workshop on Real-Time Tools, CONCUR'01, pp, 2001.

[15] W3C, "Mathematical Markup Language (MathML) Version 2.0 (Second Edition)," http://www.w3.org/TR/2003/REC-MathML2-20031021/, 2003.