MDA PLATFORM FOR COMPLEX EMBEDDED SYSTEMS DEVELOPMENT

Chokri Mraidha, Sylvain Robert, Sébastien Gérard, David Servat
CEA LIST – CEA SACLAY
F-91191 Gif-sur-Yvette Cedex France
Phone : +33 169 085 039
{chokri.mraidha; sylvain.robert; sebastien.gerard ; david.servat}@cea.fr

Abstract: Moving from code-centric to model-centric development seems to be a promising way to cope with the increasing complexity of embedded real-time systems. The Object Management Group (OMG) has been recently promoting this approach, known as Model Driven Architecture (MDA). It relies on UML model refinement and transformation as the basic step of an iterative design process. This model-centric posture has raised many questions, among which the need for an integrated MDA-based developing environment is probably the most severe one. It directly affects the reality of the adoption of this good practice by software engineers. For several years, the CEA-LIST has been involved in the field of real-time systems research and development. This work resulted in the completion of the Accord/UML toolkit, which aims at providing users with a model-driven method and supporting tools. This paper outlines the Accord/UML approach focusing on the solving of complex real-time/embedded systems development issues in this MDA process.

Keywords: Model driven development, UML, Real-time embedded systems

1. INTRODUCTION

Over the last few years, engineers have been faced with the problem of developing more and more complex embedded real-time systems in a world where time-to-market constraints are constantly increasing. Moving from code-centric to model-centric development brings significant answers to software complexity management. With its standardization the Unified Modeling Language (UML) [1] has become the lingua franca of object-oriented modeling. Existing UML-based approaches for real-time systems development [2, 3] still result in models that are hard to maintain and reuse. This drawback is principally due to the lack of model development methodologies.

The Model Driven Architecture (MDA) [4] initiative introduces architectural separation of concerns in order to provide portability, interoperability, maintainability and reusability of models. To achieve these goals, MDA recommends different kinds of models and describes ways to obtain these models from one another through model transformation processes. MDA relies on three kinds of models, which are the Computation Independent Model (CIM), the Platform Independent Model (PIM) and the Platform Specific Model (PSM).

The CIM is a view of a system from a computation independent viewpoint. This model focuses on the requirements of the system and its interactions with the environment while hiding the details of the structure of the system. In other words, the system is seen as a black box. The PIM focuses on the structure and operations of the system from a platform independent viewpoint while hiding details specific to a particular platform. In the PIM, the system is seen as a white box. The PSM combines the PIM with details specific to a particular platform to obtain a model dependent of that platform.

The idea is then to apply MDA tenets in order to facilitate development of real-time applications. Accord/UML [5, 6] is an MDA-oriented methodology entirely based on UML which aims at facilitating real-time software development by engineers who are not real-time experts. The first section of this paper gives an overview of the Accord/UML methodology, enhancing its compliance with the MDA approach. The second section accounts for several directions of research to deal with platform specificities issues for complex embedded real-time systems development, while putting emphasis on code generation process, before giving a short conclusion.

2. OUTLINES OF THE ACCORD/UML PLATFORM

Accord/UML aims at providing users with an MDA-compliant methodology and connected tools dedicated to real-time systems design. This section briefly introduces of the Accord/UML methodology before giving an overview of its associated workbench.

2.1 The Accord/UML methodology

A prototype development with the Accord/UML methodology basically consists of three successive phases, each producing one of the three MDA model kinds. For each phase, Accord/UML provides guidelines and UML extensions (gathered in a UML profile), which enable users to model system real-time features. Moving from one phase to another is facilitated by partially automating model transformations.

The preliminary analysis phase deals with requirements capture. System requirements are identified and reformatted in a set of UML diagrams (use case diagrams and high-level scenario diagrams). The resulting model gives a better-formalized view of system functionalities regardless of its internal structure. This model, called Preliminary Analysis Model (PAM) in our methodology stands for the CIM MDA model.

In the detailed analysis phase, the objective is to move from the PAM to the Detailed Analysis Model (DAM), which is the Accord/UML vision of PIM. The system is decomposed in complementary and consistent sub models: structural models (mainly class diagrams), detailed interaction models (detailed scenarios diagrams), and behavioral model (statecharts and activity diagrams). Structural models are built following a generic pattern, which consists in separating system core features from its relationships with its environment. This approach notably favors reusability and permits to define a generic mapping from PAM to DAM. Modeling real-time structural features is eased by introducing the Real-Time Object concept [7, 8], an extension of UML active objects. As far as behavioral modeling is concerned, two aspects are separated [9]: the control view through statecharts, and the algorithmic view through activity diagrams completed by an UML Action Semantics [1] compliant Action Language [10]. To ensure determinism in modeling behavioral aspects, Accord/UML also provides through its profile a set of rules to specify UML semantics variation points in the one hand and clarify some ambiguous points on the other hand. The resulting model gives an *implementation language independent* executable specification of the system.

Finally, the aim of the prototyping phase is to obtain a complete running mock-up of the application from its DAM [11]. This model is the Prototyping Model (PrM), an Accord/UML equivalent of PSM. This model is then used as an input to a specialized C++ generator, handling notably system real-time features implementation. Eventually, a runtime framework is provided to support the execution of the synthesized code on top of a Real-Time Operating System: the Accord real-time kernel, and the Accord virtual machine. The so-obtained prototype can thus be validated by test.

2.2 The Accord/UML workbench

As depicted in Figure 1, the Accord/UML methodology support consists mainly of three parts: automatic synthesis of specific design patterns relating to real-time and distribution issues; full code generation (structure + behavior) toward the Accord runtime platform; the Accord platform itself implementing high level concepts of the methodology and running on Unix, Linux or VxWorks.

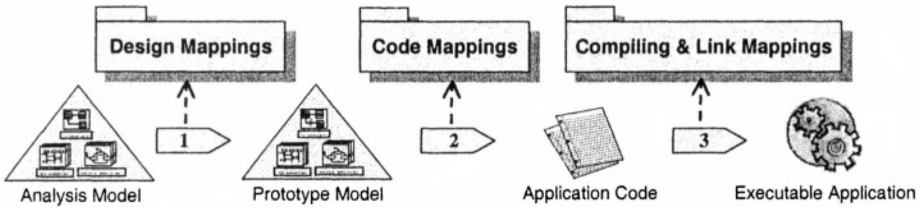

Figure 1: From analysis model to executable application.

The Accord/UML workbench relies on a generic UML-based CASE tool, Objecteering [12], which we customize for distributed real-time embedded systems design. This offers possibilities through its profile builder tool to implement UML profiles. Our toolset is then made of the Objecteering tool completed with additional modules implementing the Accord/UML profiles.

More precisely, in modeling phases, building models is done thanks to the Objecteering UML modeler, which provides a complete set of UML elements (e.g. use cases diagrams, class diagrams, state-machine diagrams, sequence diagrams...), but also using additional model elements defined in the context of the Accord/UML profiles and ensuring real-time features modeling. Stepping from one model to another is done as much as possible via Accord/UML specific model transformation rules. For instance, Accord/UML sets mapping rules to ensure automatic model transformation from Use Cases diagrams to Classes diagrams. The tenets of the approach being to define and implement as often as possible modeling rules to assist the engineer in building the application model. One could speak of MAC ("Modeling Assisted by Computer").

In addition, the Accord/UML tools provide the developer with means of validation in the earlier phases of the development. Firstly, structural and functional validation is carried out on the behavioral models. To this end, a connection has been made between Accord/UML and the Agatha tool [13-15], enabling automatic test case generation from the behavioral diagrams obtained during the detailed analysis phase. Secondly, a validation of quality of service (QoS) in terms of timing requirements is performed by a schedulability analysis of UML models [16]. This point will be discussed in more details in the following sections.

Once application models are completed, one may perform code generation from this model. To this purpose, Accord/UML provides a specialized code generator targeting C++ code (a C code generator being under development). This generator has been upgraded to integrate real-time features support conformant to the Accord/UML specification. This means that the generated code can be executed with support from the Accord kernel and Accord virtual machine [17] running on top of various operating systems, namely VxWorks, UNIX, or Linux.

3. PLATFORM DEPENDANCE ISSUES IN AN MDA PROCESS

In this section, we present the strategy adopted in Accord/UML to deal with real-time/embedded issues, before providing several examples assessing the relevance of our choices.

3.1 Rationale

Targeting real-time embedded applications instead of mainstream ones imposes a superset of constraints on the software developed, among which platform-related considerations and real-time features validation are prevailing. Actually, traditional real-time software design processes provide strategies and support tools to validate temporal (either application-specific or non-functional) properties of the system during the earlier phases of the design cycle. Moreover, in the context of embedded applications, the characteristics of the HW platform have a major influence on the final system temporal behavior and have to be taken into account to make relevant design choices. Integrating these issues in our MDA-compliant approach is thus one of the major challenges we had to face.

As a consequence, two principal objectives were aimed to in the design of our development methodology and tools: to provide a sufficient level of real-time features integration and providing ways to validate the application with respect to the HW platform, in a UML-based model-centric approach. This comes actually to an attempt to merge conflicting aspects, since the ultimate goal of the MDA approach is precisely to shield concerns linked to the platform (in terms of implementation language as well as targeted HW). Furthermore, the UML is a language, which natively provides only “raw” materials (model elements, extension mechanisms), which are voluntarily platform-independent and generalist. As a consequence, UML tools are usually designed to support mainstream software development and provide therefore very poor means of validating real-time properties. All these considerations have led us to differentiate three kinds of actions to perform:

- Adapt the UML to real-time issues, by adding or extending (with UML profiles) native model elements to provide proper ways to represent temporal features at the model level.
- Adapt existing validation strategies to our approach: this implies notably to bridge the gap between the UML and other more formal languages, and between UML modeling tools and validation tools.
- Ensure the suitability of the application with respect to its embeddability, by trying to express the HW platform characteristics at the model level and thus enabling to make design choices in accordance.

These directions have been applied and refined all along the design of our platform. In the next sections, we account for this process, by describing works addressing several specific aspects of MDA adaptation to real-time/embedded issues.

3.2 A generic architecture for smart-sensor networked application

This thread of work is focused on architectural aspects when dealing with embedded applications. Smart-sensor networked applications stand as the prototypical example of such complex, but fairly common type of architectures, featuring both a central computing resource, such as an embedded PC, and several electronic devices, such as sensors and actuators.

To cope with the integration of such heterogeneous type of both hardware and software pieces, the component paradigm is of great help. It helps give a likewise abstract view of the various parts of the system. Then the question remains as how to integrate those parts, given that, most of the time, each of them is devised in an independent fashion, which prevents from easy coping of, among many, communication matters.

This work [18, 19] is an attempt to provide a generic integration scheme in the form of a component-partition of such networked applications. In his proposition, a sensor is represented both at the application (embedded PC) and at hardware level by various components:

- At the hardware device level, the sensor is seen as two components. The first one provides interface to the hardware logic. It is specified according to the existing standards (IEEE and OMG [20, 21]). The second one embeds the user logic and functional features. Apart from some predefined interfaces it is left to the engineer to develop

- At the embedded PC level, each sensor is represented by a device-driver component, which realizes one among predefined communication patterns and provides specific service interfaces to talk to the sensor through the network. Both CAN and Ethernet protocols have been taken into account so far and special care has been given to the design of synchronization algorithms among all the device driver components, so that the whole communication delay is handled. All this logic is embedded in the device-driver model construct, from which code can then be generated.

This is a typical example of what MDA is promoting. The definition of generic integration patterns, giving a sound basis for modeling, which in turn, via code generation, is finely-tuned to specific targeted electronic device platforms.

3.3 Schedulability and performance analysis

This thread of work can be seen as a general concern for assessing system properties – functional as well as extra-functional – at the level of the model. Among those, schedulability and performance stand as the most severe ones that embedded systems are expected to provide.

Three subsequent PhD thesis have been led on this topic within our team. The pursued goal was, on the one hand to broaden the coverage of those aspects within the UML, and on the other hand to bridge the gap between such UML modeling constructs and the use of external validation tools.

As concerns the first aspect, a dedicated UML profile was designed to define a generic Action Language [10] suitable for expressing control and functional algorithms in an implementation-language-independent fashion. Besides, another profile was developed to enable the expression of worst-case execution time (WCET) properties at the model level. Based on these, both a static and a dynamic WCET analysis of the overall models are possible.

The second part consisted in an effort to derive from Accord/UML standard models a specific, scheduling-oriented model [16], suitable for interpretation within a symbolic execution based validation tool, Agatha [13, 14], developed at CEA-LIST. Once established, this link between both tool-chains enables a complete assessment of the scheduling properties of an application model, provided that WCET information is fed to the models. This approach was mainly intended for critical real-time systems, for which precise knowledge of WCET are more likely to be known.

Finally, an ongoing work is focused on performance assessment, based on the use of the enhancement of the Scheduling, Performance and Time UML profile [22] and on the generation of Layered Queuing Networks

(LQN) [23] from standard Accord/UML application models. In the same way as was done for the scheduling issue, we foresee here the opportunity to bridge the gap with tools that were devised to extract LQN properties.

3.4 From models to code

Real-time embedded systems have to meet various design constraints including consumption of energy or memory and a sufficient level of performance to satisfy real-time requirements. There are actually several kinds of real-time embedded systems. They cover a wide range of domains going from cell phones applications to nuclear power plants control systems or also spacecrafts embedded calculators. Each domain has its own constraints to meet. This concern takes place at every stage of the design process but the fact that code of the application is the last link in the chain, makes code generation an essential and critical phase of model-centric development. Code provided by generators has to meet constraints of the RT/E application itself but also has to take into account the limitation of the resources provided by the hardware supports of the application. Hence, code generation needs to be optimized for each targeted platform depending of the features it offers. Besides, there are often several solutions to generate code from a given model. For example, a state-transition model may be generated under the form of a set of nested switches [24], or using the state pattern [25], but also by generating tables. These three patterns of code generation will not have the same impact on energy, memory and performance features of the generated code.

These different patterns of code generation are proposed in the Accord/UML workbench. Hence, code for different optimization purposes can be generated from the same model. Currently, the user has to manually make the choice of the code generation pattern, but our goal is to have a "smart" code generator capable of making this choice as automatically as possible. More than constraints specifications, this requires an elaborated platform description model to gather sufficient amount of information and we also need to elaborate some heuristics, keeping in mind combinatorial explosion issues, to be able to make the most appropriate choice.

Another axis of our ongoing work on optimized code generation concerns the ability to quantify the efficiency of the generated code in terms of energy, memory or performance in order to validate the fulfilment of requirements. From our point of view, this is a very important challenge in order for model-driven development to be a success in real-time embedded system development domain.

4. CONCLUSIONS

We strongly believe that the MDA approach, or more generally design processes centered on models design, constitutes a powerful mean to facilitate real-time embedded systems development. However, this statement will be completely true only if support tools and design processes guidelines are defined and refined, taking into account the very specific aspects of this application domain.

This paper expands this core rationale by describing the Accord/UML platform, a combination of an MDA-compliant methodology and a supporting workbench for developing real-time systems. In Accord/UML, going through development process is eased by models transformation automation and code generation, and support is provided until code execution. In order to mitigate concerns linked to implementation, a seamless integration of embedded and real-time features is performed all along the development process, for instance by providing methods and tools for temporal validation, or by extending the UML to a "real-time UML". The relevance of our approach has been assessed in various applications from the automotive and telecom industry in the context of European project such as AIT-WOODDES, EAST, or ARTIST.

REFERENCES

1. OMG, *Unified Modeling Language: Superstructure Version 2.0.* 2003.
2. B.P. Douglass, *Real-Time UML : Developing Efficient Objects for Embedded Systems.* Object technology Series, ed. Addison Wesley. 98.
3. Bran Selic, Garth Gullekson, and Paul T. Ward, *Real time Object-oriented Modeling.* Wiley Professional Computing. 94: John Wiley & Sons, Inc.
4. OMG, *MDA Guide Version 1.0.1.* 2003, OMG.
5. Sébastien Gérard, et al. *Efficient System Modeling of Complex Real-time Industrial Networks Using The ACCORD/UML Methodology.* in *Architecture and Design of Distributed Embedded Systems (DIPES 2000).* 2000. Paderborn University, Germany: Kluwer Academic Publishers.
6. S. Gérard, F. Terrier, and Y. Tanguy. *Using the Model Paradigm for Real-Time Systems Develoment: ACCORD/UML.* in *OOIS'02-MDSD.* 2002. Montpellier: Springer.
7. François Terrier, et al. *A Real Time Object Model.* in *TOOLS Europe'96.* 1996. Paris, France: Prentice Hall.
8. Sébastien Gérard, et al. *A UML-based concept for high concurrency: the Real-Time Object.* in *The 7th IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC 2004).* 2004. Vienna, Austria.

9. Chokri Mraidha, et al. *A Two-Aspect Approach for a Clearer Behavior Model.* in *The 6th IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC'2003).* 2003. Hakodate, Hokkaido, Japan: IEEE.
10. Chokri Mraidha, et al., *Action Language Notation for ACCORD/UML.* 2003, CEA.
11. Patrick Tessier, et al. *A Component-Based Methodology for Embedded System Prototyping.* in *14th IEEE International Workshop on Rapid System Prototyping (RSP'03).* 2003. San Diego, USA: IEEE.
12. Softeam,*Objecteering*,http://www.obecteering.com.
13. C. Bigot, et al. *Automatic test generation with AGATHA.* in *TACAS.* 2003. Warsaw, Poland.
14. D. Lugato, et al., *Validation and automatic test generation on UML models : the AGATHA approach.* special issue of the STTT (Software Tools for Technology Transfer), 2004.
15. C. Bigot, et al. *A Semantics for UML specification to be validated with AGATHA.* in *ERTS'04.* 2004. Toulouse, France.
16. Trung Hieu Phan, et al. *Scheduling Validation for UML-modeled Real-Time Systems.* in *ECRTS 2003.* 2003. Porto, Portugal.
17. David Servat, et al. *Doing Real-Time with a Simple Linux Kernel.* in *RTLWS'2003.* 2003. Valencia, Spain.
18. C. Jouvray, et al. *Smart Sensor Modeling with the UML for Real-Time Embedded Applications.* in *IV2004.* 2004. Parma, Italy.
19. C. Jouvray, et al. *Networked UML modeling sensors.* in *ICTTA'04.* 2004. Damascus, Syria.
20. *IEEE Standard for a Smart Transducer Interface for Sensors and Actuators,* in *Network Capable Application Processor (NCAP) Information Model, IEEE Std 1451.1.* 26 june 99.
21. OMG, *Smart Transducers Interface - OMG.* 07 dec. 01.
22. OMG, *UML Profile for Schedulability, Performance and Time (ptc/02-03-02).* 2003, OMG. p. 154.
23. D.C. Petriu and C.M. Woodside, *Performance Analysis with UML: Layered Queuing Models from the Performance Profile,* in *UML for Real: Design of Embedded Real-Time Systems.* 2003, Kluwer Academic Publishers.
24. Miro Samek, *Practical Statecharts in C/C++: Quantum Programming for Embedded Systems.* 2002.
25. Erich Gamma, et al., *Design Patterns. Elements of Reusable Object-Oriented Software.* 1994: Addison-Wesley.