High level modeling for Real-time applications with UML & MARTE

Julio L. Medina and Alejandro Pérez Ruiz
Departamento de Electrónica y Computadores, Universidad de Cantabria. Santander, Spain
{julio.medina, alejandro.perezruiz}@unican.es

***Abstract*—This paper shows initial results and the research path in a methodology to use UML & the UML Profile for MARTE in the design of real-time applications. The modeling constructs used are those proposed in the High Level Application Modeling chapter of the MARTE standard. These elements are at a high abstraction level, and hence they need to be complemented with a number of constraints and rules of usage in order to get a consistent set of transformations to obtain code and analysis models automatically from them. The rules and patterns proposed in this effort are meant to address increasingly complex design intents. As a starting point in the methodology this paper shows some of the basic ones, concretely the simple independent tasking model, the passive protected data sharing, and the distributed end-to-end flows of linear execution. The models here defined are suitable to be transformed into both: schedulability analysis models and code generation models. These models are also represented in UML as a previous step to its execution, the profiling of its execution times, and the schedulability analysis.**

Keywords—code generation; modeling; UML; MARTE; model-based schedulability analysis; MAST; Ada; real-time.

I. INTRODUCTION

Model-based software development is progressively taking momentum in industry as one of the most promising software engineering approaches. It helps to create and keep assets of many kinds along the development process. It facilitates the separation of concerns, increasing the process efficiency, and finally empowering the quality of software.

For real-time applications, a model-based methodology can also help to simplify the process of building the temporal behavior analysis models. These models constitute the basis of the real-time design and the schedulability analysis validation processes. With that purpose, the designer must generate, in synchrony with the models used to generate the application's code, an additional parameterizable model, suitable for the timing validation of the system resulting out of the composition of its constituent parts. The analysis model for each part abstracts the timing behavior of all the actions it performs, and includes all the scheduling, synchronization and execution resources information that is necessary to predict the real-time qualities of the applications in which such part might be integrated. In the approach here presented, these analysis models are automatically derived from high level design models annotated with a minimum set of real-time features taken from the requirements of the application in which they are to be used. Following the generation of the application's code as a composition of the code of its constituent parts, the complete real-time analysis model of the application can also be automatically generated from the composition of the set of real-time sub-models that form it.

The research effort that this paper presents considers the model-based development of hard real-time applications, for which the definition of the corresponding schedulability analysis models is an automated result of a chain of tools and techniques used in a model driven engineering approach. Our previous efforts in this direction can be read in [1]. In this context, this paper proposes the concrete modeling elements at a high level of abstraction, useful to conceive and elaborate the system using the Unified Modeling Language (UML) [2]. This is a general purpose modeling language standardized by the Object Management Group (OMG). This is used in conjunction with its standard extensions for Modeling and Analysis of Real-Time and Embedded systems, namely the UML Profile for MARTE [3]. There are other model driven similar efforts from the software engineering perspective derived from the ASSERT project, in [4] for example UML is used, though not fully based in standard modeling extensions like MARTE.

The most widely known use of model based development techniques comprises the generation of code from structural models like class diagrams. With those automations an initial set of skeletons of the classes and structural packages that form an application is usually easy to obtain. Also some form of reverse engineering is available through the usage of specially formatted "comments" placed as textual marks surrounding the space in the code files for the "bodies" of the operations. The final implementation code is then inserted (usually typed by hand) between the textual marks that are managed by the code generators. A further refinement that generates both, specifications and bodies from models, are code generators that use state machines for modeling the behavior of the classes. This mechanism uses the operations of a class as message handlers that trigger the events between states. That way the messages from other objects can interact with the automaton of the class, though in a non-predictable order. Then, this kind of code generators is not consistent with the required scenario-based description of real-time activities used for schedulability analysis.

For this reason a different approach to the code generation is necessary if we want to keep both models in tune in a way as automated as possible. Our tactic for generating the code that goes inside the marks of the structural skeletons is the use of the behavioral models given for each operation of the class.

This work has been partially funded by the Spanish Government under grant TIN2011-28567-C03-02 (HI-PARTES). This work reflects only the author's views; the funding organism is not liable for any use that may be made of the information contained herein.

These models are usually made just for descriptive or documentation purposes, but there is no reason for not using them precisely as a specification. For this labor the more adequate modeling elements are activity diagrams. The formalization of the textual code inside actions may be either the standardized action language [5] of the OMG, or specific annotations made in the target language that specify the concrete actions to be performed.

In the context of the methodology proposed in this approach, this paper contributes to clarify the process to use from a software engineering point of view, and to define the input modeling formalisms, using UML and MARTE for expressing the needs of the designer.

The paper is organized as follows: Section 2 presents a global view of the approach and situates the contribution of this work-in-progress paper in its perspective. It also makes a brief summary of the challenges, and presents related efforts. Section 3 presents the concrete modeling elements and rules used for modeling applications compliant to (A) the simple independent tasking model, (B) the passive protected data sharing and (C) end-to-end flows of linear chains of execution. Finally some conclusions and next steps to follow in our envisioned model based engineering approach.

II. CONTEXT OF THE MODEL-BASED APPROACH

As early mentioned, here we use UML as modeling language and the UML standard extensions proposed by the MARTE profile for annotating the necessary real-time aspects at different levels of specification. A synthetic view of the approach is shown schematically in Fig. 1.

The initial model used to describe the application and its real-time features is constructed using the MARTE extensions for high level application modeling (HLAM). From this formalism, two model-to-model (M2M) transformations are used. One, indicated as M2M_A in Fig. 1, is used to create the UML representation of the analysis model. This transformation is used to create a model for each real-time situation under analysis together with the model of the processing resources, and the workload to consider. For this model the schedulability analysis modeling (SAM) capabilities of MARTE are used. The other transformation, M2M_C, is used to generate an intermediate model ad hoc for the code generation. The intermediate model, called UMLforCode in Fig. 1, is a typical

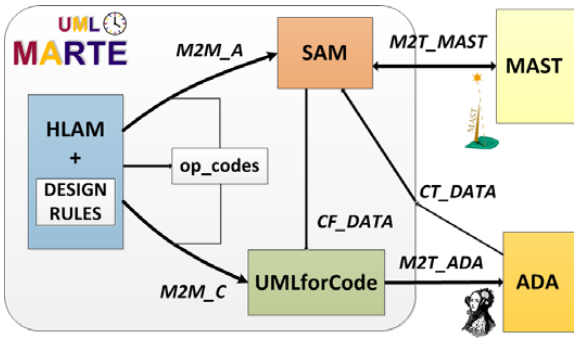

Fig. 1. Models and transformations used in our approach

UML object oriented generic model that comprises structural as well as behavioral information. The behaviors of the operations in this model are expressed by means of activity diagrams.

The model-to-text (M2T) transformation, denoted as M2T_MAST in Fig. 1, is used to generate the schedulability analysis models. It is part of our previous work [1]. An eclipse based tool [6] is available for the generation of analysis models, the invocation of the analysis tools, and the retrieval of results back into the modeling context. The tool then converts SAM models into the formalisms used by MAST [7] and then recovers its results back into the UML+MARTE model.

Another tool is also provided for generating Ada code from the UMLforCode object oriented generic model [8]. This is a model-to-text transformation, called M2T_ADA in Fig. 1. The code implemented out of the combination of M2M_C and M2T_ADA is consistent from the execution semantics point of view with the analysis models generated out of the combination of M2M_A and M2T_MAST. The transformation that generates the UMLforCode model includes the necessary instrumentation code that is used to measure and recover the approximate values for the worst, best and average execution times into the analysis model (a process called CT_DATA in Fig. 1). The op_codes table will help the transformations and tools to keep track of sections of code instrumented. Once the analysis is performed, scheduling analysis results are back annotated to the SAM models. These real-time configuration data include priorities (or relative deadlines) for the concurrent units, and priority ceilings (preemption levels or deadly floors) for shared resources. Called CF_DATA. in Fig. 1, these data are the configuration information in the UMLforCode generation model.

A. Design and analysis in the software development process

From a software engineering perspective a summary of the methodological steps to follow may be stated as:

(a) Introduction of design intent in UML using HLAM. The definition of the models to use for this step is the aim of the research work proposed by this work-in-progress paper.

(b) An initial schedulability analysis architectural validation may be done using speculative values for execution times using M2M_A in exploration mode, the extraction of schedulability analyses models with M2T_MAST, and the execution of the analysis tools (in this case MAST).

(c) Generation of UMLforCode model with M2M_C.

(d) Code generation (M2T_ADA) and execution in profiling mode (to use less speculative values in the analysis). Alternatively the code may be statically analyzed and executed with ad-hoc worst case execution time analysis tools.

(e) Generation of the SAM model with M2M_A, which now includes the recovered execution times (WCET)

(f) Extraction of final schedulability analysis models with M2T_MAST, and execution of the analysis tools (i.e. MAST)

(g) Recovery of analysis results in SAM and transposition of configuration data into the UMLforCode model.

(h) Generation of the final application code.

This paper proposes a way for HLAM models to be created so that all the transformations mentioned may work correctly. The transformations for generating UMLForCode (M2M_C) and analysis (M2M_A) models will be our next steps.

B. Related work

Following previous efforts that have studied the design of real-time systems using object oriented formalisms, we observe that most of them include the specification of the concurrency using structural models, usually at the design-for-implementation level. These dual structural-behavioral formalisms are made in the aim that this will help to realize schedulability analysis with the simple tasking model in mind and basic rate monotonic analysis (RMA) techniques later on. Unfortunately the complexity of the mechanisms used to generate the code makes this assumption not realistic, such as in ROOM [9], Octopus/UML [10], ACCORD/UML [12] [13], Comet [14], or the design model extremely constrained and monolithic such as in HRT-HOOD [15], OO-HARTS [16].

Being a syncretism of all those mentioned, and in order to ease the application of simple schedulability analysis techniques, the high level application modeling constructs in MARTE (see the HLAM section in [3]) also facilitate the use of structural models for the specification of the concurrency. But the interactions between them (including distribution) may take complex patterns that require a richer model for the analysis. Then, from the analysis perspective the end-to-end offset based analysis techniques scale far better to deal with these scenarios than the basic RMA tasking model. HLAM proposes two basic building blocks, the real-time unit: *RtUnit* and the passive protected unit: *PpUnit*. As for the behaviors in them (the code inside the marks), due to its natural complexity it is usually not just passive linear code that can be modeled as a computation time; instead they include delays, and interactions among objects and nodes, mostly when they become formed out of a composition of distributed operations (behavioral models). In these cases a state machine (the basic construct used in most of the analyzed approaches) is not directly transformable into an analysis model.

From the analysis perspective, the models that are required to apply the modern offset-based analysis techniques, are fundamentally scenarios. A scenario is an expression of the (worst case) expected or observable manifestation of the design intents (coded behaviors). This is the basis for coping with complexity that distinguishes RMA schedulability analysis techniques from those other strategies like the based on timed automata or synchronous languages.

As a modeling language for this domain, the scheduling analysis modeling section of MARTE (SAM) is also able to express that kind of scenario models, and then it is an adequate formalism to feed the corresponding analysis tools. Unfortunately these scenarios are not necessarily part of the initial specification of the system behavior. They are a means to express: the expected stimuli, the high level expected workload, and the end-to-end timing requirements, but they are usually not the basic data used for design intent or code generation drawn by the designers.

The creation of these (usually worst case) analysis oriented scenarios in tune with the final code is actually the main duty and a high responsibility of the real-time practitioner. In order to help in this labor the automation tools need the model used for code generation to have the behaviors of its operations expressed as scenarios. For this reason the adequate input models for the generation of the code inside the operations in the UMLforCode model are UML activities. This is why the tool that fills the code for the methods of the classes retrieves it from activity diagrams.

The use of scenarios has an additional benefit. This method helps to support the design of applications in terms of composable parts, which are closer in granularity to the concept of real-time objects than to the fully component-based software engineering (CBSE) interpretation of components. In a fully component-based approach, the creation of the analysis models would have to be made as a combination of both, structural elements plus their deployment. In an object-oriented model-driven approach, this later strong form of composability is in a higher level of abstraction, but still may benefit of the approach here described in order to assess a variety of non-functional properties, in our case of course the assessment of its timing properties by means of schedulability analysis.

C. Contribution of the effort here described

The contribution of this work-in-progress paper is in the clarification of the approach, the steps to follow in a software engineering process, and the initial identification of rules and concrete modeling elements in UML and MARTE so that suitable design models may be processed by the tools that the full model-based methodology presented comprises.

III. HIGH-LEVEL MODELING RULES

The basis for modeling with schedulability analysis in mind is the specification of three basic models, the platform, the logic of the application and the workload the system is expected to support. An initial set of modeling rules, which included those for describing the platform, was proposed in [1]. Here we enhance and extend it to address also code generation. For those terms in italics refer to the MARTE specification [3].

A. Modeling independent tasks

In cases where tasks are independent, the basic rules for describing the logic of the application are:

1. Each *RtUnit* have only one *schedulableResource* (thread) on it. Its behaviors (operations) may not be called from other *RtUnits*, and run under the scheduling parameters associated to that schedulable resource. Behaviors called in other passive classes run under the scheduling parameters of the calling *RtUnit*.
2. Each *RtUnit* has one and only one of its operations (UML BehavioralFeatures or behaviors) with the stereotype *RtFeature*. This has an *RtSpecification* (a comment stereotyped) in which at least the attribute *occKind*, has to be specified. This attribute indicates the *ArrivalPattern* (the triggering scheme) of the underlying task (usually a periodic pattern).

3. All the *RtUnits* deployed in a *processingResource* (a host) are handled by the same scheduler and use the same (or fully compatible) scheduling policy.
4. Each *RtUnit* whose *isMain* attribute is set to true, implies the presence of an execution host where the main service of the *RtUnit* is deployed.
5. The attribute *srPoolPolicy* holds the value *infiniteWait*

B. Modeling share data interactions

When tasks share passive data the *PpUnit* modeling construct is used, in this case these additional rules apply:

6. The *ExecKind* of *PpUnit* services is *ImmediatRemote*
7. All services of the *PpUnit* use the same protection protocol: *ImmediateCeiling* or *PriorityInheritance*
8. The *ConcurrencyPolicy* of *PpUnit* is *Guarded*.

The *concurrencyPolicy* of the kind *Concurrent* might be enabled in order to have the writer/reader *ConcurrencyKind* available, but this behavior requires additional capabilities from the analysis techniques to take really advantage of it, so in principle it is discouraged.

C. Modeling end-to-end flows

When tasks interact by triggering one another, chains of actions need to be ensemble. In this case the calling of the first action (task) in the chain determines the execution periodicity and end-to-end deadline. The following rules apply in this case:

9. The first restriction in rule 1 is here relaxed so that operations stereotyped as *RtServices* of *RtUnits* may be invoked by others using the SignalEvent semantics.
10. In this case, in order to have analyzable models, only the first calling operation in the chain may have an *ArrivalPattern* specified be means of its corresponding *RtFeature* and its *RtSpecification* comment.
11. Operations in an *RtUnit* that are not stereotyped as *RtServices* run in the context of the calling task. They are called passive and use the CallEvent semantics.

See [1] for additional rules that apply in general in specific phases of the development process.

The invocation of behaviors is made in activity diagrams. *sendSignalActions* are used for triggering *RtServices* and *callActions* for calling passive operations. The invocation of an *RtService* that holds an arrival pattern implies the initialization of the task (usually invoked in the main). This auxiliary code will be automatically inserted in the activity diagrams of the UMLforCode generation model. This will be done following the arrival pattern of the task (usually periodic).

The constraining rules described here for this HLAM input model are meant for ensuring (i) analyzability by means of schedulability analysis (ii) consistency between the analysis models and the generated code, (iii) the minimum usage of tools and transformations, and (iv) compliance with executable versions of UML, fUML [17] and the future standard for a Precise Semantics of UML Composite Structures [18].

IV. Conclusions and Future Work

This paper presents a model-based software engineering methodology for the development of real-time applications. Some steps in the necessary chain of tools have been realized and this paper shows some of the basic steps missing. It addresses the simple independent tasking model, the passive protected data sharing, and the distributed end-to-end flows of linear execution. The models compliant to the rules here defined are suitable to be transformed into both: schedulability analysis and code generation intermediate models. Next steps include, the high level transformations into these intermediate models, experiments, rules to handle interrupts, and tooling support for the complete iterative engineering process.

References

[1] J. Medina and A. Garcia Cuesta. Model-Based Analysis and Design of Real-Time Distributed Systems with Ada and the UML Profile for MARTE. In Proc. of the 16th International Conf. on Reliable Software Technologies-AdaEurope 2011, LNCS 6652, pp 89-102.

[2] Object Management Group. Unified Modeling Language version 2.4.1, OMG document formal/2011-08-06, 2011.

[3] Object Management Group, UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems, version 1.1, OMG doc. formal/2011-06-02, 2011.

[4] Silvia Mazzini, Stefano Puri, and Tullio Vardanega. An MDE Methodology for the Development of High-Integrity Real-Time Systems. Proceedings of Design, Automation and Test in Europe. Nice, France. April 20-24, 2009.

[5] Object Management Group. Action Language for Foundational UML (Alf), Concrete Syntax for a UML Action Language. OMG document ptc/2010-10-05, 2010.

[6] http://mast.unican.es/umlmast/marte2mast

[7] M. González Harbour, J.J. Gutiérrez, J.C.Palencia and J.M.Drake, MAST: Modeling and Analysis Suite for Real-Time Applications, in Proc. of the Euromicro Conference on Real-Time Systems, June 2001.

[8] http://mast.unican.es/umlmast/uml2ada/

[9] Bran Selic and Jim Rumbaugh. Using UML for Modeling Complex Real-Time Systems. Rational white papers, http://www.rational.com/products/whitepapers/UML-rt.pdf, March 1998

[10] Domiczi, R. Farfarakis and J. Ziegler. Octopus Supplement Volume 1. Nokia Research Center. http://www-nrc.nokia.com/octopus/supplement/index.html, 1999.

[11] Laila Kabous. An Object Oriented Design Methodology for Hard Real Time Systems: The OOHARTS Approach. Doctoral Theses, School Carl von Ossietzky, Universität Oldenburg. 2002

[12] F. Terrier, G. Fouquier, D. Bras, L. Rioux, P. Vanuxeem and A. Lanusse. A Real Time Object Model. Presented in TOOLS Europe'96. Paris, France. Prentice Hall, 1996

[13] A. Lanusse, S. Gerard and F. Terrier. Real-Time Modeling with UML: The ACCORD Approach. In Selected papers from the 1st. Int. Workshop on The Unified Modeling Language UML'98: Beyond the Notation. Mulhouse, France, June 3-4, 1998. Pp. 319-335. ISBN:3-540-66252-9.

[14] Hassan Gomaa. Designing Concurrent, Distributed and Real-Time Aplications with UML. ISBN 0-201-65793-7, Addison-Wesley, 2000.

[15] Alan Burns, Andy Wellings. HRT-HOOD, a structured design method for hard real-time ADA systems. ISBN 0 444 82164 3. Elsevier, 1995

[16] Mazzini S., D'Alessandro M., Di Natale M., Domenici A., Lipari G. and Vardanega T. HRT-UML: taking HRT-HOOD into UML. In Proc. of 8th Conference on Reliable Software Technologies Ada Europe, 2003

[17] Object Management Group. Semantics of a Foundational Subset for ExecutableUML Models (fUML), v1.1, OMG Document: ptc/2012-10-18. http://www.omg.org/spec/FUML/1.1

[18] Object Management Group. Precise Semantics of UML Composite Structures, Request For Proposals. OMG Document: ad/11-12-07