An MDE Methodology for the Development of High-Integrity Real-Time Systems

Silvia Mazzini, Stefano Puri
Intecs SpA, Pisa, Italy
Email: {silvia.mazzini, stefano.puri}@intecs.it

Tullio Vardanega
University of Padua, Italy
Email: tullio.vardanega@math.unipd.it

***Abstract*—This paper reports on experience gained and lessons learned from an intensive investigation of model-driven engineering methodology and technology for application to high-integrity systems. Favourable experimental context was provided for by ASSERT, a 40-month project partly funded by the EC as part of the 6th Framework Program. The goodness of fit of the MDE paradigm for the industrial domain of interest was critically assessed on a small number of candidate solutions. One of the main axes of investigation concerned HRT-UML/RCM, an advanced method and integrated tool for the model-driven development of embedded real-time software systems. HRT-UML/RCM vastly leveraged on version 2 of the OMG UML standard and combined it with the development of a domain-specific metamodel in the quest to attain correctness-by-construction from the ground up. The prototype tool developed in the project supported: (1) the separation of functional (sequential) design from the specification of real-time and concurrency requirements and properties to be preserved at run time; and (2) the exploitation of a fully generative approach to the development, equipped with support for model-based feasibility analysis and round-trip engineering.**

I. Contribution

HRT-UML/RCM, an integrated method and infrastructure for the development of embedded real-time software systems, whose acronym stands for "Hard Real-Time UML for the Ravenscar Computational Model", was one of the major results of the ASSERT (Automated proof based System and Software Engineering for Real-Time Applications) project [1] partly funded by the EC in the 6th Framework Program.

In this paper we first review the conceptual basis of the work that led to HRT-UML/RCM and introduce the main principles of the development approach that the method promotes. We discuss the core methodological aspects of the methodology, using accompanying examples, and relate our approach to MARTE, the new UML profile for the modeling of real-time and embedded systems, recently adopted by OMG. We comment on the results of the pilot experiments made to evaluate the goodness of the tool prototype and conclude by outlining the future directions of this line of work.

II. Introduction

A. Background

HRT-UML/RCM builds on conceptual and technical grounds laid down in [2], [3], [4], from a definition initiated by Intecs with research projects co-funded by the Italian Space Agency in 2001-4. The definition projects aimed at providing a comprehensive solution to the modeling of hard real-time and dependable systems, by upgrading the principles of HRT-HOOD [5] from object basedness to true (though constrained) object orientation and by incorporating them into version 1 of the UML [6]. HRT-HOOD was deemed a valuable conceptual basis, proven by years of successful use in European space industry. UML instead was chosen as the host infrastructure in view of its acknowledged stance in industrial practice as a *de facto* standard. The initial methodology was consolidated under a contract with the European Space Agency, targeted to the development of the real-time software systems for on-board applications in the space domain. With further EC funding from the 5th Framework Program, Intecs augmented the resulting concept with support for addressing control engineering aspects, by integrating Simulink functional design, as well as by addressing verification needs, ranging from combined simulation of the plant and the controller, to schedulability analysis and scheduling simulation [7].

B. The Model-Driven Engineering Approach

Model-driven development is a novel engineering paradigm that facilitates the definition, composition and integration of complex software systems. Model evolution through refinements, transformations and code generation, possibly automated by tool support, form the basis of model-driven engineering (MDE). In the MDE vision, software models are elevated to a central and governing role in the development by reaching for a higher level of abstraction than is possible with current third-generation programming languages [8]

A popular variant of MDE is the Model Driven Architecture (MDA), a major initiative of OMG to achieve a cohesive set of model-driven technology specifications [9] that use version 2 of the OMG general-purpose UML language [10] and/or specific profiles of it.

The main goal of MDA is to: (1) separate business and application logic from the underlying execution platform technology; (2) focus developers on the production of models of the application and of the relevant business logic; and (3) support the generation of platform-specific models and code by means of engineered, and possibly automated, transformations.

HRT-UML/RCM applies the MDA approach to provide an architectural framework where: (i) the designer may define platform-independent models; (ii) platform-specific models are automatically produced from platform-independent specifications using proven model-to-model transformations; (iii)

978-3-9810801-5-5/DATE09 © 2009 EDAA

platform-specific models are submitted to static analysis for feasibility in time and, possibly, other non-functional dimensions, and the analysis results are back propagated to the designer's view; (iv) once the user model is committed, its platform-specific correspondent becomes input to automatic generation of source code for the target platform.

The platform-specific modeling space in HRT-UML/RCM conforms to the Ada Ravenscar Profile, a recognized standard language subset for the programming of high-integrity real-time systems. The Ravenscar Profile caters for a reduced, efficient and certifiable model of concurrency that, in language-neutral terms, has come to be known as the Ravenscar Computational Model (RCM) [11].

The use of RCM coupled with a code generation strategy that enforces the coding restrictions typical for high-integrity software (e.g., no recursion; unbounded loops allowed exclusively to permit a thread structure to issue jobs with the desired timing behavior; no dynamic memory; etc.) guarantees that models are by definition amenable to static analysis for timing, scheduling, feasibility and sensitivity. It also makes them easier to treat by the other forms of verification typically required by high-integrity process standards.

C. HRT-UML/RCM: a Domain Specific Modeling Language

Rooted in the conceptual basis of the OMG MOF (Meta-Object Facility) [12], the core metamodel language used to define UML itself, HRT-UML/RCM defines a formal domain-specific language that promotes the separation of functional modeling from the design of the real-time architecture in which those functional models are to be incorporated, and also allows the derivation of a consistent platform-specific model that preserves the properties stipulated in the modeling space provided to the user, enables static analysis with round-trip on the model, as well as automatic code generation.

The HRT-UML/RCM metamodel follows the heavy-weight metamodeling strategy in the classification of [13] to reduce the complexity arising from the comprehensive general-purpose nature of the UML. It does so by defining specific concepts and notations, attributes, properties, relations and constraints which limit the design space so as to guarantee safer and more reliable design, to permit earlier detection and removal of development errors, and to preserve the cohesion and consistency of the approach.

III. HRT-UML/RCM Architectural Framework

The IEEE 1471 document "Recommended Practice for Architectural Description of Software-Intensive Systems" (now known as ISO/IEC 42010:2007) [14] introduces a conceptual framework that provides guidance for structuring and organizing architectural descriptions into different views. In this context, views are intended as a representation of a whole system seen from the perspective of a related set of concerns.

In keeping with this notion and thus striving to promote separation of concerns, HRT-UML/RCM provides an architectural framework that organizes the design space into distinct views, which individually address a subset of the concerns commonly addressed by software system architects: for the algorithms; for timing, concurrency and synchronization; for physical design, configuration and deployment. More dimensions of concern exist of course (e.g., safety) which we plan to incrementally address in the future.

HRT-UML/RCM organizes its modeling space in the four following views: (1) the *functional view*, in which the designer defines the sequential algorithmic behavior of the functional components of the system using UML class and state-chart diagrams; (2) the *interface view*, where the designer specifies how functional components aggregate together and what interfaces they offer and require and the relations traced among them, by augmenting the specification of the corresponding functional signatures with attributes that specify the desired concurrency semantics; (3) the *deployment view*, which specifies the physical components of the system, in terms of nodes (processors, memory, devices), partitions, interconnects, protocols, capabilities and restrictions, and allocation relations with software components; (4) the *concurrency view*, which defines the concurrent architecture of the system in terms of threads of control, communication and synchronization protocols and resources to "realize" the execution semantics specified in the interface view for the target platform specified in the deployment view, assuming an execution environment that complies with the RCM.

The functional and the interface views operate in the platform-independent modeling space, while the concurrency view and the deployment views are platform specific. The overall articulation of views is depicted in figure 1.

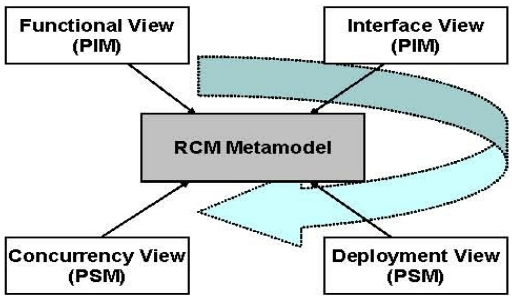

Fig. 1. The design space in HRT-UML/RCM.

The concurrency view is automatically generated from the other views. All views address distinct and non-overlapping aspects of the system, yet they are cohesively related one to another by the metamodel rules, which also determine the semantic rules for the related elements.

In the following we discuss the essential characteristics of the modeling space supported by HRT-UML/RCM.

A. Functional view

The functional view is the design space for the modeling of the sequential semantics of the system operation; this view intentionally ignores all timing, concurrency and synchronization issues that are typical of real-time system design. Those aspects are addressed separately in the interface view by

adding the required level of semantic annotation to the entities specified in the functional view.

The functional view captures structural and behavioral information in the functional domain. Structural information can be expressed with standard UML class diagrams, and thus by the definition of classes, operations and attributes. Interface realization relations between classes and provided interfaces and require relations between classes and required interfaces can also be specified. Object-oriented semantics, such as inheritance and overriding, are also allowed as long as they can be statically resolved.

The specification of behavioral information is an important step in the functional design process. It is used to feed model transformation and code generation. For instance, given a class, the specification of the attributes upon which an operation acts allows accurate derivation of the concurrency view, while the specification of how many times the execution of an operation will invoke a required operation allows accurate timing analysis (which will of course only be performed against the choice of a specific execution platform, the specification of which falls outside the scope of the functional view). Figure 2 depicts an example of the functional view of a simple controller-actuator system, which will serve as a running example throughout this paper.

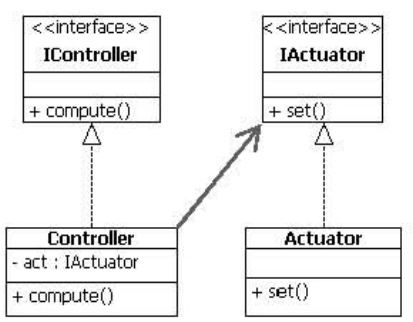

Fig. 2. A model in the functional view.

The HRT-UML/RCM metamodel implements a profile of the UML state machine in a way that, for instance, time-related information cannot be modelled. Moreover, ad-hoc behavioral metamodel entities have been defined in the place of other existing behavioral UML constructs. We chose this simplification on account of contingent project constraints: better exploitation of the UML behavioral diagram capabilities in this context is in our road map.

B. Interface view

The interface view is the design space for the modeling of the concurrency features of the system; in this view the components specified in the functional view are augmented with concurrency-related characteristics.

The primitive class entity that populates this view is termed Application-level Container (APLC) in the following. APLC are modelled first as types, and then they are instantiated and interconnected to form a precise and verifiable instance of the real-time software system under design.

An APLC is a stereotyped UML structured component; it can own parts that can be typed only with a (non-abstract) class defined in the functional view. Parts are the only features that the designer can create/delete for an APLC.

When typing a part, the provided and required interfaces of the part itself (those that derive from the typing class, in accord with UML rules) are promoted from the part up to the owner APLC. Consequently, the types of the parts automatically determine the only provided interfaces (PI) and required interfaces (RI) that can and must be exposed by the owner APLC.

Provided and required interfaces are automatically exposed by the APLC through stereotyped ports, called port clusters in HRT-UML/RCM, one port cluster per interface. A port cluster is a special UML composite port which aggregates elementary interaction points, called elementary ports. A port cluster owns one elementary port for each operation specified by the related interface. An elementary port acts as a wrapper of the associated operation; the role of this wrapper is to allow the specification of the kind of the synchronization protocol that has to be applied when the operation, implemented by the functional part, is invoked through the owner port. The range of protocols currently supported is limited to the classical forms of: (i) exclusion; and (ii) avoidance synchronization. Additional options are being considered, for example to address application-specific flavors of data freshness. Overall, port clusters are the interaction points through which APLC can be interconnected, as in component-based design.

In the interface view, the designer creates the APLC, together with their parts, and then attaches semantic decorations to all elementary provided ports (EPP), to specify the concurrent behavior that shall occur on invocation of each wrapped operation. Decoration of elementary required ports (ERP) is also possible but not mandatory.

Figure 3 shows the representation of an APLC realized with our prototype tool: it basically follows the UML notation for components. Regarding the port cluster notation in the figure,

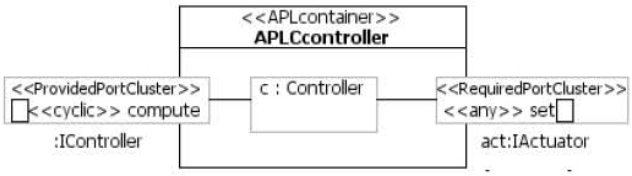

Fig. 3. An APLC in the interface view.

the elementary ports are not shown on the port cluster border (as it would follow the UML style) but they are listed inside the port cluster: this notation has been chosen to attain better graphical representation and support.

The port cluster and elementary port entities permit finer-grained modeling of component interactions, at the level of single operation of the interface, than plain UML allows at the level of the whole interface. This is one of the distinct benefits of adopting the heavy-weight metamodeling approach

instead of classic profiling. Details of the port cluster notation are shown in figure 4.

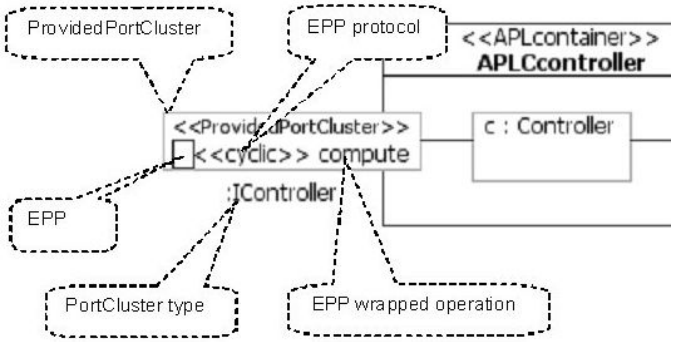

Fig. 4. Details of the PortCluster notation.

Attaching concurrency semantics to elementary ports and referring to functional entities via APLC parts facilitates the reuse of functional specifications across multiple execution scenarios. This is a substantial difference from other methodologies, such as HRT-HOOD [5] and our previous HRT-UML work [2], [3], where functional and concurrent information are tightly and inextricably bound to one another.

Following the UML, an instance of an APLC classifier comes with a set of slots [10] to specify the values of its structural features (e.g. classifier attribute). In particular, an APLC instance has one slot for each port cluster/elementary port owned by the classifier. A port cluster slot allows the interconnection of APLC instances; a slot associated to an EPP allows the designer to attach attributes to the concurrent information already modelled with the EPP. For details on the supported attributes see [15], [16].

The interpretation of HRT attributes and values takes place in the PSM space. HRT attributes have been introduced in the interface view to allow the designer to run feasibility analysis on the system model and to see the results fed back from it to this view (in what amounts to a segment of model-based round-trip engineering). In this way the designer doesn't have to understand and manipulate PSM models: the rationale behind this choice is to avoid PSM model editing that could jeopardize the guarantee of correctness of the transformational approach realized by the HRT-UML/RCM tool infrastructure.

In the interface view the APLC instances have to be interconnected in order that each required port cluster instance is bound to a provided port cluster instance that is contractually compatible with it. Semantic compatibility follows from the matching of the associated types (according to object-orientation rules) while contractual compatibility is based on value matching of the involved port instances (for example, the protocol kind of the referred ERP and EPP).

C. Deployment view

The deployment view is the modelling space where the designer specifies the physical architecture of the system and commands the way the logical architecture must map to it. The notion of Partition models a logical node, in particular a fault containment region where APLC instances can be deployed: the APLC within a partition are isolated in space, time and communication from all the other partitions in the system. (Attending to the required level of isolation is the responsibility of the RCM execution platform that runs on each node of the system). A single "ComputationalNode" may host multiple logical partitions; "ComputationalNodes" can be connected through physical links.

HRT-UML/RCM simplifies the UML deployment view: APLC instances are allocated directly to a node whereas the original UML notion of deployment interposes the artifact element, to concretize the software to use for allocation purposes. The usefulness of the artifact model entity in the embedded domain still needs to be demonstrated. Our simplification approaches the concept of allocation first developed in SysML [17] and then adopted in MARTE [18], to signify the fact that the execution platform is a relevant architectural decision.

D. Concurrency view

The functional and interface views constitute the software modeling space availed to the user. In deciding how to represent those views, both graphically and semantically, we sought a good compromise between abstraction and expressiveness. Once those views are modeled and the target platform is specified, a fully automated model transformation process may be launched to obtain the platform-specific model called the concurrency view.

The concurrency view describes the concurrent architecture of the system realized in compliance with the RCM for the target execution platform as specified in the deployment view. The concurrency view describes the RCM components that realize the concurrent semantics specified by the designer in the interface view. Those components are termed Virtual Machine Level Containers (VMLC) which are the sole legal entities accepted for execution on an RCM platform. The VMLC are typed and their legal types are as follows: passive VMLC, protected VMLC, sporadic VMLC and cyclic VMLC (see [19] for details).

The model transformation currently supported by our work proceeds in two steps: the first step generates VMLC types implementing each APLC on top of a RCM compliant platform; the second step generates interconnected VMLC instances starting from the interconnected APLC instances as specified in the interface view. The deployment of VMLC instances onto partitions is performed according to the deployment of the APLC instances specified in the deployment view.

APLC are transformed on an individual basis. EPP are grouped and every single group is transformed into a single VMLC following a set of production and semantic rules; the VMLC embeds the functional entities (parts) originally wrapped by the grouped elementary ports.

The model transformation engines embedded by HRT-UML/RCM are realized on the basis of a mathematical formalization that provably guarantees that no semantics distortion may occur in the PIM to PSM transformation [20]. By its very nature the concurrent view, together with the deployment view, is best suited to feed feasibility and sensitivity analysis

as well as code generation. For reasons of space limits we can't elaborate on the details of the mechanisms that feed this analysis and govern code generation: see [16] for details.

IV. The HRT-UML/RCM Process

The development process promoted by HRT-UML/RCM is inherently iterative and proceeds across the following four incremental steps, as depicted in figure 5:

1) *modeling phase*: the software architect starts from the functional view, with the specification of functional blocks. Functional blocks may also be imported from external modeling tools (hence expressed in terms of a foreign metamodel), so long as they do not violate the RCM constraints: in that case the mapping performed in the import must be proved to preserve semantics and to conform to RCM. Once functional blocks are complete, the software architect enters the interface view and embeds them in APLC; non-functional semantics (concurrency, timing, etc.) is defined in this modeling space. The required physical distribution is specified in the deployment view by assigning APLC to logical partitions and by deploying them on physical nodes
2) *model transformation*: the concurrency view is automatically generated from the user models, and the resulting architecture of VMLC is produced which specifies the implementation of the system on the RCM distributed execution environment
3) *feasibility and sensitivity analysis*: the platform-specific view produced in the concurrency view is submitted to feasibility and sensitivity analysis, with round-trip support for the software architecture to iterate over the modeling phase for improvements
4) *code generation*: once the software architect commits the model, the final source code for the system is automatically generated with appropriate bindings to the execution environment.

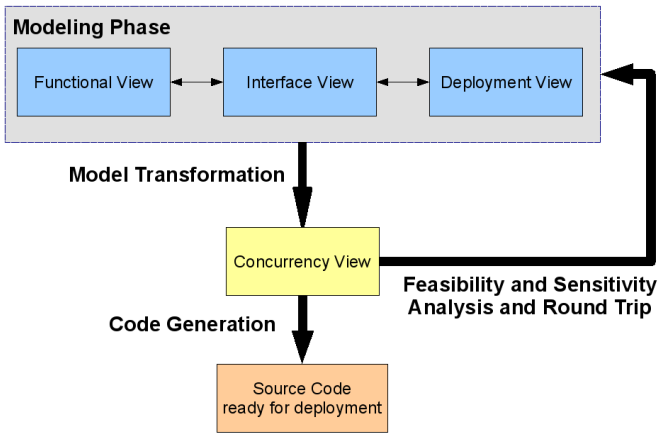

Fig. 5. The iterative nature of the HRT-UML/RCM process.

V. Industrial Evaluation

In the context of the ASSERT project a prototype tool supporting the methodology was developed and submitted to extensive industrial evaluation. Two alternative implementation solutions were explored: (i) an existing open-source UML tool (e.g., TOPCASED [21]) on which to implement an ad-hoc UML profile; (ii) a domain-specific metamodel on which to build a dedicated UML-like graphical editor from scratch. The goal to support correctness by construction in a bottom-up fashion (that is, from the very foundation of the metamodel) and the availability of mature technology from the Eclipse project [22], made the latter option appear more convenient. We therefore embarked on the development of a dedicated Eclipse plug-in. We used EMF for the implementation of the RCM metamodel, which eventually included over 90 metaclasses; 7,500 lines of Java to complete the 150,000 lines generated automatically by GMF for the graphical editor; 13,000 lines of ATL to drive model-to-model transformations; 8,000 lines of MOFscript to implement code generation; and 5,500 lines of Ada to extend MAST [23] and fit it to our needs for model-based feasibility and sensitivity analysis. The full prototype development took in excess of 5 person/years from June 2006 to July 2007. The resulting HRT-UML/RCM plug-in implements diagrammatic support for all four of the HRT-UML/RCM views and integrates model transformations for round-trip analysis and fully automated generation of Ada 2005 restricted to the RCM. See [15] for wink clips on how the HRT-UML/RCM toolset operates.

The prototype was evaluated by major industrial partners on technical and methological grounds. The case studies re-designed sizeable parts of industrial reference systems in the space domain. We briefly report on the plus and minus sides of the industrial evaluation.

A. Plus.

The integration of all modeling and analysis tools within a single development environment appears to be an industrial "must". In this regard, the integration level of all tools in the HRT-UML/RCM track was rated more than satisfactory. The next step requires the integration of support for requirements analysis and tracing.

The guarantee of consistency across model spaces was rated especially important. This was insured by a combination of metamodel-driven restrictions holding on the individual modeling spaces and the triggering of precondition constraint checking upon any model transformation step.

The declarative style of specification in the interface view was rated effective in permitting the user to abstract away from the complexity of the underlying implementation (captured by the concurrency view). This style apparently reflected the current trend of modeling practice in European space industry.

In UML a functional dependency is determined by the class members (much like in HRT-UML/RCM) or by the parameters of its methods. We only allowed the former possibility so as to guarantee that the RI be not directly dependent on a value in input to a PI call. In this manner the binding between RI and PI is always fully determined at design time, which facilitates static analysis and makes it more accurate and effective. This

choice furthermore enables us to always keep the functional view and the interface view consistent by construction.

B. Minus.

While the guaranteed consistency and synchronization of the functional and interface views earns major benefits to the modeling process, it was however felt desirable to permit the modification of PI/RI of APLC also from the interface view without having to go back to the functional view. In fact, the root cause of this limitation is purely technical, with no bearing on conceptual or methodological considerations.

Modeling support for action semantics is currently lacking, which industry regards as a serious deficiency. More research effort should be put on this area to integrate multi-purpose action modeling in the design languages, and treat action semantics as any other formalized model element.

The interface view currently lacks support for hierarchical decomposition, which makes it difficult to visually master the development of large systems. Luckily, the problem lies more in the graphical interface concept than in the methodology itself. Visibility control can in fact be enforced through attributes values, with the same net effect as hierarchical decomposition.

VI. Conclusions

HRT-UML/RCM leverages on distinct improvements to version 2 of the OMG UML standard modeling language and the solid basis of the Ravenscar Computational Model. It defines a formal domain-specific modeling language and a methodology that foster the separation of functional design from the design of real-time properties and concurrency semantics. It also permits the derivation of a consistent implementation that preserves the design properties across transformations, in full accord with the model-driven engineering paradigm.

The successful achievement of those objectives by industrial-quality technology will enhance to the current practice in several respects: (1) explicit consideration of non-functional requirements in the user model, with better control the predictability properties of the final software product; (2) higher software quality warranted by full traceability between the non-functional requirements captured by the user model and the non-functional attributes and properties exhibited by software components (VMLC) in isolation and in conjunction; (3) less costs in verification and validation thanks to proven automated transformations.

Future short-term activities aim at the consolidation of both the methodology and its enabling technology. At the top of the list we have: (i) support for hierarchical functional decomposition; (ii) behavioral modeling with statecharts, to increase the rate of functional code generation; and (iii) integration with complementary and/or domain specific languages and tools such as Matlab/Simulink, Scade, SDL; (iv) extensions to real-time attributes and concurrent semantics to the modeling of operation modes as well as of application-specific synchronization protocols.

Acknowledgements

The authors are grateful to Matteo Bordin, Marco Panunzio, Daniela Cancila, Marco Trevisan, Sue Maurizio (University of Padua) and Thanaële Preuss, Maria Rosaria Barone, and John Favaro (Intecs) for their vast contribution to the materials presented in this paper.

References

[1] "Assert project," http://www.assert-project.net.

[2] S. Mazzini, M. D'Alessandro, M. D. Natale, G. Lipari, and T. Vardanega, "Issues in mapping HRT-HOOD to UML," in *Proceedings of the 15th Euromicro Conference on Real-Time Systems*, 2003.

[3] S. Mazzini, M. D'Alessandro, M. D. Natale, A. Domenici, G. Lipari, and T. Vardanega, "HRT-UML: taking HRT-HOOD into UML," in *Proceedings of the 8th Int'l Conference on Reliable Software Technologies - Ada-Europe*, ser. LNCS(2655). Springer, 2003.

[4] S. Mazzini, M. D'Alessandro, M. D. Natale, and T. Vardanega., "Component-Based Real-Time Design: Mapping HRT-HOOD to UML," in *Proceedings of the 30th Euromicro Conference*, 2004.

[5] A. Burns and A. Wellings, *HRT-HOOD: A Structured Design Method for Hard Real-Time Systems*. Amsterdam, NL: Elsevier Science, 1995, no. ISBN 0-444-82164-3.

[6] Object Management Group, "OMG Unified Modeling Language Specification," 2001, version 1.4.

[7] S. Mazzini, J. Favaro, S. Puri, and M. Bavaro, "Software Methodology and Tool Support for Embedded Control Systems," in *Proceedings of the "Adaptability Techniques for Control System Software" session of the IFAC Conference*, 2005.

[8] B. Selic, "From Model-Driven Development to Model-Driven Engineering," http://feanor.sssup.it/ecrts07/keynotes/k1-selic.pdf, keynote talk at ECRTS'07.

[9] Object Management Group, "MDA Guide," omg/2003-06-01, 2003, version 1.0.1.

[10] ——, "UML Superstructure Specification," http://www.omg.org, 2006, final Adopted Specification, v. 2.1.

[11] A. Burns, B. Dobbing, and T. Vardanega., "Guide to the Use of the Ada Ravenscar Profile in High Integrity Systems," University of York (UK), Tech. Rep. TR YCS-2003-348, 2003.

[12] Object Management Group, "Meta Object Facility (MOF) Core Specification," http://www.omg.org, 2006, formal/06-01-01.

[13] D. Frankel, *Model Driven Architecture: Applying MDA to Enterprise Computing*. Wiley, 2003, iSBN:0-471-31920-1.

[14] "IEEE Recommended Practice for Architectural Description of Software-Intensive Systems," IEEE, Tech. Rep. IEEE Std 1471-2000, 2000, (ISO/IEC 42010:2007).

[15] D. Cancila, M. Trevisan, and T. Vardanega., "A gentle introduction to the HRT-UML/RCM methodology," http://www.math.unipd.it/~tullio/Research/ASSERT/Tutorial, 2007.

[16] M. Bordin, M. Panunzio, and T. Vardanega, "Fitting Schedulability Analysis Theory into Model-Driven Engineering," in *Proceedings of the 20th Euromicro Conference on Real-Time Systems*, 2008.

[17] OMG Systems Modeling Language (OMG SysML), "Final Adopted Specification," http://www.omg.org, May 2006, document ptc/06-05-04.

[18] H. Espinoza, H. Dubois, J. Medina, and S. Gèrard, "A General Structure for the Analysis Framework of the UML MARTE Profile," in *MARTES Workshop, 8th International Conference on Model Driven Engineering Languages and Systems*.

[19] J. Zamorano, J. de la Puente, J. Hugues, and T. Vardanega, "Runtime mechanisms for property preservation in real-time systems," in *Procedings of the Workshop on Operating Systems Platforms for Embedded Real-Time applications (OSPERT)*, K. Elphinstone, Ed., 2007, http://ertos.nicta.com.au/publications/papers/Elphinstone_07.pdf.

[20] D. Cancila, R. Passerone, and T. Vardanega., "Composability for high-integrity real-time embedded systems," in *1st Int'l Workshop on Compositional Theory and Technology for Real-Time Embedded Systems*, November 2008, http://www.cis.upenn.edu/~ishin/crts2008/crts2008.html.

[21] http://www.topcased.org/.

[22] http://www.eclipse.org/.

[23] University of Cantabria, Spain, "MAST: Modeling and Analysis Suite and Tools," http://mast.unican.es.