SOFTWARE METHODOLOGICAL AND TOOL SUPPORT FOR EMBEDDED CONTROL SYSTEMS

Silvia Mazzini, John Favaro, Stefano Puri, Michele Bavaro

Intecs S.p.A., via U. Forti - Trav. A, 5, I-56121 Pisa, Italy
silvia.mazzini@pisa.intecs.it

Abstract: The benefits of significant advances in general software engineering support environments in recent years have yet to be realized in the area of embedded control systems, due to a particularly demanding set of requirements. A toolset providing methodological and tool support is presented which addresses specific challenges to system development in the control domain including component-oriented development, reusability, object-oriented design, integration of third-party tools, and validation.

Keywords: software engineering, software tools, software reliability, validation, safety.

1. INTRODUCTION

The real-time and embedded systems market is a highly specialized area in which rapid progress has enabled the construction of ambitious distributed and multi-objective control systems. At the same time, the power and flexibility of available platforms has lead to an ever-increasing domination of the software component of complete systems solutions. Yet the provision of adequate software design methods and support still lags far behind. Consequently, today most of the cost in system development involves *ad hoc* expensive system integration and validation techniques, which rely almost exclusively on time-consuming simulations and on testing more or less complete versions of the system.

The industrial community, facing increasing commercial pressures, has now begun to demand the same powerful software design and development environments that are available to the software engineering community at large. This not only implies provision of specialized facilities such as scheduling algorithm verification, but also the ability to exploit the best practices that have evolved in the software engineering community over the years, including flexible, modular architectures, platform-independent design techniques, configuration management of program families, reusable software components – in short, the best results of the software engineering community over the past four decades.

The contribution of software engineering to the real-time embedded domain consists not only – not even most significantly – of *technologies* (such as reusable modules and code generators) but of *methodologies*, such as object-oriented design (Gamma, *et al*., 1995) and the use of standardized notations such as the Unified Modeling Language (Booch, *et al*., 1998). Methodologies and the best practices they embody have been relatively unknown to date in the real-time and embedded community, a domain where familiarity with software engineering is often acquired on an "as-needed" basis. The Unified Modeling Language (UML) has become the reference point around which tool vendors have organized their offerings to the real-time and embedded systems market. However, current UML tools lack support for the definition of explicit time constraints on the computations associated to design elements or to their collaborations, they do not offer an *a priori* analysis regarding the enforcement of the time constraints themselves, with respect to the system resource allocation strategy and the adoption of a suitable run-time computational model. Moreover none of these tools provides sufficient support for handling all the different development phases, especially for systems with a significant control component. These phases usually include design of control laws, supervision logic, real-time task scheduling, modeling of distribution and communication, etc. This paper describes a design methodology and environment that addresses such issues.

2. REQUIREMENTS FOR EMBEDDED CONTROL SYSTEM DEVELOPMENT

In order to provide a concrete context for the discussion, an application is presented that poses a number of challenges to the objective of providing methodological and tool support for advanced applications in the control domain. The application, under development at the Swiss Federal Institute of Technology in Lausanne (EPFL) under the auspices of the European Union IST project RECSYS, consists of an application involving multiple heterogeneous autonomous robots of two types (Figure 1).

Mamma (RoboX) ***Bambino*** (Smartease)

Fig. 1. *Coordinated autonomous robots.* Mamma robot is intelligent and coordinates multiple, inexpensive Bambino robots which can be sacrificed in dangerous environments.

The *Mamma* is an expensive, intelligent robot with localization capabilities; *Bambini* are inexpensive, expendable devices with limited intelligence and localization capabilities. The mission of the set of robots formed by one *Mamma* and multiple *Bambini* is to explore an indoor environment to find one or more predefined targets. The team of robots moves from one location to another in a coordinated manner led by the *Mamma*. Real-world applications include the clearing of anti-personnel mines.

The application provides an important example of a large category of applications in the robotics area and more generally in the area of autonomous agent-based systems. Such applications are characterized by:

- Independent development of individual components (e.g. *Mamma*, *Bambini*)
- Multiple copies of identical components (e.g. *Bambini*)
- Flexible assembly of components into an overall system design (e.g. *Mamma* plus variable number of *Bambini*)

These characteristics place a number of requirements on both software engineering design methodology and tool:

- The methodology and tool must not force the user to design an entire monolithic system, but rather must allow the *separate* design, architectural modeling, validation, and coding of a single subsystem (e.g. *Mamma*) to be carried out in complete independence from other application components;
- The toolset must make it relatively easy and convenient to clone an application component design (e.g. a *Bambino*) with correctly preserved semantics;
- Finally, the methodology and toolset together must make it relatively straightforward to create, configure, and validate entire application systems from different combinations of components. This requirement is related to a current line of research at EPFL on the design of reusable control components that retain robust real-time characteristics (Pont and Siegwart, 2004).

These requirements effectively bring the challenge of *component-based development* into the control and robotics domain; the first step in meeting this challenge is to amend the deficiencies of the Unified Modeling Language in this regard.

3. EXTENDING THE UML FOR HARD REAL-TIME SYSTEMS DEVELOPMENT

In this section a methodology and tool are described that address the fundamental deficiencies of the UML and associated tool support when applied to demanding real-time systems development, permitting its powerful mechanisms for component-based development to be exploited.

3.1 HRT-UML methodology

In the European aerospace community a reference methodology for design is the Hierarchical Object Oriented Design (HOOD) and in particular its extension for the modeling of hard real time systems, Hard Real Time-Hierarchical Object Oriented Design (HRT-HOOD), recommended by the European Space Agency for the development of on-board systems (Burns and Wellings, 1995). The aerospace experience shows that the design of Hard Real-Time systems needs methodologies suitable for the modeling and analysis of aspects related to time, schedulability and performance.

The HRT-UML method (D'Alessandro, *et al.*, 2002) defines an extension profile of the Unified Modeling Language, which permits the designer to express the concepts and techniques of the HRT-HOOD method in standard UML. It combines the benefits of a mature, well-understood hard real time design method and an internationally recognized object-oriented design standard notation.

Furthermore, HRT-UML provides an integrated methodological solution for the design of complex real-time embedded systems and for their evaluation and verification, according to rigorous techniques such as formal verification, model based dependability evaluation and schedulability analysis.

3.2 HRT-UML design principles

The starting point for an HRT-UML design is not the definition of *classes*, but rather the set of *objects* that compose the system and their interconnections. Therefore an HRT-UML system is built by directly defining objects and links instead of classes and relationships. This means that the first thing an HRT-UML developer has to deal with is the creation of objects in the system space, without any concern for class issues. Classes of objects become an interesting concern only when reuse of objects is foreseen. These same fundamental real-time design principles carry over into the RECSYS toolset.

An HRT-UML project is therefore composed of a set of interacting objects, which are organized in a structure according to several principles:

- **Structural Decomposition**. Objects may be decomposed into other objects, so that the system can be represented as an include graph or parent-child hierarchy of objects;
- **User-Provider Relations**. Objects may use services of other objects, so that the system can be represented as a use graph or user-provider hierarchy of objects;
- **Object Nested Notation**. Objects have a compact notation that includes the operation compartment, allowing users to avoid having to use different diagrams (i.e. class diagrams); and an optional compartment showing the object internals (i.e. the decomposition), which permits users to better understand the system topology;
- **Multiple Static Instances**. Objects may have the same structure and properties but differ in their attribute initialization values and have an independent state evolution. The common structure and properties of such objects are factored into the Underlying Class Model, and an object model, the Prototype Instance, from which a new instance may be cloned. This facility is particularly relevant to the RECSYS case study under implementation by EPFL, where multiple instances of autonomous robotic systems are created.

In summary, in the HRT-UML approach, the "class model," which reflects the constructive approach to designing the system in classical object-oriented methods, is considered as a *background* model, which can be largely derived from the object model in an automatic way. The HRT-UML toolset allows users to build their systems dealing only with objects, automatically creating and maintaining the underlying class model for the purpose of UML metamodel integrity. This is managed in a transparent way for users.

When multiple static instances are needed (as in the case of multiple Bambini for the application involving multiple autonomous robots), the class concept comes into the picture: cloning the prototype instance creates a new instance of a class. The new object may have use-relations in the system with required objects, which are represented as unbound placeholders. They have to be subsequently bound (that is, redirected) to some real object, which is visible in the instantiation environment.

3.3 Real-Time Attributes and Schedulability Analysis

HRT-UML objects and operations have associated real-time attributes, allowing specialized analyses such as schedulability analysis to be performed. The most fundamental attribute is the type of the object itself. An HRT-UML design contains the following types of objects:

- **Passive** - have no control over when invocations of their operations are executed, and do not spontaneously invoke operations in other objects;
- **Active** - may control when invocations of their operations are executed, and may spontaneously invoke operations in other objects. The most general type of object;
- **Protected** - may not have arbitrary synchronization constraints and must be analyzable for the blocking times they impose on their callers;
- **Cyclic** - represent periodic activities;
- **Sporadic** - represent sporadic (in contrast to periodic) activities.

Some further attributes are concerned with mapping timing requirements onto the design (e.g. the deadline). Other attributes have to be set before schedulability analysis can be performed (e.g. the worst case execution time). Other attributes are set after this analysis automatically (e.g. the priority).

For each specified mode of operation, cyclic and sporadic objects have a number of temporal attributes defined:

- The period of execution for each cyclic object;
- The minimum inter-arrival time for each sporadic object;
- Offset time for each cyclic object;
- Deadlines for all sporadic and cyclic activities.

In order to undertake schedulability analysis, the worst-case execution time for each thread and all operations (in all objects) must be known. After the design activity these can be estimated (taking into account the execution environment constraints) and appropriate attributes assigned. The better the estimates, the more accurate the schedulability analysis. Good estimates come from component reuse or from arguments of comparison (with existing components on other projects). During the architecture design activity the designer commits to the run-time scheduling approach – that is, the pre-emptive static priority scheduling with priorities assigned using deadline monotonic scheduling theory.

3.4 HRT-UML and UML 2.0

The most stable version of the UML is 1.4, in effect for several years and the reference version for nearly all methodologies and toolsets on the market. However, a major new version 2.0 has recently been approved for release, whose impact on the evolution of HRT-UML is discussed in this section.

Interestingly, the changes that have a major impact on HRT-UML are not those concerning real-time issues – these issues are still handled primarily (and appropriately) in profiles rather than the official language definition – but those concerning software engineering issues, and in particular issues of component-based development and reuse. Earlier versions of the UML had few (or inadequately defined) facilities for supporting hierarchical structure or component composition. They also contained inadequate mechanisms for specifying the features of components or modules intended to be reusable.

Ports, provided and required interfaces. One of the most significant weaknesses in earlier versions of the UML was the lack of mechanisms, essential to reuse, for specifying both the internal and external environment of an object. HRT-UML introduced the notion of "external placeholder objects" which model the interacting environment of an object. The external placeholders of HRT-UML may now be replaced in UML 2.0 by port-required interface couples. Having different ports – with possibly the same required interfaces, whenever more than one partner object is needed – addresses the issue of topology (in contrast, a required interface only gives the abstract specification of required objects). The HRT-UML notion of binding placeholder objects is equivalent to inter-port connection with compatible interfaces ("required" against "provided").

Structured classes. The new concept of structured classes has been introduced in UML 2.0 to represent classes composed of parts, with an explicit "nested" notation with the purpose of modeling containment hierarchies. This concept yields a more standard form of support for the HRT-UML hierarchical orientation. As noted previously, HRT-UML introduces the notion of "prototype instance" in order to model the internal hierarchical structure of a class. It turns out that this is exactly equivalent to the concept of structured classes, and that therefore a more standard representation in the meta-model and visualization at notational level is possible in mapping onto UML 2.0. Furthermore, it is notable that the port-interfaces, connectors and structured-classes concepts allow for standard modeling of the HRT-UML *delegation* relations between provided interfaces and internal objects.

Components. The component model has been revised in UML 2.0 to make a number of implicit concepts from UML 1.4 explicit and more generally applicable. A component is defined in UML 2.0 as a *modular unit with well-defined interfaces that is physically replaceable within its environment*. As noted earlier, the specification of internal and external interfaces is a necessary precondition for reusability – but it is not sufficient. A reusable component must be constructed as such: its internal and external ports and interfaces must be accurately modeled in order to encapsulate in itself *all* the possible assumptions on the surrounding environment. The component must also be made safe from any sequence of requests from its surrounding environment. This is quite different from designing a class (even a structured one) just for the system under development. UML 2.0 components, inspired by years of experience with software component architectures such as CORBA and COM, bring in the final bit of semantics necessary for true component-oriented development.

Note that, as useful and important as the concepts provided by UML 2.0 are, they are still independent of the major methodological innovations of HRT-UML, such as the focus on objects rather than classes and the mechanisms for cloning in a semantically clean fashion. UML 2.0 will help bring HRT-UML into the mainstream faster, but the contributions of HRT-UML stand on their own.

4. INTEGRATING SOFTWARE ENGINEERING AND CONTROL-ORIENTED TOOLS

The HRT-UML methodology and toolset described above go a long way toward providing the kind of support needed for the development and analysis of demanding real-time systems in the control domain; but alone they are not enough to provide complete support for what has come to be known as *platform-based design* in the control community (Balarin, *et al.*, 1997). This popular approach involves firstly the specification of pure control-related functional properties of the system, followed by mapping to candidate hardware/software architectures, and finally co-simulation of a selected hardware/software architecture in order to verify overall fulfillment of real-time and resource constraints (Liu, *et al.*, 2002).

The most widely disseminated, firmly entrenched tool for specifying functional properties of control systems is Simulink, from the Matlab family of tools. Practical and commercial necessity therefore dictates the provision of a possibility for the HRT-UML toolset to interact with the Simulink tool.

Similarly, co-simulation is generally carried out with the use of highly specialized tools, which often make use of the Matlab environment for their operation. An example of such a tool for co-simulation is TrueTime (Cervin, *et al.*, 2003; Henriksson, *et al.*, 2004). TrueTime is a Matlab-based simulator for real-time control systems developed at Lund University for co-simulation of controller task execution in real-time kernels, network transmissions, and continuous plant dynamics. It provides event-based simulation, real-

time kernel blocks, and appropriate network blocks (e.g. Ethernet, CAN, TDMA, FDMA, Round Robin, and switched Ethernet networks).

Although TrueTime is not a universal, nearly *ad hoc* standard tool like Simulink, it does represent a class of tools that is so specialized that it would be impractical to consider extending the HRT-UML toolset to cover the same functionality. A much better and more flexible solution is to provide for interaction between all three toolsets, each handling its own job in the best way possible.

Figure 2 illustrates the overall design environment provided in this manner.

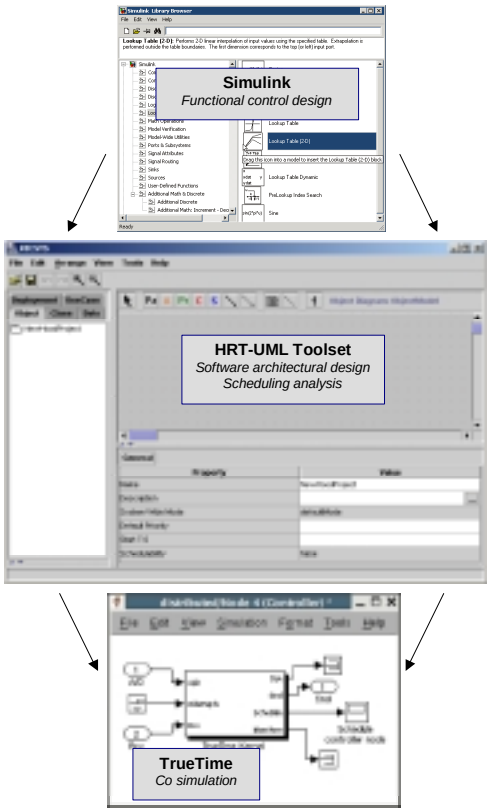

Fig. 2. *Design environment*. Simulink functional control designs are mapped and input to HRT-UML toolset where architectural design and schedulability analysis is performed. Candidate hardware-software architectures are co-simulated with TrueTime.

The interconnection of separate, autonomous toolsets into complete design environments (as opposed to attempts to built enormous, monolithic environments) has become increasingly popular as powerful, expressive information interchange mechanisms have become available. The Extended Markup Language (XML) in particular has become the catalyst for a number of important advances. For example, it has formed the basis for the definition of a standard design interchange format between UML tools; and recently it has been used to represent application family trees for real-time systems (Cechticky, *et al.*, 2004). In this context, XML techniques serve as the basis for import and export between the tools, as described in the following.

4.1 From control design to software design

The schema for the import of the functional control design is illustrated in Figure 3.

The MDL file describing the Simulink control model is parsed and according to a pre-defined DTD translated by the tool into a well-formed XML file containing all the relevant information. Then this XML tree is processed and navigated using the XML library and the corresponding HRT-UML specialized entities are created and made available to the user.

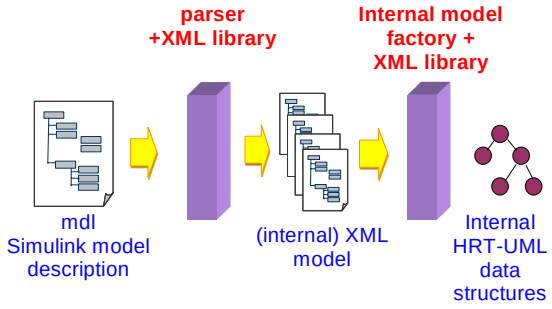

Fig. 3. *Import from Simulink*. Import is based on use of XML standard language constructs.

Note that the first step of the process, from MDL to XMI, has been designed in order to be able to implement with minor effort (i.e. simply providing the right parser) the import of models from other functional modeling environments such as Scilab.

4.2 From software design to co-simulation tool

The automated export process from the design model to the co-simulation tool is illustrated in Figure 4.

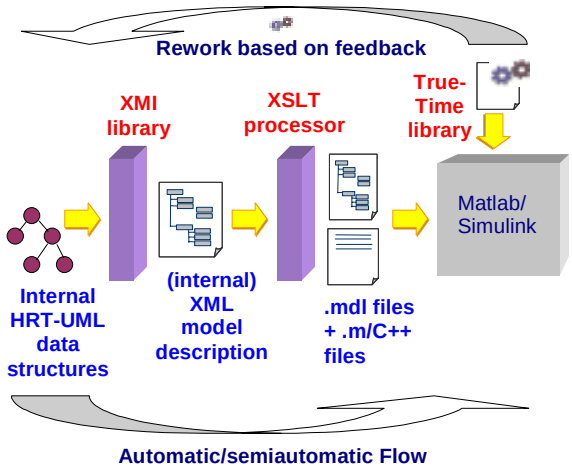

Fig. 4. *Export to co-simulation*. Appropriate files are generated for TrueTime environment.

When the user requests a combined simulation, the following sequence is performed automatically:

- ask the user for the tool's required run time settings;
- the model abstractions of the system that are relevant for simulation are stored in an internal XML format;
- the XML file is then pre-processed by XSLT in order to apply *ad hoc* templates and generate suitable files for the target tools;
- the appropriate tool is run on the resulting files.

As in the case of import, the export mechanisms have been defined to maximize ease of substitution of one co-simulation tool for another.

Figure 4 also gives an indication of the design workflow carried out by the user: as described above, the major software architectural design activity is carried out within the HRT-UML environment, and then co-simulation files are automatically generated. At this point, the workflow effectively moves into the validation activity. If co-simulation reveals that the specified system performance constraints are not met, then the user returns to the HRT-UML environment and reworks the software architecture – for example, selecting a different set of task allocations.

A common objection to this workflow is often heard: why not carry out the entire design workflow directly in the co-simulation tool? Why "waste" time with a separate architectural design in the UML environment? Transferring the entire workflow into the co-simulation tool initially sounds appealing; but it reflects a fundamentally myopic point of view of small, monolithic system development. A co-simulation tool is not a software engineering environment. It provides no facilities addressing the issues raised earlier, such as the specification of reusable components together with required and provided interfaces. How might a *Bambino* specified in a co-simulation tool be incorporated into a completely different system? The co-simulation tool remains silent on that question, whereas the HRT-UML environment is specifically designed to address it. How would entire variants of alternative software architectures be managed and archived in a co-simulation tool? It was never intended to address such issues. Not even a powerful environment like Matlab addresses these important software engineering issues; they require a different perspective.

5. CONCLUSIONS

The number of challenges for the control engineer in developing non-trivial software/hardware systems is significant; some of those challenges are addressed in this paper. But the most significant challenge involves a change in mindset in the control community. It is time to stop viewing systems in the control domain as "control engineering artifacts with a software-related dimension," but rather as "software engineering artifacts with a control-related dimension." Only through this change in perspective will the real obstacles to the development of large-scale, flexible, component-oriented systems in this domain begin to be addressed effectively.

REFERENCES

Balarin, F., M. Chiodo, P. Giusto, H. Hsieh, A. Jurecska, L. Lavagno, C. Passerone, A. Sangiovanni-Vincentelli, E. Sentovich, K. Suzuki, B. Tabbara (1997). *Hardware-Software Co-Design of Embedded Systems: The POLIS Approach*. Kluwer Academic Press, Dordrecht.

Booch, G., I. Jacobsen, J. Rumbaugh (1998). *The Unified Modeling Language User Guide*. Addison-Wesley, Boston.

Burns, A. and A. Wellings (1995). *HRT-HOOD: A Structured Design Method for Hard Real-Time Systems*. Elsevier Science, Amsterdam.

Cechticky, V., A. Pasetti, O. Rohlik, W. Schaufelberger (2004). XML-Based Feature Modelling. In: *Software Reuse: Techniques and Tools* (J. Bosch and C. Krueger (Ed)), pp. 101-114. Springer Verlag, Heidelberg.

Cervin, A., D. Henriksson, B. Lincoln, J. Eker, K-E. Årzén (2003). How Does Control Timing Affect Performance? *IEEE Control Systems Magazine*, **23:3**, pp. 16–30.

D'Alessandro, M., S. Mazzini, M. Di Natale, G. Lipari (2002). HRT-UML: A Design Method for Hard Real-Time Systems based on the UML Notation. In: *Proc. 2002 Conf. On Data Systems in Aerospace*. Dublin, Ireland.

Gamma, E., R. Helm, R. Johnson, J. Vlissides (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Boston.

Henriksson, D., A. Cervin, K-E. Årzén (2004). TrueTime: Real-time Control System Simulation with MATLAB/Simulink. In: *Proc. Nordic MATLAB Conference*. Copenhagen, Denmark.

Liu, J., J. Eker, J. Janneck, E. Lee (2002). Realistic Simulations of Embedded Control Systems. In: *Proc. 15th IFAC World Congress on Automatic Control*. Barcelona, Spain.

Pont, F. and R. Siegwart (2004). Towards Improving Robotic Software Reusability Without Losing Real-Time Capabilities. In: *Proc. First International Conf. on Informatics in Control, Automation and Robotics*. Setubal, Portugal.