

# Position Paper

## “Model-Based Development of Embedded Software Beyond UML”

Bernhard Schätz

TU München, Faculty for Computer Science, Boltzmannstr.3, D-85748 Garching (Munich)  
schaetz@in.tum.de

During the last decades computer science has largely dealt with traditional fields of software development, e.g. business information software applications. Due to its increasing economic relevance, software development for embedded systems is more and more targeted by computer science research. Embedded software systems differ from traditional software systems in several aspects: first, often there is no sophisticated user interface; second, the software controls physical processes connected to the system via sensors and actors; finally, embedded software implements time-critical behaviour treating periodic and non-periodic events. Typically, areas of applications for embedded software are production and automation engineering, aerospace, automotive engineering, or telecommunication.

Originally, embedded software development was driven by programming-centric approaches, where both classes of programming languages used: application-specific (e.g., IEC 1131, CHILL, Protel) programming languages, and general-purpose programming languages (e.g., Assembler, C, Ada) sometimes using restricted sub-languages (e.g., SPARK Ada). Similar to other areas of application, in the domain of embedded software, e.g. in automobile technology, there is an increasing need for more abstract models to be applied in the engineering process. This development is underpinned by the dramatic increase of complexity caused by an ever-increasing amount of functionalities offered by the software as well as the increased demand for inter-networking of embedded components (e.g., app. 50 million lines of object code for ISDN exchange software (1996), app. 10 million lines of source code for software deployed for automotive control in premium cars (2002)). This trend is furthermore boosted by the security and safety requirements of mission-critical software, by an increased demand for reuse and variants induced by shortened product cycles, as well as by distributed software development and supply processes. Depending on the field of application, different modelling approaches were developed, for example synchronous data flow languages or discrete event-controlled machines (cp., e.g., [Lee00]).

Originally, some application domains (e.g. automotive technology, production technology) used software as a replacement for ‘classical’ electronic approaches in control and feedback control engineering (e.g., engine control). Accordingly, tools like Matlab/Simulink or ASCET-SD, and corresponding libraries as the Automotive Block Library by MSR-MEGMA (Manufacturer Supplier Relationship Engineering Graphic Model Exchange) were used, the later including functions like integrators, characteristic

curves, or filters. As a consequence, modelling elements like synchronous function blocks, signals, periods and tasks have come to the fore.

In other application areas (e.g., telecommunication), software was prevalingly deployed to handle discrete, event-controlled tasks (e.g. call switching). Accordingly, tools like Telelogic Tau or ObjectTime were applied. These modelling approaches focus on communicating components, messages, states, and send/receive events. Due to their similarity to distributed applications (e.g., CORBA applications) these concepts were readily applied in modelling business applications. Because of their possibility to describe reactive systems with soft real-time requirements, they were subsumed by the term UML-RT (UML for Real-Time) and finally included in UML (e.g. [OMG04]).

While these two domains of embedded software systems (discrete event-driven systems and synchronous time-driven systems) were hardly linked in the past, they are now growing together. Therefore, software engineering needs a standard modelling approach for both areas of application of embedded software, in order to describe and analyse in combination including their implementation.

Since UML-RT aims at reactive systems more than at embedded systems, a UML profile was developed [OMG03] to include concepts and terms applied in real-time programming (e.g. event, action, resource, schedule). The main focus is the integration of a resource-model into UML, i.e., linking terms like resource, action or duration could be relate to elements of UML like behaviour, state, transition or message. Similar to UML itself, the profile is focused on providing a language for describing resources of embedded systems - independent of concrete notations or tool implementations as much as possible. However, it does not describe the application of the model in the development process.

Providing both a global meta-model for reactive software systems as well as the integration of resource-related language elements established an important precondition to tool-supported treatment of embedded systems. However, some further tasks must be accomplished to provide comprehensive modelling and development approaches. These include:

- to focus on the fundamental modelling concepts for reactive software (e.g., components, channels, events) like in UML-RT, instead of implementational concepts (e.g., object, attribute, method invocation), to stress modelling instead of implementation
- to integrate *domain-specific concepts* (e.g., function, period, execution-time; as well as processor, path of communication, task, schedule) into the general meta-model (e.g., in terms of a profile for embedded systems, instead of domain-specific conceptual models for resources only, as in the RT profile), in order to support different *application-specific* views to embedded software
- to combine these newly introduced domain-specific concepts to tailor-made modelling approaches (e.g., synchronous dataflow functions, time-discrete and event-controlled systems) and to define appropriate and *simple execution models* for these different domain-specific models/levels of abstraction (e.g., abstraction from timing properties for synchronous dataflow, time-synchronous signal-based communication), in order to reduce the complexity of those different views
- to link the introduced approaches to *different levels of abstraction* when modelling distributed, embedded software (e.g., functional architecture with functions and services, logical architecture with clusters of communicating components, technical archi-

texture with control units and communication paths), in order to integrate them into a domain-specific development process (e.g. ITEA EAST/EEA, cp. [FvdB+03]).

- to embed models from different levels of abstraction into a *development process* and to define transitions between these models (e.g., from function blocks with undelayed operations and communication to delayed components). It is necessary to provide such transitions using analytical methods (e.g., consistency between dataflow on function block level and its realization as message exchange on the component level) and generative methods (e.g. calculation of communication schedules for the transition from the logical to the technical architecture) to support a tool-based development process down to the implementation in hard- and software.

Naturally, modelling reactive aspects and capturing resource requirements play an important role in developing embedded software. However, it is only the first step in managing the development difficulty. As a consequence, the partitioning of existing modelling approaches and the development process(es) into domain-specific and manageable pieces is the prerequisite to cope with the increasing complexity of embedded systems.

By identifying domain-specific abstract description techniques, combining them into consistent models of a system under development, and by supporting tool-based analytical and constructive transitions between those models, a model-based development process is established. Model-Driven Architecture [OMG01] suggests a two-layered approach using a platform-independent model (PIM) and a platform-dependent model (PDM). However, to construct software for embedded systems, dealing with aspects like product-lines, integration of safety-critical software, and efficient and time-critical implementation, more refined models are needed. These models -- supporting different views of a system under development -- can be grouped into four different groups, each group supporting a specific architecture:

- **Functional Architecture:** Models of the functional architecture describe the functionalities offered by the system (from the application point of view), including their relations. The functional architecture focuses on the description of those functions independent of their integration (at the level of the logical architecture) to support the identification of variability points and product lines. A very simple model, e.g. function trees as used in Structured Analysis or Use Cases used in the UML [OMG04], only describe the interfaces of those functions and their hierarchical order (generally, in form of a uses-relation) (see, e.g., [FvdB+03]). More detailed models additionally describe their input/out-relation, e.g., synchronous data flow models for periodic, time-triggered functions, or their reactive behavior, e.g., extended state transition diagrams or sequence diagrams to describe sporadic, event-driven functionality (see, e.g., [Lee00]).
- **Logical Architecture:** Models of the logical architecture describe how the system is *structured into logical components* cooperating by (message) communication: the component hierarchy is defined *independent of the implementation platform*. A logical architecture focuses on the integration of all functions of the functional architecture relevant for the system under development by combining several functions into a single component. Typical models of this architecture include structural diagrams describing the interfaces of components and the communication paths (e.g., channels) between them, as well as the behavior of those components, described using, e.g., state-transition diagrams (see, e.g., [BLS+00], [Lyo98]).

- **Platform Architecture:** Models of the platform describe the *available resources for the implementation* of the system under development, i.e. the available sensors and actors, processing units, and communication paths (buses, lines). Depending on the level of detail, the platform can be described in form of a hardware abstraction layer (focusing on the above aspects and ignoring issues like different memory forms, different communication protocols) or by giving the concrete technical platform (addressing issues like RAM, Flash ROM, bus access) (see, e.g., aspects in [OMG03]).
- **Implementation Architecture:** Models of the implementation architecture describe the structure of the actual implementation of the system under development. The implementation architecture focuses on the definition of an implementation of the logical architecture in terms of the platform architecture. These models include the *description of the structure of the system* (i.e., modules or tasks including their interfaces) as well as their *dynamic aspects* (i.e., task schedules, bus schedules) (see, e.g., aspects in [OMG03]). Furthermore, the implementation architecture includes models to describe *the mapping between the structure of the implementation and the implementation platform* (e.g., by assigning computation tasks to processing units).

To increase the quality and the efficiency of the development process, model-based development is inherently linked with a CASE-based development process. While being far from the maturity of commercial tools, with AutoFOCUS/Quest ([BLS+00], [SPH+02]) or Ptolemy [HLL+03], prototypes have been and are developed demonstrating the applicability of the approach.

## References

- [BLS+00] P. Braun, H.Lötzbeyer, B.Schätz, O.Slotosch. *Consistent Integration of Formal Methods*. IN: Tool and Algorithms for the Construction and Analysis of Systems (TACAS 2000). Susanne Graf, Michael Schwartzbach (eds.) Springer Verlag
- [FvdB+03] Freund, U. von der Beeck, M. Braun, P. Rappl, M. *Architecture Centric Modeling of Automotive Control Software*. SAE 2003-01-0856, Detroit 2003.
- [Lee00] Lee, E. *What's Ahead for Embedded Software*. IEEE Computer, 2000.
- [Lyo98] Andrew Lyons. *UML for Real-Time Overview*. Whitepaper, www.rational.com, 1998.
- [OMG01] Object Management Group. *Model Driven Architecture (MDA)* Version 1.0. www.uml.org, 2001.
- [OMG03] Object Management Group. *UML<sup>TM</sup> Profile for Schedulability, Performance, and Time Specification*. Version 1.0 formal/03-09-01. www.uml.org, 2003.
- [OMG04] Object Management Group. *Unified Modeling Language: Superstructure*. Version 2.0. OMG Adopted Specification ptc/03-08-02. www.uml.org, 2004.
- [SPH+02] B.Schätz, A. Pretschner, F.Huber., J. Phillips. *Model Based Development*. Technical Report. TUMI-0402. Technische Universität München. 2002.
- [HLL+03] Ch. Hylands, E. Lee, J. Liu, X. Liu, S. Neuendorffer, Y. Xiong, H. Zheng. *Introduction to Ptolemy II*. University of California, Berkeley. 2003.