White Paper

Model-Based Software and Systems Development

Bernhard Schätz, Manfred Broy, Franz Huber, Jan Philipps, Wolfgang Prenninger, Alexander Pretschner, Bernhard Rumpe

TU München, Faculty for Computer Science, Boltzmannstr.3, D-85748 Garching (Munich)
{schaetz|broy|huberf|philipps|prennige}@in.tum.de

ETH Zurich, Departement Informatik, Haldeneggsteig 4, CH-8092 Zürich
pretscha@inf.ethz.ch

TU Braunschweig, Informatikzentrum, Mühlenpfordtstr. 23, D-38106 Braunschweig
rumpe@tu-bs.de

The construction of reliable (embedded) software can be significantly improved using explicit model-based tool-supported development process. Research results suggest that such a process can contribute essentially to increase the quality of the developed product as well as to better efficiency of the development itself:

- [Jon91] shows that more than 50 % of serious errors are made during design (25 % during implementation); about 30 % of medium class errors are made during design (30 % during implementation)
- [Jon91] shows that analytical techniques performed on early-phase description of the product (e.g., structured approaches, design reviews) require generally at least less than 50 % of the effort in both error detection and correction needed for later-phase techniques (e.g., integration test, field test)
- [Jon91] shows that those analytical techniques of the early phases are at least twice as effective to detect errors of the early phases than those later-phase techniques.
- [BMJ+96] shows that especially in a development process requiring a high level of product quality, CASE support can significantly increase productivity.

Therefore, in order to improve both the *efficiency of the development process* (by increasing the degree of mechanization), and the *quality of the development product* (by decreasing the amount of possible defects), support for a model-based development for embedded systems should go beyond the following forms of methodological and CASE tool support (see also [SHH+03]):

- **Using an implementation-level model:** Modeling the system at implementation level rather than at a more abstract level leads to a limited development process, focusing on the implementation and integration phase. Thus, e.g., defect analysis is limited to implementation level defects. This excludes simple defects like message interface incompatibility between processes executed on different nodes since those messages are described as a byte-block oriented bus-protocol. Furthermore, due to the gap between the earlier phases and the implementation level, even design specifications are not always related to the implementation. This is often leading to inconsistencies between the design and the implementation.
- **Using an OO-model for embedded software:** OO-based approaches supply different views of the system including non-executable views supporting early phases (e.g., use case description, interaction scenarios) making consistency analysis available in those phases. However, those views offer only limited abstraction from the OO operational model; e.g., (synchronous) method calls are used to model interaction between tasks rather than the more suitable message- or signal-based communication. Furthermore, additional domain-specific aspects (e.g., bus schedules, task switching) are not modeled explicit. Therefore - to obtain deployable code - those aspects are laid off to the coding phase outside the modeling capability of the tool, leading to similar problems as with implementation-level models.
- **Using a Draw-and-Generate Tool:** Domain-specific approaches generally support a more detailed model including aspects like preemption or time-driven communication allocatable to bus slots; the tools make use of this information to generate deployable code. However, sophisticated analysis techniques are not available on the level of the model; therefore, the analysis of defects introduced on the level of the model is performed manually or is delayed until the execution of the generated model. Thus, e.g., the tool does not ensure the consistency between time-driven communication and allocated bus schedules.
- **Using a loosely coupled tool chain:** A loosely coupled tool chain generally splits the development process in tool-related phases with substantial gaps between those phases; often a high degree of integration is missing. Examples for those gaps in the development process are scenario-based descriptions of the behavior that

cannot be used to generate equivalent test cases on the implementation level, or counter-examples generated by verification or validation on the level of implementation, which are not expressed on the level of the abstract system. Therefore, the tool chain depends heavily on those forward and backward integrations and the common model bridging the tool chain. Since those steps are limited by the information explicitly represented in model of each tool of the chain, the degree of coupling is limited by the quality of those models and their interdependence. If, e.g., the bus signal communicated on the implementation level cannot be related back to messages communicated between abstract components, counter-examples cannot be expressed on that abstract level.

In contrast, for optimal support, a model-based software development process fulfills the following properties:

- **Adequate Models:** During the development process, different aspects of the system under development are addressed (for the domain of embedded systems, e.g., overall functionality, time and resource limitation, partitioning and deployment, scheduling) during different phases. Specific models are available for the different phases (e.g., requirements analysis, design, implementation, integration) of the development process. Since software systems, and especially embedded software systems, are moving from monolithic single-functionality programs to distributed, interactive multi-functionality networks, a central aspect of these models is treatment of interaction and communication as well as time-related aspects. Furthermore, models must support specific aspects needed for the application domain (for the domain of embedded real-time systems, e.g., notions like messages/signals, task, schedule, controller, bus). By supporting domain- and application specific modeling elements, model information about the system under development becomes available, leading to stronger analysis and generation techniques (e.g, in the domain of embedded systems, checking the worst case time bounds of a task, or generating a bus schedule from the communication description of a component model).
- **Abstract Models:** Models should contain only those aspects needed to support the development phase they are applied for (e.g., modeling the interaction between components by messages rather then method calls to the bus driver or the operating system). By abstraction models reduce the complexity of description as much as possible as well as the possibility to produce faulty descriptions (e.g., ensuring type correctness between communication ports rather than using byte block messages on the level of bus communication). Furthermore, abstract models also support descriptions of the system under development in early analysis and design phases, making analysis and generation support available even at early stages of the development process (e.g., completion of state-based description of a component to introduce standard behavior for undefined situations).
- **Integration by Analysis:** The relation between different models during the development is supported. This includes the analysis of different models used during the same phase (e.g., checking the consistency of a scenario and a complete behavioral description of a component) as well as different phases (e.g., checking a bus communication schedule against the abstract communication behavior of the corresponding abstract component). This leads to a higher level of product quality, especially by supporting analysis of the system at earlier stages; additionally, it also increases process efficiency by supporting earlier detection of defects.
- **Integration by Generation:** The transition between different models in the development process is supported by generating models out of each other. This includes forward generation (i.e., from models of an earlier phase to a latter phase; e.g., the generation of test cases from a behavioral description) as well as backward generation (i.e., from a latter model to an earlier one; e.g., the generation of an abstract scenario-based description from an execution trace of the implementation level). This leads to increased process efficiency by a higher degree of automation as well as increased product quality by eliminating defects introduced by manual development steps.

For the development of (embedded) software, several models with different degrees of abstraction are used. 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 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).

In a model-based development process, each architectural level has an associated collection of description techniques (e.g., data flow diagrams, system structure diagrams, platform diagrams bus schedule tables), conceptual models integrating the concepts of those description techniques, as well as semantical models to support the above-mentioned properties.

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 [HSE97], Quest [BLS+00], and the extended functionality of AutoFOCUS2 [SPH+02] 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

[BMJ+96] T. Bruckhaus, N. Madhavji, I. Jannsen, J. Henshaw. *The Impact of Tools on Software Productivity*. IEEE Software, 9. 1996.

[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.

[HSE97] F. Huber, B. Schätz, G. Einert. *Consistent Graphical Specification of Distributed Systems*. In: FME '97: 4th International Symposium of Formal Methods Europe, Lecture Notes in Computer Science 1313, pp. 122 - 141, John Fitzgerald, Cliff B. Jones, Peter Lucas (ed.), Springer.

[Jon91] C. Jones. *Applied Software Measurement*. McGraw-Hill, 1991.

[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.

[OMG03] Object Management Group. UMLTM 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.

[SHH+03] B. Schätz, T. Hain, F. Houdek, W.Prenninger, M. Rappl, J. Romberg, O. Slotosch, M Strecker, A. Wisspeintner. *CASE-Tools for Embedded Systems*. TUM-I0309. Technical Report, TU München, 2003.