# Research Challenges in Embedded and Hybrid Systems

Insup Lee and Oleg Sokolsky
Department of Computer and Information Science
University of Pennsylvania

## 1. Introduction

An embedded system consists of a collection of components that interact with each other and with their environment through sensors and actuators. Many embedded systems are part of safety-critical applications, e.g., avionic systems, manufacturing, automotive controllers, and medical devices. There are several factors that complicate the design and implementation of embedded systems. First, the software complexity of embedded systems has been increasing steadily as microprocessors become more powerful. To mitigate the development cost of software, embedded systems are being designed to flexibly adapt to different environments. The requirements for increased functionality and adaptability make the development of embedded software complex and error-prone. Second, tight resource constraints distinguish embedded systems from most other computer-based systems. The need to satisfy the system requirements and still stay within the resource constraints makes the design space more complicated. Multi-dimensional trade-offs between different resources are hard to capture within exiting design technologies. Third, embedded systems are increasingly networked to improve functionality, reliability and maintainability. Networking makes embedded software even more difficult to develop, since composition and abstraction principles are poorly understood.

Embedded software systems have been developed traditionally in an ad-hoc manner by practicing engineers and programmers. Knowledge of system behavior and functionality is contained in the mind of the domain-expert designer, and is only imperfectly captured and translated into system products by the engineer and programmer. Design based on mathematical modeling plays a critical role in other engineering disciplines, such as structural engineering. Although much progress in developing formalisms and tools based on them have been made during the last three decades, their use has been limited by the expense (i.e., time, computational resources, and expertise) required to employ them. The complexity of embedded software systems has increased to a point where it is not possible to envision the development of high-quality embedded software systems without using such techniques in the future.

Embedded systems are being developed by control engineers and software engineers. Traditionally, control engineers view the problem as designing of robust controllers using continuous dynamics, whereas software engineers have a discrete view of the problem. Their approach is to abstract physical characteristics of the environment and to treat the system as reactive systems. Hybrid system modeling combines these two approaches and is believed to be natural for specification of embedded systems. In hybrid systems, continuous evolutions are specified by sets of differential and also possibly by algebraic constraints, while discrete changes in an execution are captured by transitions that jump from one set of continuous constraints to another. Continuous dynamic evolve over time, whereas discrete transitions are instantaneous while changing model variables during the jump. As larger embedded systems are being modeled, modeling languages are designed with advanced features from the programming language domain. For example, modular modeling techniques such as hierarchy, concurrency, instantiation, and scoping are used to structure the model in a more intuitive way and make it mode compact. Coupled with modular modeling are compositional semantic definitions that enable modular analysis of models, making verification problems easier to tackle.

*Example application area.* As an illustration of the importance of the embedded and hybrid systems area, consider the medical industry. The medical industry is especially significant because it represents 20 cents of every dollar spent in the United States. Software is involved in the management and production of medical products, as well as in the products themselves. Medical devices with embedded processors are safety-critical systems requiring high reliability and correctness with respect to the requirements. Typically, such a device consists of sensors that monitor the human body (for

example, glucose level or blood pressure), software that makes decisions based on these readings, and actuators that execute the corrective action (for example, release of insulin or other drugs). The high-level design of the control algorithm needs to account for the the constantly changing human body. Complex continuous models are used to represent such body systems as heart or kidneys. Development of dependable medical devices can benefit from integrated hybrid models of the software and its environment. Furthermore, the medical industry needs certification methods that are scientifically well-founded and also practical for users with a diverse set of skill levels. As technology evolves, medical devices are becoming more complex and autonomous and are increasingly being built from Off-the-Shelf software components. In order for the Center for Devices and Radiological Health (CDRH), within the FDA, to perform its regulatory function, credible and timely evidence is needed for assurance that the device software will perform as intended [9].

There are several hybrid modeling techniques that have been developed in recent years (e.g., CHARON [1], CheckMate [2], Masaccio [8], Modelica [5], SHIFT [3], Simulink/Stateflow [14], etc.). With the hybrid modeling approach, it is possible to model faithfully both the system and its environment. This is important since, in an embedded system, computation by software components is to react to the continuous evolution of physical components and system environment. It is the interplay between continuous and discrete behaviors that are critical in the correct functioning of an embedded system. There are many challenges to elevate the use of hybrid system models as common practice of embedded systems development such as medical devices. In the rest of the paper, these challenges are grouped and discussed under modeling and analysis, implementation and validation, and certification.

## 2. Modeling Challenges

An embedded system can be modeled on a variety of levels and using a plethora of formalisms. Some visual formalisms, such as Statecharts [6], are targeted at easy comprehension of the model by the users. Other formalisms may facilitate analysis of the model, sometimes at the expense of readability. Yet others may be suitable for code generation, often at the expense of formal sophistication. It is commonly believed that no single formalism will solve all the modeling problems. Different modeling assumptions are suitable for each modeling task. For example, hybrid systems formalisms often used in behavioral analysis of embedded systems (for example, CHARON [1]), abstract away computation time, assuming that sensor readings, actuation, mode switches, etc., happen instantaneously. This assumption greatly simplifies analysis, but may make the model physically unrealizable. Furthermore, a hybrid system model usually models both the embedded system and its environment. Such a model, when used for code generation, has to "forget" the environment part. On the other hand, a formalism for schedulability analysis, such as ACSR [10], does not have to model details of computation, instead capturing only computation times and periodicity of task executions. Such a model, of course, cannot be used, for example, in code generation.

*Eliciting formal models from informal requirements.* The development of most systems starts with informal requirements that specify how the system (consisting of hardware and software) and the user or environment are expected to behave and their interactions. Research is needed to facilitate the elicitation of design models from such informal requirements (with appropriate NLP techniques and tools to facilitate the process). It is important for the elicitation process to be intuitive for the domain engineers of embedded systems. Furthermore, the resulting requirements specification should be amenable to analysis and refinement during subsequent stages of the design process. Balancing these aspects is the first challenging problem since the goals are often contradictory. For example, formalisms that make the analysis easier are usually far removed from the problem domain. Furthermore, requirements start as informal description but need to be formalized to support analysis. Another challenging problem is how to facilitate the translation from informal description to formal specifications and to capture assumptions made by domain experts in informal requirements.

*Model validation.* The model must be verified and validated to ensure it is correct and consistent with its intended purposes. Such analysis can be done using model checking and simulation on design specifications. However, tools supporting these techniques need to be improved to handle complex embedded systems. Furthermore, there is much work done on how to validate that models are indeed the ones that are intended. Also, it should be possible to use the same models to validate an implementation. The requirements, design, and implementation need to be related such that if one can show that the design model satisfies the requirements and the implementation conforms to the design, then the implementation meets the requirements.

*Multiple uses of modeling artifacts.* One of the reasons why formal methods techniques are not used in practice is that formal models are only used at the design phase, but become useless down the development process. We need to explore ways to reuse the various design artifacts during the other development phases, such as coding and testing phases. The potential promising areas include the use of models for automatic test generation and code generation, as well as the use of hybrid models validated at the design time for run-time verification. We need to explore how to ensure that investments on hybrid systems models and analysis pay off directly in the final product development.

*Sharing of modeling artifacts.* We believe that it is easier (in theory) to share models than code. This is because models are at higher level of abstraction than code, and thus, tied less with target platforms. There has not been much support and effort with open model (*a la* open source) development so far. Such an endeavor should help to elevate model-based development into main-stream activities.

*Composable models.* The notion of composition is necessary to deal effectively with designing of large complex systems. Thus, design models should be composable to facilitate reuse and sharing, as well as just to be able to put together a complex system using simpler components. The composition of design can be either for homogeneous models or for heterogeneous models. Traditionally, formal method research has concentrated in composing specifications in the same modeling language or paradigm. To elevate the modeling to a high level, it is important that we start investigating how to compose models with different purposes (e.g., one design model for the physical layout of a sensor network and another design model for the protocol used between sensor nodes) to determine interaction between different views and to understand the overall design.

*Certification based on model.* To be able to evaluate the quality of embedded systems, we need to develop a certification process based on sound scientific foundations. The certification can be done in two steps: first certify that a design has the right properties, and then, certify that an implementation conforms to the design. With proper scientific foundations, it should be possible to measure quantitatively how well a system meets its requirements.

## 3. Implementation and Validation Challenges

A model-based approach is an emerging paradigm for developing robust software, and has been the focus of increasing research effort. Models are used during the design phase to ensure systems under consideration have desired properties. Benefits of high-level modeling can be significantly improved if models can be used for code generation as well as the validation of implementation.

*Code generation.* One promising approach is to generate code from models. However, precise translation from hybrid system models to code is difficult because there is a gap between the platform-independent semantics of a hybrid model and the implementation of the model. For example, models are defined in the continuous-time domain whereas code executes on computers with discrete time. A common approach is to associate a model with a sampling rate before code generation, and rely on an approximate algorithm that computes the next state numerically. Depending on the choice of the sampling rate and the algorithm, the behavior of the code may vary significantly due to numerical errors, different sampling rates, and real-time scheduling. In the worst case, discrepancy between behaviors at model-level and code-level could render analysis results at the model level meaningless for implementation since implementation may exhibit behaviors that are not possible in the model. The problem is compounded if components are to be executed in distributed systems due to communication and synchronization delays. Since the correspondence between the model and the generated and synthesized code is rarely formalized, it makes difficult to reason about the discrepancy between the model and the code. Bridging this gap is a research challenge.

*Implementation validation with respect to models.* There is a gap between design and implementation since the implementation contains a lot more details than the design. In particular, there is no guarantee that an implementation is consistent with a design model unless the implementation is derived automatically from the design model. Since it is not yet possible to completely generate an implementation from design, it is important to ensure the implementation is consistent with the design model. There are two ways to use a design model to validate an implementation: model-based testing and model-based run-time monitoring and checking. The former is to generate a test suite from a design model and then apply tests to an implementation. The latter is to observe the execution of a running system and ensure that its run-time behavior is consistent with those described in the design specification, and is also known as run-time verification.

*Test generation from hybrid system models.* Testing is the most widely used validation technique in practice. We can improve the rigor of testing-based validation by using system models as the source of test cases. For embedded systems, testing is usually performed in a controlled environment; therefore, the most natural medium for test generation is a hybrid system model, capturing the expected behavior of the environment as well as the prescribed behavior of the implementation.

There are several challenges for automatic generation of tests from hybrid system models: 1) Observability and controllability for the continuous variables: A system-under-test may not have adequate API to allow the tester to control and observe behaviors. It is sometimes necessary to insert test code to support testing, which in turn can affect the system behavior. 2) Approximating reachable regions by abstraction: A test is a trajectory through the reachable state space of a hybrid system model. The size of the reachable state space is usually too large, and thus, it is necessary to group states into regions by abstraction techniques. 3) Identifying test coverage criteria: Tests are generated by applying test criteria. For example, state and transition coverage criteria are used for the control-flow model of the program, whereas definition and use coverage criteria are used for the data-flow model of the program. Coverage of continuously changing inputs needs to be studied better.

*Run-time verification of embedded systems.* Run-time verification is a novel assurance technology for computer-based systems that has gained significant attention in recent years. The underlying premise of the run-time verification approach is that correctness of a system cannot be fully guaranteed by design-time methods. On the one hand, complexity of real-world systems will always exceed capabilities of analysis tools. On the other hand, design-time analysis is usually applied to system models, and there is always a possibility of a mismatch between the system model and its implementation. Unpredictable environments make design-time analysis even more complex for embedded systems.

Run-time verification offers the possibility of checking the system implementation that is operating on real inputs from its environment. This eliminates all effects of model mismatches, since no model is involved in the analysis process. Several methodologies for run-time verification have emerged recently, most notably [4, 7, 11]. Most of these approaches target single-processor systems and are not suitable, in their present form, for checking of networked embedded systems. Furthermore, run-time validation of continuous trajectories has not been systematically addressed.

A major concern for run-time verification of embedded systems is monitoring overhead. Most run-time verification approaches rely on instrumentation of the system code to extract observations. Tight real-time constraints found in some embedded systems may not allow for extensive instrumentation. The overhead of instrumentation can make the system violate its timing constraints or even make it unschedulable. The challenge is to reliably predict the effect of run-time verification on the system performance and find systematic ways to reduce overhead.

## 4. Certification Challenges

Safety-critical computer systems must inevitably meet certain government regulatory requirements. Examples in the United States are regulation of medical devices by the FDA, regulation of nuclear power plants by the DOE, and regulation of civil aircraft by the FAA. Regulatory authorities require a comprehensive system safety program, which includes as one component design assurance (assuring the absence of design defects, where a software defect is considered a design defect in this context). Strictly speaking, the FAA (for example) does not legally mandate specific technical means for demonstrating design correctness; to do so might limit both innovation and manufacturer liability. However, civil aircraft certification may be streamlined by demonstrating compliance with FAA-approved guidelines, the two most relevant to our proposal being RTCA/DO-178B for software and RTCA/DO-254 for digital hardware[12, 13]. These guidelines specify planning and development and supporting process activities that must be conducted (e.g., requirements, design, verification, configuration management); together with specific requirements, methods, work products, and deliverables for those activities. Verification at high levels of assurance must include multiple independent means of verification, as well as review, analysis, and testing activities. There is a great deal of flexibility in meeting guideline requirements, and in practice each development organization has its own specific process that must be shown to be compliant with the guidelines.

A long-known shortcoming of current development practices is the use of informal, largely textual top-level specifications. Such specifications typically contain many ambiguities, do not completely cover all possible situations, and often contain internal inconsistencies. However, extensive formalization has proven difficult, largely because the available formal notations are not simultaneously comprehensible (easily written and understood by typical engineers), concise (descriptions can be short and rely on common domain knowl-

edge and notations), and comprehensive (all aspects of a system can be specified in the notation) [15].

Another major shortcoming of current development practices is the large amount of error-prone manual effort spent deriving various specifications from each other. More formal behavioral specifications may be derived from the informal top-level specification, but this involves additional work, including additional verification activities to show that the derived formal specification is consistent with the original informal specification. Any particular formal specification captures only a portion of the original informal specifications, so one must carefully track which top-level specifications are captured in which derived formal specifications, and which top-level specifications are not formalized in any way. Test specifications are similarly typically produced by hand, with only review (i.e., human judgment) to ascertain whether the original specifications have been correctly and completely captured in the test specifications. Automatic execution of tests from test specifications is common in practice, but initial test results are typically checked by hand, or compared with an oracle (e.g., a system simulation) that was itself developed largely by hand at great expense with many defects. For this reason, certification guidelines require extensive effort to establish traceability and consistency between the various work products. However, there is little technology for rigorous and automated traceability and consistency checking other than the use of a database of relationships that have been entered by hand [15].

Formal modeling techniques have great promise of alleviating the above shortcomings; however, much research is needed to realize this promise. Also, traceability considerations need to be built into the formalism rather than be an external concern.

## 5. Conclusions

There are many advantages for using hybrid system models during the development of embedded systems. To benefit from them, however, many challenges including those described in this paper need to be addressed.

## References


[1] R. Alur, T. Dang, J. Esposito, R. Fierro, Y. Hur, F. Ivancic, V. Kumar, I. Lee, P. Mishra, G. Pappas, and O. Sokolsky. Hierarchical modeling and analysis of embedded systems. *Proceedings of the IEEE*, 90(1):11–28, Jan. 2003.

[2] A. Chutinan and B. Krogh. Verification of polyhedral-invariant hybrid automata using polygonal flow pipe approximations. In *Hybrid Systems: Computation and Control, Second International Workshop*, LNCS 1569, pages 76–90, 1999.

[3] A. Deshpande, A. Gollu, and L. Semenzato. The shift programming language and run-time system for dynamic networks of hybrid automata. Technical Report UCB-ITS-PRR-97-7, University of California at Berkeley, 1997.

[4] D. Drusinsky. The Temporal Rover and the ATG Rover. In *Proceedings of 7th International SPIN Workshop, LNCS 1885*, volume 1885, pages 323–329, 2000.

[5] H. Elmqvist, S. E. Mattsson, and M. Otter. Modelica – The new object-oriented modeling langugae. In *Proceedings of the 12th European Simulation Multiconference*, pages 127–131, 1998.

[6] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8:231–274, 1987.

[7] K. Havelund and G. Rosu. Monitoring Java Programs with JavaPathExplorer. In *Proceedigns of the Workshop on Runtime Verification*, volume 55 of *Electronic Notes in Theoretical Computer Science*, 2001.

[8] T. Henzinger. Masaccio: A formal model for embedded components. In *Proceedings of IFIP International Conference on Theoretical Computer Science*, pages 549–563, 2000.

[9] P. Jones. Personal Communication, March 24 2003.

[10] I. Lee, P. Brémond-Grégoire, and R. Gerber. A Process Algebraic Approach to the Specification and Analysis of Resource-Bound Real-Time Systems. *Proceedings of the IEEE*, pages 158–171, Jan 1994.

[11] I. Lee, S. Kannan, M. Kim, O. Sokolsky, and M.Viswanathan. Runtime assurance based on formal specifications. In *Proceedings of the Int. Conf. on Parallel and Distributed Processing Techniques and Applications - PDPTA'99*, June 1999.

[12] Radio Technical Committee for Aeronautics. RTCA/DO-178B, Software Considerations in Airborne Systems and Equipment Certification, December 1992.

[13] Radio Technical Committee for Aeronautics. RTCA/DO-254, Design Assurance Guidance for Airborne Electronic Hardware, April 2000.

[14] The Mathworks Inc. http://www.mathworks.com.

[15] S. Vestal. Personal Communication, Nov 2001.