Model-Based Design of Embedded Software*

Rajeev Alur
Department of Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19103
Email: alur@cis.upenn.edu
URL: www.cis.upenn.edu/~alur/

October 31, 2001

Abstract

Traditional control engineering provides mathematical tools for designing control laws for dynamical systems. Programming languages and software engineering, while rich in techniques for layering and structuring of complex software with multiple features and concurrency, abstract from real-time and dynamics. As software embedded in physical devices gets complex and distributed, the emerging discipline of *hybrid systems* that combines control engineering and software science, can provide the foundation for systematic design. This paper outlines promising research directions for modeling and analysis of embedded systems for improved design automation and increased safety.

1 Introduction

An embedded system typically consists of a collection of digital programs that interact with each other and with an analog environment. Examples of embedded systems include manufacturing controllers, automotive controllers, engine controllers, avionic systems, medical devices, micro-electromechanical systems, and robots. As computing tasks performed by embedded devices become more sophisticated, the need for a sound discipline for writing embedded software becomes more apparent (c.f. [Lee00, HK01]). Model-based design paradigm, with its promise for greater design automation and formal guarantees of reliability, is particularly attractive given the following trends.

Software Design Notations. Modern object-oriented design paradigms such as *Unified Modeling Language* (UML) allow specification of the architecture and control at high levels of abstraction in a modular fashion, and bear great promise as a solution to managing the complexity at all stages of the software design cycle [BJR97]. There are emerging tools such as RationalRose (see www.rational.com) that support modeling, simulation, and code generation, and are increasingly becoming popular in domains such as automotive software and avionics.

Control Engineering. Traditionally control engineers have used tools for continuous differential equations such as MATLAB (see www.mathworks.com) and MATRIXX for modeling of the plant behavior, for deriving and optimizing control laws, and for validating functionality and performance of the model through analysis and simulation. This design methodology has strong mathematical foundations, and leads to robust designs of systems such as cruise control and throttle control, where the primary complexity is in the continuous dynamics. Tools such as SIMULINK recently augmented the continuous modeling with state-machine-based modeling of discrete control.

*Research reported in this paper is based on join projects with colleagues in the Hybrid Systems Group at University of Pennsylvania. Our research on embedded systems is supported by NSF award CCR99-70925, SRC award 99-TJ-688, DARPA ITO Mobies award F33615-00-C-1707, NSF CAREER award CCR97-34115, NSF ITR award, and ARO MURI award.

Formal Specification and Verification. Formal methods for rigorous specification and verification of correctness requirements have witnessed increased interest and acceptance due to a shift in emphasis towards lightweight methods that can be automated and used for debugging at early stages of design [CW96, JR00]. In particular, model checking is emerging as an effective technique for debugging of high-level models (see [CK96] for a survey). Model checkers such as SMV [McM93] and SPIN [Hol97] have been successful in revealing subtle errors in cache coherency protocols in multiprocessors and communication protocols in computer networks. There are also emerging techniques for extracting models from code using abstraction. Tools such as Bandera [CDH$^+$00] and SLAM [BR00] have applied predicate abstraction for analysis of C or Java programs. In recent years, the model checking paradigm has been successfully extended to models with continuous variables [AD94, AHH96] leading to tools such as UPPAAL [LPY97] and HyTech [HHW97].

These trends motivate research in formal modeling and analysis of embedded software systems. We outline a few research directions of main interest to us in the following section. The ultimate goal is develop tools that will allow designers to develop embedded systems with interacting multi-modal components with the same rigor as applied in design of individual controllers. Such tools will dramatically reduce the cost of testing and maintenance, and also assure increased reliability required in safety-critical applications.

2 Research Themes

2.1 Hybrid Modeling

Traditionally, control theory and related engineering disciplines, have addressed the problem of designing robust control laws to ensure optimal performance of processes with continuous dynamics. This approach to system design largely ignores the problem of implementing control laws as a piece of software and issues related to concurrency and communication. Computer science and software engineering, on the other hand, have an entirely discrete view of the world, which abstracts from the physical characteristics of the environment to which the software is reacting to, and is typically unable to guarantee safety and/or performance of the embedded device as a whole. An embedded system consisting of sensors, actuators, plant, and control software is best viewed as a *hybrid* system. There has been a lot of recent research on modeling, control, and verification of hybrid systems (see [VvS99, LK00] for proceedings of the annual workshop). The relevance of hybrid modeling has been demonstrated in various applications such as coordinating robot systems, engine control, collision avoidance for autonomous air-vehicles, chemical process control systems, animations, and biomolecular networks.

An important research issue in hybrid modeling of embedded software is identifying the appropriate levels of abstractions that can be used to relate the low-level control viewpoint with the high-level software viewpoint. As an example, consider programming of autonomous mobile robots, say, programming Sony's AIBO robots to play soccer (see `www.aibo.com`). Existing methodology focuses on relatively low-level control commands such as "go-to-ball" whereas a software designer needs to focus on high-level game-playing strategies, switching between defensive and offensive modes, and communication to coordinate different players. The high-level model still needs some abstract view of the dynamics of motion, but does not have to worry about the control inputs to individual joints. Thus, designers of control laws as well as designers of game strategies need "hybrid" models, but different ones. The whole design can be greatly facilitated by a clear understanding of *layering* of hybrid models.

2.2 Modeling Languages

Embedded software is currently written in low-level languages. Tools used by control engineers allow block-structured, hierarchical, visual design, but do not reflect the state-of-the-art programming language concepts such as type systems, encapsulation, object-oriented designs, interfaces etc. Even the early models for hybrid systems such as phase transition systems [MMP91] and hybrid automata [AHH96] were primarily concerned with formal semantics, and showed little structure. Programming languages, with the notable exception of the synchronous programming languages such as Esterel [BG88], do not adequately deal with time as a first-class entity. Modeling languages for embedded software can borrow nicely from both traditions.

2.3 Behavioral Interfaces

The notion of interfaces in object-oriented languages is typically based on static types. Since control engineers are used to dealing with mathematical functions that capture continuous dynamics, components in embedded systems should have behavioral interfaces that capture static types as well as the hybrid dynamics of the interaction of the component with the environment (see, for instance, the PTOLEMY project [Lee00]). Such behavioral interfaces can be built for languages that support compositional formal semantics. Formal semantics leads to definitions of *semantic* equivalence (or refinement) of specifications based on their observable behaviors, and compositional means that semantics of a component can be constructed from the semantics of its subcomponents. Such formal compositional semantics is a cornerstone of concurrency frameworks such as CCS [Mil80], and is a prerequisite for developing modular reasoning principles such as compositional model checking and systematic design principles such as stepwise refinement. Besides the known difficulties in defining semantics for languages with rich constructs such as inheritance and exceptions, the global nature of time makes it challenging to define semantics of hybrid components in a modular fashion.

2.4 Automated Analysis

The greatest value of model-based design is the opportunity to subject models to useful analysis in early stages of design to reveal potential errors. A classical and useful way of analysis is simulation. If we couple hybrid dynamics with stochastic behavior, developing reliable simulation tools itself is an interesting research problem. We are particularly interested in exploring the potential of model checking for powerful debugging of embedded software. The state-of-the-art computational tools for model checking of hybrid systems are of two kinds. Tools such as UPPAAL [LPY97] and HYTECH [HHW97] limit the continuous dynamics to simple abstractions such as rectangular inclusions (e.g. $\dot{x} \in [1, 2]$), and compute the set of reachable states exactly and effectively by symbolic manipulation of linear inequalities. On the other hand, emerging tools such as CHECKMATE [CK99] and **d/dt** [ABDM00], approximate the set of reachable states by polyhedra by optimization techniques. Even though these tools have been applied to interesting real-world examples after appropriate abstractions, scalability remains a challenge, and developing new techniques for efficient verification of hybrid systems remains an active research area.

3 Relevant Ongoing Projects

3.1 CHARON: Hierarchical Hybrid Modeling

In this project we are designing the modeling language, CHARON, for high-level specification of interacting embedded systems with an associated suite of analysis tools [ADE+01]. CHARON allows visual as well as textual specifications of structured state machines, and is built on top of Java.

CHARON allows specification of hybrid dynamics. Discrete updates are specified by *guarded actions* labeling transitions connecting the modes. Some of the variables in CHARON can be declared *analog*, and they flow continuously during continuous updates that model passage of time. The evolution of analog variables can be constrained by differential constraints, algebraic constraints, and invariants which limit the allowed durations of flows.

In CHARON, the building block for describing the system architecture is an *agent* that communicates with its environment via shared variables. The language supports the operations of *composition* of agents to model concurrency, *hiding* of variables to restrict sharing of information, and *instantiation* of agents to support reuse. The building block for describing flow of control inside an atomic agent is a *mode*. A mode is basically a hierarchical state machine, that is, a mode can have submodes and transitions connecting them. Variables can be declared locally inside any mode with standard scoping rules for visibility. Modes can be connected to each other only via well-defined entry and exit points. We allow *sharing* of modes so that the same mode definition can be instantiated in multiple contexts.

CHARON supports observational trace semantics for both modes and agents. The key result is that the set of traces of a mode can be constructed from the traces of its submodes. This result leads to a compositional notion of refinement for modes. Suppose we obtain an implementation design I from a specification design

S simply by locally replacing some submode N in S by a submode M. Then, to show I refines S, it suffices to show that M refines N.

Predicate abstraction has emerged to be a powerful technique for extracting finite-state models from infinite-state discrete programs. Recently we have developed algorithms and tools for reachability analysis of hybrid systems by combining the notion of predicate abstraction with recent techniques for approximating the set of reachable states of linear systems using polyhedra. Given a hybrid system and a set of user-defined boolean predicates, we consider the finite discrete quotient whose states correspond to the all possible truth assignments to the input predicates. The tool performs an on-the-fly exploration of the abstract system by using weakest preconditions to compute abstract transitions corresponding to the discrete switches and conservative polyhedral approximations to compute abstract transitions corresponding to continuous flows. Compared to tools such as CHECKMATE and **d/dt**, this approach requires significantly less computational resources as the emphasis is shifted from computing the reachable set to searching in the abstract quotient. We have demonstrated the feasibility of the proposed technique by analyzing a parametric timing-based mutual exclusion protocol and safety of a simple controller for vehicle coordination.

We have used CHARON for case studies in multirobot coordination and for automotive challenge problems in the DARPA Mobies program. CHARON has also found unanticipated application domains: for rapid prototyping of animation strategies in physics-based modeling, and for specification of biological systems such as luminescence in the bacterium *Vibrio Fischeri*. More details about CHARON are available at `www.cis.upenn.edu/mobies/charon/`.

3.2 Scenario-based Requirements

Scenario-based specifications such as message sequence charts offer an intuitive and visual way of describing design requirements. Such specifications focus on message exchanges among communicating entities in distributed software systems. Recent standardization of syntax and semantics (MSC'96 or Z.120) by ITU and integration into modern object-oriented software engineering methodologies such as UML, lead us to believe that scenario-based specifications will play an increasingly important role in design of concurrent systems. We are developing analysis techniques and tools for effective use of scenario-based requirements in software design.

Our initial effort led to a visual tool for detection of race conditions in MSCs [AHP96], and is used by developers in Lucent (see `cm.bell-labs.com/cm/cs/what/ubet/`). An interesting recent proposal concerns the notion of "inference" of scenario-based requirements [AEY00]. When a designer draws multiple scenarios as requirements, a question of interest is whether the behaviors they have described are realizable by some distributed implementation. We have presented a language-theoretic framework to formalize such and related questions. In particular, we have developed a polynomial-time algorithm to derive unspecified and possibly unwanted scenarios that are "implied" by the specified set of MSCs.

3.3 HERMES: Model Checking of Hierarchical State Machines

Software modeling languages such as STATECHARTS [Har87] use hierarchical state machines for structured specification of control flow. Our research is aimed at developing techniques for exploiting the hierarchical structure for reducing the computational requirements of algorithms for state-space exploration. We have developed a model checker called HERMES for creating, manipulating, and verifying hierarchical models.

For enumerative checking, the key relevant features of a structured model are information hiding and sharing. The interface of each mode declares the variables that it reads and writes. Our first optimization avoids recomputing transitions within a mode from two distinct states that agree on the information relevant to a mode. The second optimization is relevant when the same mode definition is shared within multiple contexts. We have demonstrated significant savings in computational time at a small price in extra memory usage in analysis of network protocols such as TCP and PPP.

For symbolic checking, the transition relation is maintained indexed by the modes and their control points, providing a generalization of the traditional *conjunctively partitioned* representation. The state-sets generated during search are also maintained indexed by control points allowing the use of typing information for early quantification. Our algorithm computes the macro-transitions (i.e., sequences of individual transitions leading from entry to exit points) of a mode by first computing the strongly-connected-components

of the underlying graph of control points. We have demonstrated the savings of the symbolic checker using circuits from the ISCAS benchmark.

References

[ABDM00] E. Asarin, O. Bournez, T. Dang, and O. Maler. Approximate reachability analysis of piecewise-linear dynamical systems. In *Hybrid Systems: Computation and Control, Third International Workshop*, LNCS 1790, pages 21–31. 2000.

[AD94] R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[ADE+01] R. Alur, T. Dang, J. Esposito, R. Fierro, Y. Hur, F. Ivancic, V. Kumar, I. Lee, P. Mishra, G. Pappas, and O. Sokolsky. Hierarchical hybrid modeling of embedded systems. In *Embedded Software, First International Workshop*, LNCS 2211, pages 14–31. Springer, 2001.

[AEY00] R. Alur, K. Etessami, and M. Yannakakis. Inference of message sequence charts. In *Proc. of 22nd Int. Conf. on Software Engineering*, 2000.

[AHH96] R. Alur, T.A. Henzinger, and P.-H. Ho. Automatic symbolic verification of embedded systems. *IEEE Transactions on Software Engineering*, 22(3):181–201, 1996.

[AHP96] R. Alur, G.J. Holzmann, and D. Peled. An analyzer for message sequence charts. *Software Concepts and Tools*, 17(2):70–77, 1996.

[BG88] G. Berry and G. Gonthier. The synchronous programming language ESTEREL: design, semantics, implementation. Technical Report 842, INRIA, 1988.

[BJR97] G. Booch, I. Jacobson, and J. Rumbaugh. *Unified Modeling Language User Guide*. Addison Wesley, 1997.

[BR00] T. Ball and S. Rajamani. Bebop: A symbolic model checker for boolean programs. In *SPIN 2000 Workshop on Model Checking of Software*, LNCS 1885, pages 113–130. Springer, 2000.

[CDH+00] J.C. Corbett, M.B. Dwyer, J. Hatcliff, S. Laubach, C.S. Pasareanu, Robby, and H. Zheng. Bandera: Extracting finite-state models from Java source code. In *Proceedings of 22nd International Conference on Software Engineering*, pages 439–448. 2000.

[CK96] E.M. Clarke and R.P. Kurshan. Computer-aided verification. *IEEE Spectrum*, 33(6):61–67, 1996.

[CK99] A. Chutinan and B.K. 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.

[CW96] E.M. Clarke and J.M. Wing. Formal methods: State of the art and future directions. *ACM Computing Surveys*, 28(4):626–643, 1996.

[Har87] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8:231–274, 1987.

[HHW97] T.A. Henzinger, P. Ho, and H. Wong-Toi. HyTech: a model checker for hybrid systems. *Software Tools for Technology Transfer*, 1, 1997.

[HK01] T. Henzinger and C. Kirsch, editors. *Embedded Software, First International Workshop*. LNCS 2211. Springer, 2001.

[Hol97] G.J. Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*, 23(5):279–295, 1997.

[JR00] D. Jackson and M. Rinard. The future of software analysis. In *The future of software engineering*, 2000.

[Lee00] E.A. Lee. What's ahead for embedded software. *IEEE Computer*, pages 18–26, September 2000.

[LK00] N. Lynch and B.H. Krogh, editors. *Hybrid Systems: Computation and Control*. LNCS 1790. Springer, 2000.

[LPY97] K. Larsen, P. Pettersson, and W. Yi. Uppaal in a nutshell. *Springer International Journal of Software Tools for Technology Transfer*, 1, 1997.

[McM93] K. McMillan. *Symbolic model checking: an approach to the state explosion problem*. Kluwer Academic Publishers, 1993.

[Mil80] R. Milner. *A Calculus of Communicating Systems*. LNCS 92. Springer-Verlag, 1980.

[MMP91] O. Maler, Z. Manna, and A. Pnueli. From timed to hybrid systems. In *Real-Time: Theory in Practice, REX Workshop*, LNCS 600, pages 447–484. Springer-Verlag, 1991.

[VvS99] F. Vaandrager and J. van Schuppen, editors. *Hybrid Systems: Computation and Control*. LNCS 1569. Springer, 1999.