September 2005

Distributed-Code Generation from Hybrid Systems Models for Time-delayed Multirate Systems

Madhukar Anand
University of Pennsylvania, anandm@cis.upenn.edu

Sebastian Fischmeister
University of Pennsylvania, sfischme@saul.cis.upenn.edu

Jesung Kim
University of Pennsylvania, jesung@saul.cis.upenn.edu

Insup Lee
University of Pennsylvania, lee@cis.upenn.edu

Distributed-Code Generation from Hybrid Systems Models for Time-delayed Multirate Systems

Abstract
Hybrid systems are an appropriate formalism to model embedded systems as they capture the theme of continuous dynamics with discrete control. A simple extension, a network of communicating hybrid automata, allows for modeling distributed embedded systems. Although it is possible to generate code from such models, it is difficult to provide formal guarantees in the code with respect to the model. One of the reasons for this is that, the model is set in continuous time and concurrent execution with instantaneous communication, whereas the generated code is set in discrete time with delayed communication. This can introduce semantic differences between the model and the code such as missed transitions, faulty transitions, and altered continuous behavior. The goal of faithful code generation is to minimize these differences.

In this paper, we propose a relaxed criteria of *relative* faithful implementation. Based on this criteria, we propose dynamically adjusting the guard at runtime using estimates of errors for preventing faulty transitions. We also identify a sufficient condition to ensure no missed transitions in the code.

Keywords
Hybrid Systems, Distributed Systems

Distributed-Code Generation from Hybrid Systems Models for Time-delayed Multirate Systems*

Madhukar Anand, Sebastian Fischmeister, Jesung Kim, and Insup Lee
Dept. of Computer and Information Science, University of Pennsylvania
Philadelphia, PA, USA
{anandm,sfischme,jesung,lee}@saul.cis.upenn.edu

ABSTRACT

Hybrid systems are an appropriate formalism to model embedded systems as they capture the theme of continuous dynamics with discrete control. A simple extension, a network of communicating hybrid automata, allows for modeling distributed embedded systems. Although it is possible to generate code from such models, it is difficult to provide formal guarantees in the code with respect to the model. One of the reasons for this is that, the model is set in continuous time and concurrent execution with instantaneous communication, whereas the generated code is set in discrete time with delayed communication. This can introduce semantic differences between the model and the code such as missed transitions, faulty transitions, and altered continuous behavior. The goal of faithful code generation is to minimize these differences.

In this paper, we propose a relaxed criteria of faithfulness, coined *relative* faithful implementation. Based on this criteria, we propose dynamically adjusting the guard at runtime using estimates of errors for preventing faulty transitions. We also identify a sufficient condition to ensure no missed transitions in the code.

Categories and Subject Descriptors: D.2.4 [Software/Program Verification]: Reliability.

General Terms: Reliability, Design, Verification.

Keywords: Hybrid Systems, Distributed Systems.

1. INTRODUCTION

Hybrid systems are an appropriate modeling paradigm for embedded control software, because it can be used to specify continuous change and discrete transition of system states [1, 17]. The benefits of precise modeling can be enhanced if code is generated automatically while maintaining a correspondence with the model.

Since computer systems are discrete, code generation from hybrid systems models requires a *discretized* hybrid systems model. Such a model uses a designer-specified rate by which the continuous state evolves. Since a single processor executes discretely, concurrency in the model is broken on several processors (either on a single system or multiple systems).

Discretization affects switching. In the discretized code, transitions can only occur at the defined rate, i.e., the sampling time. If a transition in the model is only enabled between the sampling times, then it will be missed, called *missed transition*, and the code will behave differently from the model. Example 1 below illustrates such a missed transition.

EXAMPLE 1. *The system comprises two communicating hybrid automata A_0 and A_1. A_0 executes every $0.0002s$ and A_1 every $0.0003s$. The communication delay between them, denoted by ϕ_{12}, equals $0.0001s$ and P_A denotes the state of the automaton. The following execution trace shows that this discretization leads to a missed transition q_2 in A_1. In the trace, we use $x(A_i)$ to denote the local estimate of x at A_i.* □

t	$x(A_0)$	P_{A_0}	$y(A_1)$	$x(A_1)$	P_{A_1}
0.0002	*0.0004*	q_0	*0.0000*	*0.0000*	q_1
0.0003	*0.0004*	q_0	*0.0003*	*0.0004*	q_1
0.0004	*0.0008*	q_0	*0.0003*	*0.0004*	q_1
0.0006	***0.0012***	q_0	***0.0006***	***0.0008***	q_1

This can be prevented, if the rate is high enough compared to the enabled interval of the guard [3]. On the other hand, the rate must be low enough to provide sufficient computing time on the target platform between each execution.

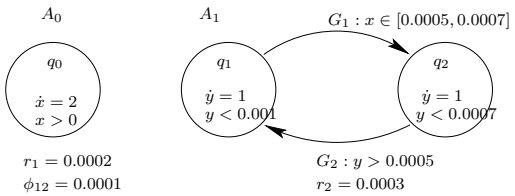

Figure 1: An example for missed transition.

In a discretized system, each component will be assigned an individual rate resulting in a *multirate* system comprising of multiple, different rates. Difference in rates can introduce faulty transitions. A *faulty transition* is an invalid change of system state due to an incomplete snapshot of the system's state. The transition would be disabled, if the snapshot were accurate. For instance, Example 1, if G_1 is changed to $x \in [0.0005, 0.0008]$, then A_1 will transit to q_2 at time 0.0006. This transition is faulty, since at that instant $x = 0.0012$ and the guard should not be enabled.

Problem Statement. In this paper, we consider the problem of faithful implementation of hybrid-systems models with the following assumptions: (1) We consider a system of rectangular linear

*This research is supported in part by NSF CCR-0209024, NSF CCF-0429948, ARO DAAD19-01-1-0473, ARO W911NF-05-1-0182, and OEAW APART-11059.

EMSOFT'05, September 19–22, 2005, Jersey City, New Jersey, USA.

hybrid automata models in which the dependency of different automata is either through dynamics or through switching and the dependency graphs are acyclic, (2) We consider a distributed implementation of the different agents which may have different sampling rates and (3) The system is also assumed to be affected by communication delays that could be arbitrary but bounded. Given such a model, the focus of this paper is to address faulty transitions, missed transitions, and identify sufficient conditions for preventing them.

Proposed Approach. In [14], we have proposed a scheme to prevent faulty transitions by shrinking the guard set to account for the errors due to mixed sampling times. However, introducing such a margin into the guard in turn increases the chances of a missed transition, since it reduces the time interval during which a transition must be taken. We mitigate this problem in this paper by proposing a runtime technique that extends on our previous approach. The idea is that some kinds of errors, e.g., communication delay between distributed components, are better analyzed at runtime as they depend on current dynamics. By instrumenting at runtime, we can avoid a pessimistic decision to avoid a faulty transition. Our scheme is based on the premise that the communication delay can be bounded. To show feasibility, we present a possible configuration of time-deterministic distributed system in Section 4.

Related work. Commercial modeling tools such as SIMULINK also support code generation and address the effect of errors in the code. However, their concerns are largely limited to numerical errors occurring each step during simulation, and the effect of such errors on to discrete behavior is not addressed rigorously. Synchronous languages for reactive systems, such as STATECHARTS [12], ESTEREL [6], and LUSTRE [11], also support code generation. However, they do not support hybrid systems modeling. SHIFT [7] is a language for hybrid automata that also also supports code generation, but the focus is on dynamic networks. Model-based development of embedded systems is also promoted by other projects with orthogonal concerns: Ptolemy supports integration of heterogeneous models of computation [8] and GME supports metamodeling for development of domain-specific modeling languages [15].

Code generation from hybrid models was introduced with focus on single-thread execution in [3]. This was extended to multithreaded models accounting for faulty transitions in [14] and unirate distributed systems with no delays in [4]. To guarantee value and time determinism in the discretized code, we use an E machine approach [13] with the concept of logical execution time in our on-going prototype. However, we plan to generate E code for the E machine directly from the CHARON model without using the Giotto specification language.

The remainder of the work is structured as follows: Section 2 introduces the system model. Section 3 presents our approach of how to prevent missed and faulty transitions. Section 4 provides an overview of the implementation and Section 5 describes the current and future work.

2. SYSTEM MODEL

In this framework, we assume that there is a network of hybrid automata (called *agents*) communicating via a set of shared variables. We denote a single agent by $\mathcal{A} = (A, SV)$ where A is the model for the agent and SV is the set of shared variables. A system of communicating hybrid agents is a tuple $\mathcal{C} = \langle \mathcal{A}_0, \ldots, \mathcal{A}_n \rangle$.

DEFINITION 1. *(DCHA) Given a system of communicating hybrid agents $\mathcal{C}$, and a discrete time domain $LT = \{lt_0, lt_1, \ldots\}$ where $lt_i \in \mathbb{Q}^+$ and $\forall i,\ lt_{i+1} > lt_i$, the discretized system of communicating agents (DCHA) is given by $\mathcal{D} = \langle (A, SV, LT)_0, \ldots, (A, SV, LT)_n \rangle$.* □

Note that the DCHA is the discretization of the continuous model that is implemented on an actual platform. The guarantees for faithful implementation are given with respect to this model. In [14], the authors provide a rigorous definition of *system of communicating agents* and their semantics.

When the agents of the discretized model is mapped to real-time tasks for execution, each agent is assigned a period. This period is normally equal to the frequency of evaluation for an agent.

DEFINITION 2. *(Code) The code, implementing a DCHA $\mathcal{D}$ denoted by $\mathcal{K}$, is given by the tuple $\mathcal{K} = \langle (A, SV, LT, clk, h, \sigma)_0, \ldots, (A, SV, LT, clk, h, \sigma)_n \rangle$, where clk represents the physical time, σ represent the local copy of the shared variables, h is the frequency of evaluation.* □

The frequency h reflects the logical time between evaluations, i.e., $(lt_{i+1} - lt_i)$. Typically, this is constant for any agent and is equal to the period of the real-time task on which the agent is mapped. In a multirate system, different agents have different rates. This version abstracts away several details, e.g., the set of synchronization mechanisms, the collection of subroutines implementing the DCHA, scheduling details, and exact mapping of code into memory. In [5], we provide a descriptive exposition of this.

Finally, we denote the logical communication time between agents $\mathcal{A}_i$ and $\mathcal{A}_j$ by $\phi(i, j)$ and a time-delayed code implementation of $\mathcal{D}$ by $(\mathcal{K}, \phi)$.

3. VALIDATION OF GENERATED CODE

To provide a faithful implementation, we have to guarantee the absence of faulty and missed transitions [4]. The implementation should also be independent of the scheduling. However, in our system, delays can vary for each agent and also between single executions of the same agent. Such behavior introduces errors resulting in incorrect behavior. For example, in one mode, an agent could depend on x that arrives $0.0001s$ late and in another mode, it could depend on y that arrives $0.0002s$ later. Therefore, we relax the criteria for a faithful implementation by requiring that code enters the state of the model no later than the maximum possible delay. Formally, we can define,

DEFINITION 3. *(Relative Faithful Implementation) Let VC be the set of all variables and α_x be the maximum bound on the error of a variable x. Given a trace of states of the code $\mathcal{K}$ for an agent $\mathcal{A}_j$, $\langle q_0, q_1, \ldots \rangle$, at physical time-stamps $\langle clk_0, clk_1, \ldots \rangle$, if, $\forall clk$,*

1. *$\forall x \in VC, |x_{\mathcal{D}} - x_{\mathcal{K}}| < \alpha_x$, where $x_{\mathcal{K}}$ and $x_{\mathcal{D}}$ represent the value of variable in the code and the model respectively.*
2. *$\forall j,\ \exists q_{\mathcal{D}}, q_{\mathcal{K}} = q_{\mathcal{D}}\ , (lt_{\mathcal{D}} - lt_{\mathcal{K}}) < \phi_j + \varphi$ where $q_{\mathcal{K}}$ is the state of the code at logical time $lt_{\mathcal{K}}$, at physical time clk, $q_{\mathcal{D}}$ is the projection of the state of the model onto the code for $\mathcal{A}_j$ at logical time $lt_{\mathcal{D}}$, $\phi_j = \max_i \phi(i, j)$ and φ is the maximum skew due to different rates of updates.*

then, code for $\mathcal{A}_j$ is said to be a relative faithful implementation. If $\forall j$, the code for $\mathcal{A}_j$ is a relative faithful implementation, then $\mathcal{K}$ is a relative faithful implementation of $\mathcal{D}$. □

Condition 1 states that the error in continuous variables be bounded and Condition 2 states that the delay in timing of transitions be bounded. We can now provide approaches that safeguard against faulty and missed transitions and thus allow for a relative faithful implementation of a hybrid-systems model.

3.1 Preventing Faulty Transitions

A faulty transition occurs either because the transition was taken on the basis of an older value of the variable or because of numerical errors in the variables.

Static instrumentation was introduced in [14] as an approach to prevent faulty transitions. The idea there was to instrument the guards and the invariants with maximum possible error in variables and switch conservatively.

DEFINITION 4. *(Static Instrumentation) Let p be the state of agent $\mathcal{A}$ with $E_{\mathcal{A}}(p)$ being the set of discrete transitions, $I_{\mathcal{A}}(p)$ the set of invariants in that state, and $G_{\mathcal{A}}(e), e \in E_{\mathcal{A}}(p)$ the set of guards. Let $\forall x, \delta_{p,x}$ give the upper bound on the numerical and timing errors. If l and u denote lower and upper bounds respectively, then,*

1. $\forall p, \forall I \in I_{\mathcal{A}}(p)$,
   $l(I_x) \leftarrow l(I_x) + \delta_{p,x}$, $u(I_x) \leftarrow u(I_x) - \delta_{p,x}$.
2. $\forall p, \forall G \in G_A(e)$,
   $l(G_x) \leftarrow l(G_x) + \delta_{p,x}$, $u(G_x) \leftarrow u(G_x) - \delta_{p,x}$.

   where G_x, I_x denote the projection of the guard and invariant on the variable x in agent $\mathcal{A}$ in state p. □

Once the guards and the invariants have been instrumented, the code generated from $\mathcal{D}$ can be assured of no faulty transitions. This is true because the instrumented guard remains disabled for the duration of synchronization and possible numerical errors. For the proof of this claim, we refer the reader to [14]. Though guarantees can be given statically with this scheme, since the guard and invariant sets are shrunk, the probability of not taking a transition increases. Yet another disadvantage is that it is not always possible to determine this error bound beforehand as with most differential equations, it is only possible to get a local estimate of error which is only available at runtime.

The two factors contributing to the error leading to faulty transitions are (1) Numerical errors (e.g., round-off, truncation) and (2) Timing related errors. While the numerical errors are static, the timing errors change over time as they comprise of communication delay ϕ which is bounded and the skew due to different frequencies of update that is dynamic. If h_j is the agent's $\mathcal{A}_j$ execution frequency and depends on variable x updated by agent $\mathcal{A}_i$ with frequency h_i, and $\phi(i,j)$ is a fixed constant, then, the maximum skew φ_{max} is given by $\varphi(i,j)_{max} = \max_{n\in[1..N]} \left(np_j - \left\lfloor \frac{np_j - \phi}{p_i} \right\rfloor p_i\right)$ where $N = \frac{LCM(p_i,p_j)}{p_j}$. Now, if we instrument the guard taking this into account at runtime, we can improve over the static approach. Another advantage of a runtime approach is that we can have a value-specific simplification leading to less pessimistic bounds. For example, if $\dot{x} > 0$, x is an increasing function of time and hence the update for x will be greater than the current estimate. Therefore, if we are evaluating a condition such as $x > 5$, we will not have to instrument it for timing errors.

DEFINITION 5. *(Dynamic Instrumentation) Let p be a state of agent $\mathcal{A}_j$ with $E_{\mathcal{A}_j}(p)$ being the set of discrete transitions, $I_{\mathcal{A}_j}(p)$ the set of invariants in that state, and $G_{\mathcal{A}_j}(e), e \in E_{\mathcal{A}_j}$ the set of guards. Let $\forall x, \beta_{p,x}(t), \gamma_{p,x}(t), t \in T = [lt, lt + d]$ give the bound in T due to timing and numerical errors, respectively. If $\mathcal{I}$ is an indicator function, l and u denote the lower and upper bounds respectively, and the condition on derivatives holds in T,then,*

1. $\forall p, \forall I \in I_{\mathcal{A}_j}(p)$,
   $l(I_x) \leftarrow l(I_x) + \mathcal{I}(\dot{x} \geq 0, t \in T)\beta_{p,x}(t) + \gamma_{p,x}(t)$
   $u(I_x) \leftarrow u(I_x) - \mathcal{I}(\dot{x} \leq 0, t \in T)\beta_{p,x}(t) - \gamma_{p,x}(t)$
2. $\forall p, \forall G \in G_{\mathcal{A}_j}(e)$,
   $l(G_x) \leftarrow l(G_x) + \mathcal{I}(\dot{x} \geq 0, t \in T)\beta_{p,x}(t) + \gamma_{p,x}(t)$
   $u(G_x) \leftarrow u(G_x) - \mathcal{I}(\dot{x} \leq 0, t \in T)\beta_{p,x}(t) - \gamma_{p,x}(t)$

   where G_x, I_x denote the projection of the guard and invariant on the variable x in agent $\mathcal{A}$ in state p. □

Note that with dynamic instrumentation, we have to calculate the error that would happen in the *future*, i.e., the error that would be caused when a particular value is read.

EXAMPLE 2. *Consider the case of Example 1 with $G_1 : x \in [0.0001, 0.001]$. If we statically instrument, we have to change the guard G_1 to $x \in [0.0005, 0.0006]$ because the maximum error $\alpha_x = 2(\varphi(1,2)_{max} + \phi(1,2)) = 2(0.0001 + 0.0001) = 0.0004$ and then we can have a run as,*

t	x	P_{A_0}	y	$x(A_1)$	P_{A_1}
0.0002	*0.0004*	q_0	*0.0000*	*0.0000*	q_1
0.0003	*0.0006*	q_0	*0.0003*	*0.0004*	q_1
0.0004	*0.0008*	q_0	*0.0003*	*0.0004*	q_1
0.0006	***0.0012***	p_0	***0.0006***	***0.0008***	q_1

The transition to q_2 is missed at time $t = 0.0006$. If we had used dynamic instrumentation, then, the guard would have been instrumented to $x \in [0.0003, 0.0008]$, since the error $\alpha_x = 2(\varphi(1,2) + \phi(1,2)) = 2(0 + 0.0001) = 0.0002$ and the transition to q_2 would have been taken. □

The theorem below formally states that dynamic instrumentation prevents faulty transitions.

THEOREM 1. *Let the code $\mathcal{K}$ of the model $\mathcal{D}$ be implemented on a distributed platform. For every agent $\mathcal{A}_j$, let p be the current state with $I_{\mathcal{A}_j}(p)$ the set of invariants in that state, and $G_{\mathcal{A}_j}(e)$ the set of guards. If $\forall G \in G_{\mathcal{A}_j}(e)$ that evaluate to true, G is dynamically instrumented with $\beta_{p,x}(t)$ and $\gamma_{p,x}(t)$ for all shared variables x, then, there will be no faulty transitions.* □

The timing errors can be computed in practice, by delegating this responsibility to different entities such as the agent that writes the shared variable or the agent that reads with different advantages.

3.2 Preventing Missed Transitions

Missed transitions are transitions that are enabled in the model but not taken in the code. They occur either because the guard is not evaluated sufficiently or scheduling affected the order of evaluation. In general, a transition will not be missed, if it stays enabled long enough to be detected. The theorem below gives a sufficient condition to prevent missed transitions.

THEOREM 2. *Let the code $\mathcal{K}$ of the model $\mathcal{D}$ be implemented on a distributed platform, h_i and h_j be the frequency of sampling in $\mathcal{A}_i$ and $\mathcal{A}_j$ respectively, and the instrumented guard set $G(p) \subseteq I(p)$ be such that an active transition is enabled in state p. If $G(p)$ and $I(p)$ overlap by $\Omega = 2h_j + \sum_{(k_1,k_2)\in E}(2h_{k_1} + \phi(k_1,k_2) + \varphi(k_1,k_2)_{max})$, where the summation is over all frequencies along the longest path in the switching dependency graph E of G, and $\varphi(k_1,k_2)$ represents the maximum skew at agent $\mathcal{A}_{k_2}$ due to $\mathcal{A}_{k_1}$, then, the transition will be detected and will not be missed if they are taken as soon as enabled.*

PROOF. *(sketch)* First consider the case where the transitions are dependent on independent updates (such as sensor readings) but not updates from other agents. In the code, evaluation of the guard condition might be scheduled at some time $jh, j \in \mathbb{Z}^+$ but

a guard may be enabled in the model immediately after that i.e., at time $jh+\epsilon, \epsilon > 0$. This will be detected in the code during the next evaluation which may be scheduled as late as $(j+2)h - \eta$. Since we assume that the transition will be taken as soon as detected, this transition will be taken at $(j+2)h - \eta$. Letting $\eta, \epsilon \rightarrow 0$, in the worst case, the instrumented guard and invariant should overlap by at least $2h$ to avoid missing the transition.

Now consider the general case where the transition may depend on updates from other agents. Here, an update may be delayed by the communication (ϕ) and the skew due to different rates (ψ) in addition to $2h$ at each agent due to sampling. By observing that this delay could be as large as sum of delays along the longest path in the switching dependency graph, we get the desired result. □

The condition for overlap can be checked statically starting from the node with least dependency and iterating until the most dependent. Theorems 1 and 2 provide a sufficient condition to ensure a relative faithful implementation that we record in the following corollary.

COROLLARY 1. *Let the code $\mathcal{K}$ of the model $\mathcal{D}$ be implemented on a distributed platform. If, (1) in the code for every agent $\mathcal{A}_i$, every dynamically instrumented $G \in G_{\mathcal{A}_j}$ and corresponding invariant I satisfy the condition of overlap in Theorem 2, and (2) all variables in $\mathcal{K}$ have bounded error, then, $\mathcal{K}$ is a relative faithful implementation of $\mathcal{D}$.* □

4. IMPLEMENTATION

We use CHARON [2] to implement ideas introduced in this paper. CHARON allows for modular specification of interacting hybrid systems and supports automatic code generation [3]. Once the model is specified in CHARON, the code generator for an agent takes a sampling period as an input and produces code that approximates the continuous behavior of the model. During this process, the guards and invariants are statically instrumented for numerical errors and the condition of overlap in Theorem 2 is used to check whether the code can be guaranteed against missed transitions. The responsibility of dynamic instrumentation to ensure no faulty transitions is delegated to the runtime environment. Specifically, the runtime environment, (1) ensures that the updates are delivered time deterministically, and, (2) provides the values of delay (ϕ) and skew (ψ) for dynamic instrumentation.

As an underlying infrastructure of the distributed real-time system in our ongoing implementation, we use an existing system [16, 10, 9], which supports the concept of logical execution time (LET) [13]. This system provides time and value determinism across distributed nodes and is therefore apt to implement distributed CHARON models with constant and guaranteed delays at the cost of runtime performance. The worst-case execution time of an agent and the worst-case communication time together define the LET of the agent and thus its highest execution rate. This allows us to run each agent on a different node in a distributed system while it is oblivious of the distribution. If required, all the modified values can be communicated after the agent finishes execution and before the end of the LET. The limiting factor here is the shared communication channel between nodes. Given the agents' frequencies and their node assignment, we can calculate the required communication, build a communication schedule, and validate that there are no collisions on the network. In [16], the authors describe how this can be done with tasks and the algorithms and tools can be reused for this implementation.

5. CONCLUSIONS

Hybrid systems based code generation is a promising yet challenging approach for producing reliable embedded software. Providing formal guarantees is difficult due to the semantic differences between the model and the code arising as a result of discretization and communication delays.

In this paper, we have presented an approach to guarantee faithful switching semantics that involves preventing missed and faulty transitions. In contrast to related and prior work, we have defined a notion of relative faithful implementation for systems with multiple rates of evolution. Based on this notion, we have proposed a runtime instrumentation technique to prevent faulty transitions and identified sufficient conditions to prevent missed transitions in the generated code. We have also sketched an implementation to show how timing delays can be bounded and accounted for, in the model.

In the future, we will consider fully integrating the infrastructure into the CHARON development environment and provide comprehensive techniques to detect missed transitions in the code at runtime.

6. REFERENCES

[1] R. Alur, C. Courcoubetis, N. Halbwachs, T. Henzinger, P. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Comp. Science*, 138:3–34, 1995.

[2] R. Alur, R. Grosu, Y. Hur, V. Kumar, and I. Lee. CHARON: a language for modular specifi cation of multi-agent hybrid systems. Technical Report MS-CIS-00-01, Dept. of Computer and Information Science, University of Pennsylvania, Jan. 2000.

[3] R. Alur, F. Ivančić, J. Kim, I. Lee, and O. Sokolsky. Generating embedded software from hierarchial hybrid models. In *Proceedings of ACM Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES)*, 2003.

[4] M. Anand, J.Kim, and I.Lee. Code generation from hybrid systems models for distributed embedded systems. In *Proceedings of the IEEE ISORC*, pages 166–173, 2005.

[5] M. Anand, J.Kim, and I.Lee. Sound code generation from hybrid system models: Some theoretical results. In *Technical report, MS-CIS-05-03, University of Pennsylvania*, 2005.

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

[7] A. Deshpande, A. Göllu, and P. Varaiya. SHIFT: a formalism and a programming language for dynamic networks of hybrid automata. In *Hybrid Systems V*, LNCS 1567. Springer, 1996.

[8] J. Eker, J. Janneck, E.A. Lee, J. Liu, X. Liu, J. Luvig, S. Neuendorffer, S. Sachs, and Y. Xiong. Taming heterogeneity–the Ptolemy approach. *Proceedings of the IEEE*, 91(1):127–144, 2003.

[9] S. Fischmeister. Multi-dimensional schedules for media-access control in time-triggered communication. In *Proc. of the IEEE Symposium on Computers and Communications (ISCC'05)*. IEEE Press, 2005.

[10] S. Fischmeister and G. König. On Using Multiple Virtual Machines For Distributed Real-Time Systems. Technical Report TR-08, University of Salzburg, Jakob-Haringer-Str. 2, 5020 Salzburg, Austria, Feb. 2005 2005.

[11] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous dataflow programming language Lustre. *Proceedings of the IEEE*, 79:1305–1320, 1991.

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

[13] T.A. Henzinger, C.M. Kirsch, M.A.A. Sanvido, and W. Pree. From control models to real-time code using Giotto. *IEEE Control Systems Magazine*, February 2003.

[14] Y. Hur, J. Kim, I. Lee, and J.-Y. Choi. Sound code generation from communicating hybrid models. In *Proceedings of HSCC*, LNCS 2993, pages 432–447, 2004.

[15] G. Karsai, J. Sztipanovits, A. Ledeczi, and T. Bapty. Model-integrated development of embedded software. *Proceedings of the IEEE*, 91(1):145–164, 2003.

[16] G. König. Using Interpreters for Scheduling Network Communication in Distributed Real-Time Systems. Master's thesis, University of Salzburg, Jakob-Haringer-Str. 2, 5020 Salzburg, Austria, March 2005.

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