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

Follow this and additional works at: http://repository.upenn.edu/cis_papers

Recommended Citation

Postprint version. Copyright ACM, 2005. This is the author's version of the work. It is posted here by permission of ACM for your personal use. Not for redistribution. The definitive version was published in Proceedings of the 5th ACM International Conference on Embedded Software (EMSOFT 2005), pages 210-213.
Publisher URL: http://doi.acm.org/10.1145/1086228.1086267
This paper is posted at ScholarlyCommons. http://repository.upenn.edu/cis_papers/178
For more information, please contact repository@pobox.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

Comments
Postprint version. Copyright ACM, 2005. This is the author's version of the work. It is posted here by permission of ACM for your personal use. Not for redistribution. The definitive version was published in Proceedings of the 5th ACM International Conference on Embedded Software (EMSOFT 2005), pages 210-213. Publisher URL: http://doi.acm.org/10.1145/1086228.1086267

This conference paper is available at ScholarlyCommons: http://repository.upenn.edu/cis_papers/178
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 A₀ and A₁. A₀ executes every 0.0003s and A₁ every 0.0002s. The communication delay between them, denoted by $\phi_{12}$, equals 0.0003s 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 $A_i$.

<table>
<thead>
<tr>
<th>t</th>
<th>$x(A_0)$</th>
<th>$P_{A_0}$</th>
<th>$x(A_1)$</th>
<th>$P_{A_1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0002</td>
<td>0.0004</td>
<td>q₀</td>
<td>0.0003</td>
<td>0.0000</td>
</tr>
<tr>
<td>0.0004</td>
<td>0.0004</td>
<td>q₀</td>
<td>0.0003</td>
<td>0.0004</td>
</tr>
<tr>
<td>0.0004</td>
<td>0.0008</td>
<td>q₀</td>
<td>0.0003</td>
<td>0.0004</td>
</tr>
<tr>
<td>0.0006</td>
<td>0.0012</td>
<td>q₀</td>
<td>0.0006</td>
<td>0.0008</td>
</tr>
</tbody>
</table>

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
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 meta-modelling 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 multi-threaded models accounting for faulty transitions in [14] and unidirectional 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 C machine approach [13] with the concept of logical execution time in Section 4. Proposed Approach.

DEFINITION. The code, implementing a DCHA D denoted by k, is given by the tuple k = ((A, SV, LT; clk, h, s)0, . . . , (A, SV, LT; clk, h, s)n), where clk represents the physical time, s 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., (lti+1 − lti). 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 Ai and Aj by φ(i, j) and a time-delayed code implementation of D by (k, φ).

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. Let VC be the set of all variables and €x be the maximum bound on the error of a variable x. Given a trace of states of the code k for an agent Ai, (q0, q1, . . . , q), at physical time-stamps (clk0, clk1, . . . ), if ∀clk,

1. ∀x ∈ VC, |xp − xc| < €x, where xc and xp represent the value of variable in the code and the model respectively.

2. ∀j, ∀q, q = qt, (lq − lt) < φ + ϕ where xc is the state of the code at logical time lt, xc at physical time clk, q is the projection of the state of the model onto the code for Aj at logical time lq, φ = max φ(i, j) and ϕ is the maximum skew due to different rates of updates. then, code for Aj is said to be a relative faithful implementation. If ∀j, the code for Aj is a relative faithful implementation, then k is a relative faithful implementation of 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 \( A \) with \( E_A(p) \) being the set of discrete transitions, \( I_A(p) \) the set of invariants in that state, and \( G_A(e), e \in E_A(p) \) 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. \( \forall p, \forall I \in I_A(p), \)
   \[ l(I_x) = l(I_x) + \delta_p, x, u(I_x) = u(I_x) - \delta_p, x \]
2. \( \forall p, \forall G \in G_A(e), \)
   \[ l(G_x) = l(G_x) + \delta_p, x, u(G_x) = u(G_x) - \delta_p, x \]

where \( G_x, I_x \) denote the projection of the guard and invariant on the variable \( x \) in agent \( A \) in state \( p \).

Once the guards and the invariants have been instrumented, the code generated from \( D \) 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 \( \phi \) which is bounded and the skew due to different frequencies of update that is dynamic. If \( b_j \) is the agent’s \( A_i \) execution frequency and depends on variable \( x \) updated by agent \( A_i \) with frequency \( f \), and \( \phi(i, j) \) is a fixed constant, then, the maximum skew \( \phi_{max} \) is given by \( \phi(i, j)_{max} = \max_{n \in \{1, N\}} \left( np_j - \frac{np_j - \phi}{p_i} \right) \)

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 \( \delta > 0 \), \( x \) 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 \( A \) with \( E_A(p) \) being the set of discrete transitions, \( I_A(p) \) the set of invariants in that state, and \( G_A(e), e \in E_A(p) \) the set of guards. Let \( \forall x, \delta_p, x(t), \gamma_p, x(t), t \in \mathbb{T} = [l, l + \delta] \) give the bound in \( T \) due to timing and numerical errors, respectively. If \( I \) is an indicator function, \( l \) and \( u \) denote the lower and upper bounds respectively, and the condition on derivatives holds in \( T \), then,

1. \( \forall p, \forall I \in I_A(p), \)
   \[ l(I_x) = l(I_x) + \mathbb{T}(\delta_x \geq 0, t \in T)\delta_p, x(t) + \gamma_p, x(t) \]
   \[ u(I_x) = u(I_x) - \mathbb{T}(\delta_x \leq 0, t \in T)\delta_p, x(t) - \gamma_p, x(t) \]
2. \( \forall p, \forall G \in G_A(e), \)
   \[ l(G_x) = l(G_x) + \mathbb{T}(\delta_x \geq 0, t \in T)\delta_p, x(t) + \gamma_p, x(t) \]
   \[ u(G_x) = u(G_x) - \mathbb{T}(\delta_x \leq 0, t \in T)\delta_p, x(t) - \gamma_p, x(t) \]

where \( \delta_x, I_x \) denote the projection of the guard and invariant on the variable \( x \) in agent \( A \) 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 \( G_1 : x \in [0.0001, 0.0001]. \) If we statically instrument, we have to change the guard \( G_1 \) to \( x \in [0.0000, 0.0001] \) because the maximum error \( \alpha_x = 2(\varphi(1, 2)_{max} + \phi(1, 2)) = 2(0.0001 + 0.0001) = 0.0002 \) and then we can have a run as:

<table>
<thead>
<tr>
<th>t</th>
<th>x</th>
<th>( P_{A_0} )</th>
<th>y</th>
<th>x(A_1)</th>
<th>( P_{A_1} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0002</td>
<td>0.0004</td>
<td>0.90</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.91</td>
</tr>
<tr>
<td>0.0004</td>
<td>0.0006</td>
<td>0.90</td>
<td>0.0000</td>
<td>0.0001</td>
<td>0.91</td>
</tr>
<tr>
<td>0.0006</td>
<td>0.0012</td>
<td>0.90</td>
<td>0.0006</td>
<td>0.0008</td>
<td>0.91</td>
</tr>
</tbody>
</table>

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 \( \alpha_x = 2(\varphi(1, 2) + \phi(1, 2)) = 2(0.0001 + 0.0001) = 0.0002 \) 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 set \( K \) of the model \( D \) be implemented on a distributed platform. For every agent \( A_i \), let \( p \) be the current state with \( I_A(p) \) the set of invariants in that state, and \( G_A(e) \) the set of guards. If \( V \in G_A(e) \) that evaluate to true, \( G \) is dynamically instrumented with \( \delta_p, x(t) \) 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 \( K \) of the model \( D \) be implemented on a distributed platform, \( h_i \) and \( h_j \) be the frequency of sampling in \( A_i \) and \( A_j \) respectively, and the instrumented guard set \( G(p) \subseteq I(p) \) such that an active transition is enabled in state \( p \). If \( G(p) \) and \( I(p) \) overlap by \( \Omega = 2h + \sum (k_1, k_2) \in C^2 (2h_k_1 \phi(k_1, k_2) + \phi(k_1, k_2) )_{max} \), where the summation is over all frequencies along the longest path in the switching dependency graph \( E(p) \), and \( \phi(k_1, k_2) \) represents the maximum skew at agent \( A_k \) due to \( A_k \), 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 \( jh, j \in \mathbb{Z}^+ \)
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 $\eta, \epsilon \to 0$, 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 ($\phi$) and the skew due to different rates ($\psi$) 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. \hfill \Box

The condition for overlap can be checked statically starting from
the node with least dependency and iterating until the most depen-
dent. 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 $K$ of the model $D$ be implemented
on a distributed platform. If, (1) in the code for every agent $A_i$,
every dynamically instrumented $G \in G_{A_i}$ and corresponding in-
variant $I$ satisfy the condition of overlap in Theorem 2, and (2) all
variables in $K$ have bounded error, then, $K$ is a relative faithful
implementation of $D$. \hfill \Box

4. **IMPLEMENTATION**

We use Charon [2] to implement ideas introduced in this pa-
per. Charon allows for modular specification of interacting hy-
brid 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 approx-
imates the continuous behavior of the model. During this process,
the guards and invariants are statically instrumented for numeri-
cal errors and the condition of overlap in Theorem 2 is used to
check whether the code can be guaranteed against missed transi-
tions. The responsibility of dynamic instrumentation to ensure no
faulty transitions is delegated to the runtime environment. Specifi-
cally, the runtime environment, (1) ensures that the updates are de-

givered time deterministically, and, (2) provides the values of delay
($\phi$) and skew ($\psi$) for dynamic instrumentation.

As an underlying infrastructure of the distributed real-time sys-
tem 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 dis-

gributed nodes and is therefore apt to implement distributed Charon
models with constant and guaranteed delays at the cost of run-
time 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 obvi-
ous 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 communi-
cation channel between nodes. Given the agents’ frequencies and
their node assignment, we can calculate the required communica-
tion, 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 chal-

lenging approach for producing reliable embedded software. Pro-

viding 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 faith-
ful 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 mul-
tiple 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 compre-
hensive techniques to detect missed transitions in the code at run-
time.

6. **REFERENCES**

[1] R. Alur, C. Courcoubetis, N. Halbwachs, T. Henzinger, P. Ho, X. Nicollin,
A. Olivo, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid
modular specification of multi-agent hybrid systems. Technical Report
MS-CIS-00-01, Dept. of Computer and Information Science, University of
software from hierarchical hybrid models. In Proceedings of ACM Confer-
distributed embedded systems. In Proceedings of the IEEE ISORC, pages
models: Some theoretical results. In Technical report, MS-CIS-05-03,
University of Pennsylvania, 2005.
[7] A. Deshpande, A. Gollu, and P. Varaiya. SHIFT: a formalism and
a programming language for dynamic networks of hybrid automata. In Hybrid
and Y. Xiong. Taming heterogeneity—the Ptolemy approach. Proceedings of the
[9] S. Fischmeister. Multi-dimensional schedules for media-access control in
time-triggered communication. In Proc. of the IEEE Symposium on Comput-
Distributed Real-Time Systems. Technical Report TR-08, University of
models to real-time code using Giotto. IEEE Control Systems Magazine,
February 2003.
communicating hybrid models. In Proceedings of HSCC, LNCS 2993, pages
development of embedded software. Proceedings of the IEEE, 91(1):145–164,
2003.
[16] G. Kiong. 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.
Real-Time: Theory in Practice, REK Workshop, LNCS 600. Springer-Verlag,