Qualifying Examination Proposal

Component-Based Hierarchical Modeling of Systems with Continuous and Discrete Dynamics

Jie Liu
Department of Electrical Engineering and Computer Science
University of California, Berkeley

December 3, 1999

1. Motivation

The increasing complexity of embedded system design often requires the designers to model systems with both continuous and discrete dynamics. Examples include mixed-signal systems that have both continuous and discrete parts and digital systems that interact with a continuous environment. Although each individual model may be relatively well-understood, the integration of heterogeneous models brings additional difficulties and complexities to the design.

This work focuses on the ordinary differential equation (ODE) based *continuous time* (CT) model and two kinds of discrete models, a timed one – the *discrete event* (DE) model, and an untimed one – the *finite state machine* (FSM) model. Following circuit design communities, we call the composition of CT and DE the *mixed-signal* model; following control and computation communities, we call the composition of CT and FSM the *hybrid system* model.

Mixing heterogeneous models to design complex systems is receiving more and more attention from both academic and industrial perspectives. SPLICE [18], a mixed-signal circuit design tool, tries to speed up the simulation by extending the SPICE-like ODE solver to embrace an event-driven simulation engine for digital parts. Unfortunately, it does not accurately detect events, which makes the tight feedback between analog and digital parts hard to handle. Simulink[1], originally a continuous-time control system design tool, has been enhanced to model sample-data systems and, to some extent, discrete-event systems [23]. The hardware description language communities, like VHDL and Verilog, extend and standardize the capability of modeling continuous dynamics in their previous discrete-event based languages [24, 25]. Both of these approaches attempt to come up with a unified model to capture semantically different components, and lack hierarchy for managing the complexity of a design.

One of the most noticeable efforts in mixing continuous and discrete dynamics is the study of hybrid systems [2]. A solid theoretical framework for modeling and analyzing hybrid systems is under rapid development [1, 4, 8, 10, 15], with applications in air traffic management, transportation systems, automotive, manufacturing systems, and electromechanical systems etc. [3, 6, 20, 21]. Nevertheless, a simulation tool with clean semantics and composability that leverages the theory of hybrid system is still under high demand [16].

Component-based models view the building blocks of a system to be components. As shown in Figure 1, components are “black boxes.” The interface of a component and the communication scheme among components are specified, while the contents of a component can be implemented with a different model. The component-based design provides a clean way to integrate different models by hierarchically composing heterogeneous components [5]. This hierarchical composition allows one to manage the complexity of a design by information hiding, and to enhance the possibility of design reuse.

1. A software package from Mathworks Inc.

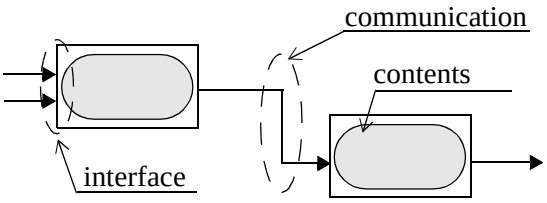

Figure 1. Heterogeneous components

My Thesis

I propose a component-based framework to model systems with both continuous and discrete dynamics. The framework cleanly integrates heterogeneous models by using hierarchical compositions to hide the implementation details of one component from other components, and keeping the components at the same level of hierarchy interacting in the same way. The signal conversions at the boundaries and the execution control among continuous and discrete components are studied. A correct and efficient simulation engine is built. I propose to scale up the framework in order to support modeling of distributable systems. I also expect to contribute via a mathematical understanding of the mixed-signal model.

2. Current Progress

2.1.Component-based model for continuous and discrete dynamics.

- *Continuous Time Model*

We consider an initial value problem of ODE for continuous time systems, and specify it using components, as shown in Figure 2. In this model, components communicate via piecewise continuous waveforms. And the components are continuous maps from input waveforms to output waveforms. The simulation of this model is to solve the ODE numerically with respect to the inputs at a discrete set of time points, and produce outputs at those points. The component-based model imposes no difficulty on numerical ODE solving methods. In fact, the evaluation of the *f* and *g* functions can be achieved by executing the corresponding components in their I/O topological order.

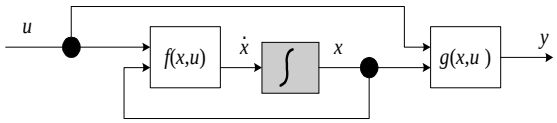

Figure 2. A component-based model for continuous time systems.

- *Discrete Event Model*

In the discrete event model, components communicate by a set of events that have discrete locations on the time line. An event has a time stamp and a value. A component, when executed, can consume input events and produce output events. The output events are required to be no earlier in time than the input events (this property is called *causality*). The simulation of this model utilizes a global event queue. When a component generates an output event, the event is placed in the queue, which sorts events by their time stamps. At each iteration of the simulation, events with the smallest time stamp will be dequeued, and their destination components will be executed.

- *Finite State Machine Model*

In the FSM model, as shown in Figure 3, there is a finite set of states (the bubbles), a finite set of events, an initial state, and transitions from states to states (the arcs). The set of events do not necessarily have a notion of time. A transition is associated with a trigger condition and an action. A trigger condition could be a predicate on input events, and an action might be producing output events. The execution of the system starts from the initial state. For each input event, if the trigger condition on a transition starting

from the current state is true, then the transition is taken and the action associated will be performed. The end state of the transition becomes the new current state.

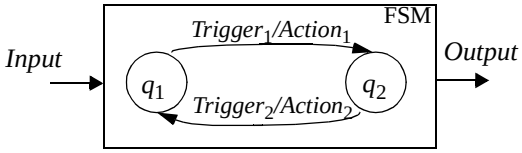

Figure 3. A finite state machine component.

2.2.Signal Conversions and Breakpoint Handling.

A fundamental issue for composing heterogeneous models is how to make a component implemented in one model to expose an interface of another model. Since the two kinds of dynamics have distinct types of signals, the conversion of signals is essential.

- *Event generation*

Event generation is to generate discrete events from piecewise continuous waveforms. We classify two types of events, *time events* and *state events*. The time stamps of time events are known beforehand, while the time stamps of state events depend on the value of the state variables in the CT system. In general, the time stamps of state events cannot be predicted accurately in advance. Special treatment has to be done in the process of ODE solving.

- *Waveform generation*

Waveform generation is to generate piecewise-continuous waveforms from discrete-event signals. This conversion is application dependent. In general, any extrapolations of previous events are reasonable.

- *Breakpoint handling in CT Simulation*

In order to handle event generation and inputs generated from discrete events, we define breakpoints in the continuous-time model. A *breakpoint* is a time point in the CT model when the right-hand side (RHS) of the ODE is not sufficiently smooth, or the output map is not continuous. The numerical ODE solver cannot cross breakpoints in one integration step since either the smooth-RHS assumption is violated or an event need to be produced. According to whether a breakpoint can be predicted in advance, we classify two kinds of breakpoints, *predictable* ones and *unpredictable* ones. For example, time events and unsmoothness in input signals are predictable, while state events and unsmoothness in state variables are unpredictable. Predictable breakpoints can be stored in a table and handled more efficiently. The numerical integration step sizes are now controlled based on three factors:

•*Error control*. This reflects the trade-off between speed and accuracy of a simulation. In general, for a given ODE solving method, a smaller step size means a more accurate result. But it also means more function evaluations and long simulation time.

•*Convergence*. Implicit numerical methods use fixed-point iteration or Newton iteration to solve the induced algebraic equations. Choosing smaller step size may help improve the initial guess.

•*Breakpoints*. Before each integration step, the breakpoint table is queried, and the intent step size (adjusted from the first two factors) may be reduced so that it does not cross a predictable breakpoint. Unpredictable breakpoints are handled by querying components after each integration step. An unpredictable breakpoint is iteratively located within an error tolerance, before the integration continues.

These techniques allow discrete components to expose continuous interface, as shown in Figure 4, and vice versa, as in Figure 5. The breakpoint handling mechanisms built in the continuous-time model suggests that the simulation of discrete dynamics does not have to handle continuous signals.

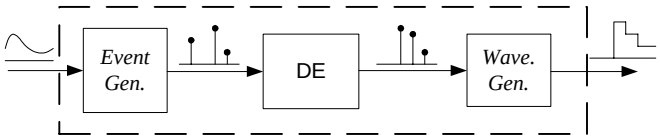

Figure 4. A DE component in a CT model.

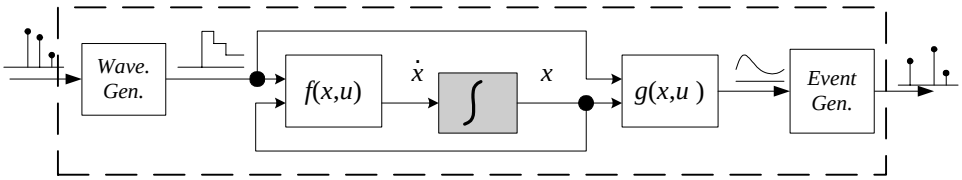

Figure 5. A CT component in a DE model.

2.3.Executing Heterogeneous Components

The execution control of heterogeneous components is critical for a correct and efficient simulation engine. Both CT and DE are timed models. There will be multiple time variables in different components. In our hierarchical composition of models, we call the time at the highest level of hierarchy the "global time," and the time maintained within a component the "local time."

- *DE inside CT*

Since time advances monotonically in CT and events are generated chronologically, the DE component will receive input events monotonically in time. In addition, a composition of causal DE components is causal [11], so the time stamps of output events from a DE component are always greater than or equal to those of the corresponding input events. Thus, from the CT system point of view, the events (breakpoints) produced by a DE component are always predictable.

- *CT inside DE*

When a CT component is contained in a DE system, as shown in Figure 5, the CT component is required to be causal, like all other components in the DE system. This suggests that the local time of the CT component should be greater than or equal to the global time, whenever it is executed [13].

This ahead-of-time execution implies that the CT component should be able to remember its past states and be ready to rollback if the input event is earlier than the local time. The state it needs to remember is the state of the component after it has processed an input event. Consequently, the CT component should not emit detected events to the outside DE system before the global time reaches the event time. Instead, it should request an execution from the DE system at the event time, and wait until its safe to emit it.

- *CT-FSM-CT*

A hierarchical composition of FSM and CT is shown in Figure 6. Although FSM is an untimed model,

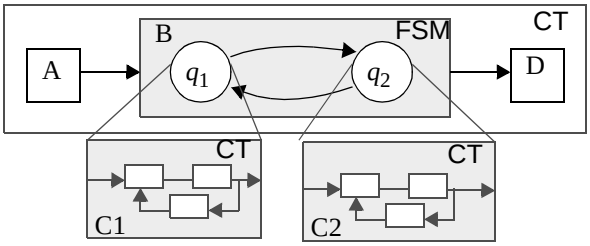

Figure 6. A hierarchical hybrid system.

its composition with a timed model requires it to transfer the notion of time from its external model to its internal model. A CT component, by adopting the event generation technique, can have both continuous and discrete signals as its outputs. The FSM may use predicates on them to build trigger conditions. Actions associated with transitions are usually reset to the initial conditions of integrators in the destination state.

During continuous evolution, the system is simulated as a CT system where the FSM is replaced by the continuous component of its current state. After each time point of CT simulation, the triggers on the transitions starting from the current FSM state are evaluated. If a trigger is enabled, the FSM makes the corresponding transition. The continuous dynamics of the destination state is initialized by the action on the transition. The simulation continues with the transition time treated as a breakpoint.

Currently, the framework for component-based modeling of systems with continuous and discrete dynamics has been built in the heterogeneous modeling and design environment, Ptolemy II [7]. The techniques discussed in the previous sections have been implemented. Several examples have been built to demonstrate the correct simulation results from using these techniques [27, 14].

3. Future Work

I intend to extend the current achievements in two directions, a denotational semantics for the mixed-signal model and a distributable component framework for further scaling up the component model.

3.1.Denotational Semantics for the Mixed-signal Model

Compared to the relative maturity of hybrid system theory, the composition of CT and DE has not been rigorously studied. A denotational semantics [19] defines the behavior of a model by the mathematical relationship on the signals. The study of such semantics for mixed-signal systems may give a deep insight on the properties like determinism and lack of Zeno phenomena (infinitely many events in a finite time interval), as well as a theoretical guide on the implementation of simulators.

The mathematical framework I want to use is the tagged-signal model developed by Lee and Sangiovanni-Vincentelli [12]. The model has been shown to be effective in studying the denotational semantics of DE models [11].

The first question I am trying to answer is related to determinism. Although both DE models and CT models can individually be shown to have a unique behavior under simple conditions (using Cantor metric and L^{∞} space, respectively), the conditions for mixed-signal model may not be trivial. A reason is that the Cantor metric is a metric on time while the $\| \|_{\infty}$ is a metric on values. I would like to come up with conditions that make a composition of continuous-time components (together with event generation and waveform generation processes) to be causal as a discrete-event component, and conditions that make a composition of discrete-event components (together with event generation and waveform generation processes) to be Lipschitz as a continuous-time component.

For the Zeno phenomena, I expect that conditions on the event generation, waveform generation, and system dynamics may help avoid it. The study that Johansson et al. have done on Zeno hybrid automata [9] might be helpful.

3.2.Distributable components

The component-based model has the advantage that there is no restrictions that the components be physically related. It provides a way to protect intellectual property more effectively. In addition, complex embedded systems usually have communication subsystems such that the components interact remotely. A design environment that has the capability to model distributed components is highly desirable.

CORBA [22] and similar middleware techniques [17, 26] provide a promising framework to make the distribution of components transparent to designers. To support them in a design environment, the key questions to be answered are the granularity of distribution, the communication styles among components, and the information to be shared. I intend to embrace the middleware concept in the component model, explore different options of design choices, and provide a practical implementation of it.

4. Conclusion

I expect to contribute a distributable component-based modeling technique for systems with continuous and discrete dynamics, a mathematical framework for it, a correct and efficient simulation strategy, and a practical implementation of it within a component-based multi-model design environment.

References

[1] R. Alur, T. Henzinger, G. Lafferriere, and G.J. Pappas, “Discrete Abstractions of Hybrid Systems,” Submitted to *the Proceedings of the IEEE*, 1999.

[2] P. Antsaklis, W. Kohn, A. Nerode, and S. Sastry (ed.), *Hybrid Systems II*, LNCS 999, Springer 1995.

[3] A. Balluchi, M.D. Di Benedetto, C. Pinello, C. Rossi, and A. Sangiovanni-Vincentelli, “Hybrid control in automotive applications: the cut-off control,” *Automatica*, vol.35, (no.3), March 1999, p.519-535.

[4] S. Bornot, J. Sifakis, and S. Tripakis, “Modeling Urgency in Timed Systems,” *Compositionality: the significant difference*, COMPOS'97, LNCS 1536, Springer 1997.

[5] W.-T. Chang, A. Kalavade and E. A. Lee, “Effective Heterogeneous Design and Cosimulation,” chapter in *Hardware/Software Co-design*, G. DeMicheli and M. Sami, eds., NATO ASI Series Vol. 310, Kluwer Academic Publishers,1996.

[6] C. Cloet, M. Krucinski, R. Horowitz, and M. Tomizuka, “A hybrid control scheme for a copier paperpath,” *Proceedings of the 1999 American Control Conference*, San Diego, CA, June 1999, p.2114-2118.

[7] J. Davis etc., “Ptolemy II: Heterogeneous Concurrent Modeling and Design in Java,” UCB/ERL M99/40, University of California, Berkeley, CA 94720.

[8] T. A. Henzinger, “The theory of hybrid automata,” *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science* (LICS 1996), p.278-292.

[9] K.H. Johansson, M. Egerstedt, J. Lygeros, and S. Sastry, “On the Regularization of Zeno Hybrid Automata,” submitted to *Systems Control Letters, Special Issue on Hybrid Systems.*

[10] P. Kopke, T. Henzinger, A. Puri and P. Varaiya, “What's Decidable About Hybrid Automata?”, *27th Annual ACM Symposium on Theory of Computing* (STOCS), 1995, p.372-382.

[11] E.A. Lee, “Modeling Concurrent Real-time Processes Using Discrete Events,” Invited paper to *Annals of Software Engineering*, Special Volume on Real-Time Software Engineering, to appear.

[12] E. A. Lee and A. Sangiovanni-Vencentelli, “A Framework for Comparing Models of Computation,” *IEEE Transactions on CAD*, Vol. 17, No. 12, Dec. 1998.

[13] J. Liu, *Continuous-Time and Mixed-Signal Simulation in Ptolemy II*, UCB/ERL M98/74, University of California, Berkeley, CA 94720.

[14] J. Liu, X. Liu, T.J. Koo, B. Sinopoli, S. Sastry, and E.A. Lee, “A hierarchical hybrid system model and its simulation,” to appear in *IEEE Conference on Decision and Control* (CDC’99).

[15] N.A. Lynch, R. Segala, F.W. Vaandrager, and H.B. Weinberg, “Hybrid I/O automata,” *Hybrid System III*, LNCS 1066, Springer-Verlag, 1996, p.496-510.

[16] P. J. Mosterman, “An Overview of Hybrid Simulation Phenomena and Their Support by Simulation Packages,” *Hybrid Systems: Computation and Control* (HSCC’99), LNCS 1569, Springer, 1999, p.165-177.

[17] R. Orfali and D. Harkey, *Client/Server Programming with JAVA and CORBA*, Wiley Computer Publishing, 1997.

[18] R. A. Saleh and A. R. Newton, *Mixed-Mode Simulation*, Kluwer Academic Publishers, 1990.

[19] D.A. Schmidt, *Denotational Semantics: A Methodology for Language Development*, Allyn and Bacon, Newton, MA, 1986.

[20] Claire Tomlin, *Hybrid Control of Air Traffic Management Systems*, Ph.D. Thesis, Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, 1998.

[21] P. Varaiya, “Smart cars on smart roads: problems of control,” *IEEE Trans. on Automatic Control*, vol.38, (no.2), Feb. 1993. p.195-207.

[22] T*he Common Object Request Broker: Architecture and Specification* (formal/99-10-07 (CORBA 2.3.1)), Object Management Group, Inc., 1999.

[23] *Simulink 3 User's Guide*, The Mathworks Inc., 1999.

[24] *Verilog-AMS Language Reference Manual*, Version 1.2, Open Verilog International, June 1998.

[25] *VHDL Language Reference Manual* (*Integrated with VHDL-AMS Changes*), IEEE Standard 1076.1-1999.

[26] The Objectspace, Inc. web page, "http://www. objectspace.com"

[27] The Ptolemy II web page, "http://ptolemy. eecs.berkeley.edu/ptII"

Qualifying Examination Proposal

Component-Based Hierarchical Modeling of Systems with Continuous and Discrete Dynamics

Jie Liu
Department of Electrical Engineering and Computer Science
University of California, Berkeley

December 3, 1999

1. Motivation

The increasing complexity of embedded system design often requires the designers to model systems with both continuous and discrete dynamics. Examples include mixed-signal systems that have both continuous and discrete parts and digital systems that interact with a continuous environment. Although each individual model may be relatively well-understood, the integration of heterogeneous models brings additional difficulties and complexities to the design.

This work focuses on the ordinary differential equation (ODE) based *continuous time* (CT) model and two kinds of discrete models, a timed one – the *discrete event* (DE) model, and an untimed one – the *finite state machine* (FSM) model. Following circuit design communities, we call the composition of CT and DE the *mixed-signal* model; following control and computation communities, we call the composition of CT and FSM the *hybrid system* model.

Mixing heterogeneous models to design complex systems is receiving more and more attention from both academic and industrial perspectives. SPLICE [18], a mixed-signal circuit design tool, tries to speed up the simulation by extending the SPICE-like ODE solver to embrace an event-driven simulation engine for digital parts. Unfortunately, it does not accurately detect events, which makes the tight feedback between analog and digital parts hard to handle. Simulink[1], originally a continuous-time control system design tool, has been enhanced to model sample-data systems and, to some extent, discrete-event systems [23]. The hardware description language communities, like VHDL and Verilog, extend and standardize the capability of modeling continuous dynamics in their previous discrete-event based languages [24, 25]. Both of these approaches attempt to come up with a unified model to capture semantically different components, and lack hierarchy for managing the complexity of a design.

One of the most noticeable efforts in mixing continuous and discrete dynamics is the study of hybrid systems [2]. A solid theoretical framework for modeling and analyzing hybrid systems is under rapid development [1, 4, 8, 10, 15], with applications in air traffic management, transportation systems, automotive, manufacturing systems, and electromechanical systems etc. [3, 6, 20, 21]. Nevertheless, a simulation tool with clean semantics and composability that leverages the theory of hybrid system is still under high demand [16].

Component-based models view the building blocks of a system to be components. As shown in Figure 1, components are “black boxes.” The interface of a component and the communication scheme among components are specified, while the contents of a component can be implemented with a different model. The component-based design provides a clean way to integrate different models by hierarchically composing heterogeneous components [5]. This hierarchical composition allows one to manage the complexity of a design by information hiding, and to enhance the possibility of design reuse.

1. A software package from Mathworks Inc.

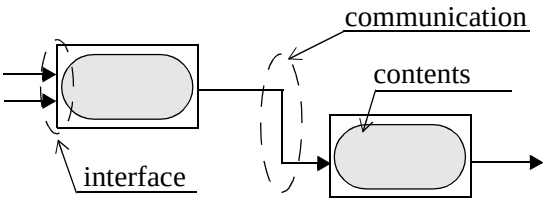

Figure 1. Heterogeneous components

My Thesis

I propose a component-based framework to model systems with both continuous and discrete dynamics. The framework cleanly integrates heterogeneous models by using hierarchical compositions to hide the implementation details of one component from other components, and keeping the components at the same level of hierarchy interacting in the same way. The signal conversions at the boundaries and the execution control among continuous and discrete components are studied. A correct and efficient simulation engine is built. I propose to scale up the framework in order to support modeling of distributable systems. I also expect to contribute via a mathematical understanding of the mixed-signal model.

2. Current Progress

2.1.Component-based model for continuous and discrete dynamics.

- *Continuous Time Model*

We consider an initial value problem of ODE for continuous time systems, and specify it using components, as shown in Figure 2. In this model, components communicate via piecewise continuous waveforms. And the components are continuous maps from input waveforms to output waveforms. The simulation of this model is to solve the ODE numerically with respect to the inputs at a discrete set of time points, and produce outputs at those points. The component-based model imposes no difficulty on numerical ODE solving methods. In fact, the evaluation of the *f* and *g* functions can be achieved by executing the corresponding components in their I/O topological order.

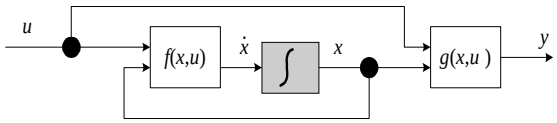

Figure 2. A component-based model for continuous time systems.

- *Discrete Event Model*

In the discrete event model, components communicate by a set of events that have discrete locations on the time line. An event has a time stamp and a value. A component, when executed, can consume input events and produce output events. The output events are required to be no earlier in time than the input events (this property is called *causality*). The simulation of this model utilizes a global event queue. When a component generates an output event, the event is placed in the queue, which sorts events by their time stamps. At each iteration of the simulation, events with the smallest time stamp will be dequeued, and their destination components will be executed.

- *Finite State Machine Model*

In the FSM model, as shown in Figure 3, there is a finite set of states (the bubbles), a finite set of events, an initial state, and transitions from states to states (the arcs). The set of events do not necessarily have a notion of time. A transition is associated with a trigger condition and an action. A trigger condition could be a predicate on input events, and an action might be producing output events. The execution of the system starts from the initial state. For each input event, if the trigger condition on a transition starting

from the current state is true, then the transition is taken and the action associated will be performed. The end state of the transition becomes the new current state.

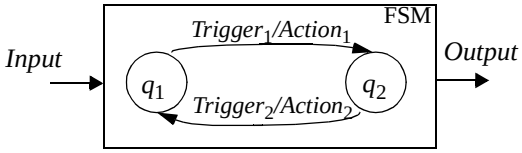

Figure 3. A finite state machine component.

2.2.Signal Conversions and Breakpoint Handling.

A fundamental issue for composing heterogeneous models is how to make a component implemented in one model to expose an interface of another model. Since the two kinds of dynamics have distinct types of signals, the conversion of signals is essential.

- *Event generation*

Event generation is to generate discrete events from piecewise continuous waveforms. We classify two types of events, *time events* and *state events*. The time stamps of time events are known beforehand, while the time stamps of state events depend on the value of the state variables in the CT system. In general, the time stamps of state events cannot be predicted accurately in advance. Special treatment has to be done in the process of ODE solving.

- *Waveform generation*

Waveform generation is to generate piecewise-continuous waveforms from discrete-event signals. This conversion is application dependent. In general, any extrapolations of previous events are reasonable.

- *Breakpoint handling in CT Simulation*

In order to handle event generation and inputs generated from discrete events, we define breakpoints in the continuous-time model. A *breakpoint* is a time point in the CT model when the right-hand side (RHS) of the ODE is not sufficiently smooth, or the output map is not continuous. The numerical ODE solver cannot cross breakpoints in one integration step since either the smooth-RHS assumption is violated or an event need to be produced. According to whether a breakpoint can be predicted in advance, we classify two kinds of breakpoints, *predictable* ones and *unpredictable* ones. For example, time events and unsmoothness in input signals are predictable, while state events and unsmoothness in state variables are unpredictable. Predictable breakpoints can be stored in a table and handled more efficiently. The numerical integration step sizes are now controlled based on three factors:

•*Error control*. This reflects the trade-off between speed and accuracy of a simulation. In general, for a given ODE solving method, a smaller step size means a more accurate result. But it also means more function evaluations and long simulation time.

•*Convergence*. Implicit numerical methods use fixed-point iteration or Newton iteration to solve the induced algebraic equations. Choosing smaller step size may help improve the initial guess.

•*Breakpoints*. Before each integration step, the breakpoint table is queried, and the intent step size (adjusted from the first two factors) may be reduced so that it does not cross a predictable breakpoint. Unpredictable breakpoints are handled by querying components after each integration step. An unpredictable breakpoint is iteratively located within an error tolerance, before the integration continues.

These techniques allow discrete components to expose continuous interface, as shown in Figure 4, and vice versa, as in Figure 5. The breakpoint handling mechanisms built in the continuous-time model suggests that the simulation of discrete dynamics does not have to handle continuous signals.

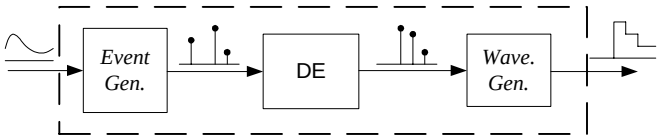

Figure 4. A DE component in a CT model.

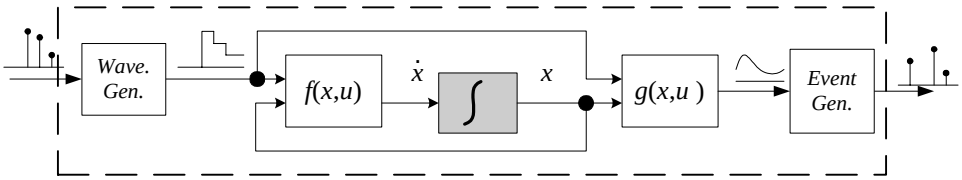

Figure 5. A CT component in a DE model.

2.3.Executing Heterogeneous Components

The execution control of heterogeneous components is critical for a correct and efficient simulation engine. Both CT and DE are timed models. There will be multiple time variables in different components. In our hierarchical composition of models, we call the time at the highest level of hierarchy the "global time," and the time maintained within a component the "local time."

- *DE inside CT*

Since time advances monotonically in CT and events are generated chronologically, the DE component will receive input events monotonically in time. In addition, a composition of causal DE components is causal [11], so the time stamps of output events from a DE component are always greater than or equal to those of the corresponding input events. Thus, from the CT system point of view, the events (breakpoints) produced by a DE component are always predictable.

- *CT inside DE*

When a CT component is contained in a DE system, as shown in Figure 5, the CT component is required to be causal, like all other components in the DE system. This suggests that the local time of the CT component should be greater than or equal to the global time, whenever it is executed [13].

This ahead-of-time execution implies that the CT component should be able to remember its past states and be ready to rollback if the input event is earlier than the local time. The state it needs to remember is the state of the component after it has processed an input event. Consequently, the CT component should not emit detected events to the outside DE system before the global time reaches the event time. Instead, it should request an execution from the DE system at the event time, and wait until its safe to emit it.

- *CT-FSM-CT*

A hierarchical composition of FSM and CT is shown in Figure 6. Although FSM is an untimed model,

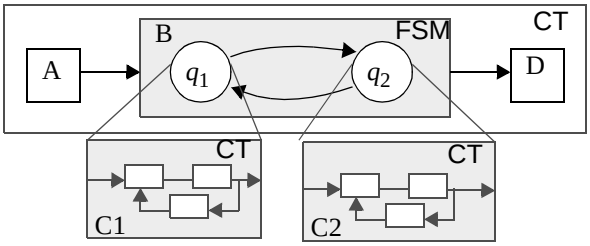

Figure 6. A hierarchical hybrid system.

its composition with a timed model requires it to transfer the notion of time from its external model to its internal model. A CT component, by adopting the event generation technique, can have both continuous and discrete signals as its outputs. The FSM may use predicates on them to build trigger conditions. Actions associated with transitions are usually reset to the initial conditions of integrators in the destination state.

During continuous evolution, the system is simulated as a CT system where the FSM is replaced by the continuous component of its current state. After each time point of CT simulation, the triggers on the transitions starting from the current FSM state are evaluated. If a trigger is enabled, the FSM makes the corresponding transition. The continuous dynamics of the destination state is initialized by the action on the transition. The simulation continues with the transition time treated as a breakpoint.

Currently, the framework for component-based modeling of systems with continuous and discrete dynamics has been built in the heterogeneous modeling and design environment, Ptolemy II [7]. The techniques discussed in the previous sections have been implemented. Several examples have been built to demonstrate the correct simulation results from using these techniques [27, 14].

3. Future Work

I intend to extend the current achievements in two directions, a denotational semantics for the mixed-signal model and a distributable component framework for further scaling up the component model.

3.1.Denotational Semantics for the Mixed-signal Model

Compared to the relative maturity of hybrid system theory, the composition of CT and DE has not been rigorously studied. A denotational semantics [19] defines the behavior of a model by the mathematical relationship on the signals. The study of such semantics for mixed-signal systems may give a deep insight on the properties like determinism and lack of Zeno phenomena (infinitely many events in a finite time interval), as well as a theoretical guide on the implementation of simulators.

The mathematical framework I want to use is the tagged-signal model developed by Lee and Sangiovanni-Vincentelli [12]. The model has been shown to be effective in studying the denotational semantics of DE models [11].

The first question I am trying to answer is related to determinism. Although both DE models and CT models can individually be shown to have a unique behavior under simple conditions (using Cantor metric and L^{∞} space, respectively), the conditions for mixed-signal model may not be trivial. A reason is that the Cantor metric is a metric on time while the $\| \|_{\infty}$ is a metric on values. I would like to come up with conditions that make a composition of continuous-time components (together with event generation and waveform generation processes) to be causal as a discrete-event component, and conditions that make a composition of discrete-event components (together with event generation and waveform generation processes) to be Lipschitz as a continuous-time component.

For the Zeno phenomena, I expect that conditions on the event generation, waveform generation, and system dynamics may help avoid it. The study that Johansson et al. have done on Zeno hybrid automata [9] might be helpful.

3.2.Distributable components

The component-based model has the advantage that there is no restrictions that the components be physically related. It provides a way to protect intellectual property more effectively. In addition, complex embedded systems usually have communication subsystems such that the components interact remotely. A design environment that has the capability to model distributed components is highly desirable.

CORBA [22] and similar middleware techniques [17, 26] provide a promising framework to make the distribution of components transparent to designers. To support them in a design environment, the key questions to be answered are the granularity of distribution, the communication styles among components, and the information to be shared. I intend to embrace the middleware concept in the component model, explore different options of design choices, and provide a practical implementation of it.

4. Conclusion

I expect to contribute a distributable component-based modeling technique for systems with continuous and discrete dynamics, a mathematical framework for it, a correct and efficient simulation strategy, and a practical implementation of it within a component-based multi-model design environment.

References

[1] R. Alur, T. Henzinger, G. Lafferriere, and G.J. Pappas, “Discrete Abstractions of Hybrid Systems,” Submitted to *the Proceedings of the IEEE*, 1999.

[2] P. Antsaklis, W. Kohn, A. Nerode, and S. Sastry (ed.), *Hybrid Systems II*, LNCS 999, Springer 1995.

[3] A. Balluchi, M.D. Di Benedetto, C. Pinello, C. Rossi, and A. Sangiovanni-Vincentelli, “Hybrid control in automotive applications: the cut-off control,” *Automatica*, vol.35, (no.3), March 1999, p.519-535.

[4] S. Bornot, J. Sifakis, and S. Tripakis, “Modeling Urgency in Timed Systems,” *Compositionality: the significant difference*, COMPOS'97, LNCS 1536, Springer 1997.

[5] W.-T. Chang, A. Kalavade and E. A. Lee, “Effective Heterogeneous Design and Cosimulation,” chapter in *Hardware/Software Co-design*, G. DeMicheli and M. Sami, eds., NATO ASI Series Vol. 310, Kluwer Academic Publishers,1996.

[6] C. Cloet, M. Krucinski, R. Horowitz, and M. Tomizuka, “A hybrid control scheme for a copier paperpath,” *Proceedings of the 1999 American Control Conference*, San Diego, CA, June 1999, p.2114-2118.

[7] J. Davis etc., “Ptolemy II: Heterogeneous Concurrent Modeling and Design in Java,” UCB/ERL M99/40, University of California, Berkeley, CA 94720.

[8] T. A. Henzinger, “The theory of hybrid automata,” *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science* (LICS 1996), p.278-292.

[9] K.H. Johansson, M. Egerstedt, J. Lygeros, and S. Sastry, “On the Regularization of Zeno Hybrid Automata,” submitted to *Systems Control Letters, Special Issue on Hybrid Systems.*

[10] P. Kopke, T. Henzinger, A. Puri and P. Varaiya, “What's Decidable About Hybrid Automata?”, *27th Annual ACM Symposium on Theory of Computing* (STOCS), 1995, p.372-382.

[11] E.A. Lee, “Modeling Concurrent Real-time Processes Using Discrete Events,” Invited paper to *Annals of Software Engineering*, Special Volume on Real-Time Software Engineering, to appear.

[12] E. A. Lee and A. Sangiovanni-Vencentelli, “A Framework for Comparing Models of Computation,” *IEEE Transactions on CAD*, Vol. 17, No. 12, Dec. 1998.

[13] J. Liu, *Continuous-Time and Mixed-Signal Simulation in Ptolemy II*, UCB/ERL M98/74, University of California, Berkeley, CA 94720.

[14] J. Liu, X. Liu, T.J. Koo, B. Sinopoli, S. Sastry, and E.A. Lee, “A hierarchical hybrid system model and its simulation,” to appear in *IEEE Conference on Decision and Control* (CDC’99).

[15] N.A. Lynch, R. Segala, F.W. Vaandrager, and H.B. Weinberg, “Hybrid I/O automata,” *Hybrid System III*, LNCS 1066, Springer-Verlag, 1996, p.496-510.

[16] P. J. Mosterman, “An Overview of Hybrid Simulation Phenomena and Their Support by Simulation Packages,” *Hybrid Systems: Computation and Control* (HSCC’99), LNCS 1569, Springer, 1999, p.165-177.

[17] R. Orfali and D. Harkey, *Client/Server Programming with JAVA and CORBA*, Wiley Computer Publishing, 1997.

[18] R. A. Saleh and A. R. Newton, *Mixed-Mode Simulation*, Kluwer Academic Publishers, 1990.

[19] D.A. Schmidt, *Denotational Semantics: A Methodology for Language Development*, Allyn and Bacon, Newton, MA, 1986.

[20] Claire Tomlin, *Hybrid Control of Air Traffic Management Systems*, Ph.D. Thesis, Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, 1998.

[21] P. Varaiya, “Smart cars on smart roads: problems of control,” *IEEE Trans. on Automatic Control*, vol.38, (no.2), Feb. 1993. p.195-207.

[22] T*he Common Object Request Broker: Architecture and Specification* (formal/99-10-07 (CORBA 2.3.1)), Object Management Group, Inc., 1999.

[23] *Simulink 3 User's Guide*, The Mathworks Inc., 1999.

[24] *Verilog-AMS Language Reference Manual*, Version 1.2, Open Verilog International, June 1998.

[25] *VHDL Language Reference Manual* (*Integrated with VHDL-AMS Changes*), IEEE Standard 1076.1-1999.

[26] The Objectspace, Inc. web page, "http://www. objectspace.com"

[27] The Ptolemy II web page, "http://ptolemy. eecs.berkeley.edu/ptII"

Qualifying Examination Proposal

Component-Based Hierarchical Modeling of Systems with Continuous and Discrete Dynamics

Jie Liu
Department of Electrical Engineering and Computer Science
University of California, Berkeley

December 3, 1999

1. Motivation

The increasing complexity of embedded system design often requires the designers to model systems with both continuous and discrete dynamics. Examples include mixed-signal systems that have both continuous and discrete parts and digital systems that interact with a continuous environment. Although each individual model may be relatively well-understood, the integration of heterogeneous models brings additional difficulties and complexities to the design.

This work focuses on the ordinary differential equation (ODE) based *continuous time* (CT) model and two kinds of discrete models, a timed one – the *discrete event* (DE) model, and an untimed one – the *finite state machine* (FSM) model. Following circuit design communities, we call the composition of CT and DE the *mixed-signal* model; following control and computation communities, we call the composition of CT and FSM the *hybrid system* model.

Mixing heterogeneous models to design complex systems is receiving more and more attention from both academic and industrial perspectives. SPLICE [18], a mixed-signal circuit design tool, tries to speed up the simulation by extending the SPICE-like ODE solver to embrace an event-driven simulation engine for digital parts. Unfortunately, it does not accurately detect events, which makes the tight feedback between analog and digital parts hard to handle. Simulink[1], originally a continuous-time control system design tool, has been enhanced to model sample-data systems and, to some extent, discrete-event systems [23]. The hardware description language communities, like VHDL and Verilog, extend and standardize the capability of modeling continuous dynamics in their previous discrete-event based languages [24, 25]. Both of these approaches attempt to come up with a unified model to capture semantically different components, and lack hierarchy for managing the complexity of a design.

One of the most noticeable efforts in mixing continuous and discrete dynamics is the study of hybrid systems [2]. A solid theoretical framework for modeling and analyzing hybrid systems is under rapid development [1, 4, 8, 10, 15], with applications in air traffic management, transportation systems, automotive, manufacturing systems, and electromechanical systems etc. [3, 6, 20, 21]. Nevertheless, a simulation tool with clean semantics and composability that leverages the theory of hybrid system is still under high demand [16].

Component-based models view the building blocks of a system to be components. As shown in Figure 1, components are “black boxes.” The interface of a component and the communication scheme among components are specified, while the contents of a component can be implemented with a different model. The component-based design provides a clean way to integrate different models by hierarchically composing heterogeneous components [5]. This hierarchical composition allows one to manage the complexity of a design by information hiding, and to enhance the possibility of design reuse.

1. A software package from Mathworks Inc.

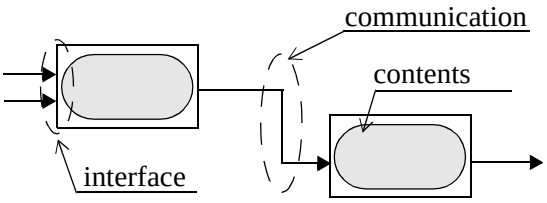

Figure 1. Heterogeneous components

My Thesis

I propose a component-based framework to model systems with both continuous and discrete dynamics. The framework cleanly integrates heterogeneous models by using hierarchical compositions to hide the implementation details of one component from other components, and keeping the components at the same level of hierarchy interacting in the same way. The signal conversions at the boundaries and the execution control among continuous and discrete components are studied. A correct and efficient simulation engine is built. I propose to scale up the framework in order to support modeling of distributable systems. I also expect to contribute via a mathematical understanding of the mixed-signal model.

2. Current Progress

2.1.Component-based model for continuous and discrete dynamics.

- *Continuous Time Model*

We consider an initial value problem of ODE for continuous time systems, and specify it using components, as shown in Figure 2. In this model, components communicate via piecewise continuous waveforms. And the components are continuous maps from input waveforms to output waveforms. The simulation of this model is to solve the ODE numerically with respect to the inputs at a discrete set of time points, and produce outputs at those points. The component-based model imposes no difficulty on numerical ODE solving methods. In fact, the evaluation of the *f* and *g* functions can be achieved by executing the corresponding components in their I/O topological order.

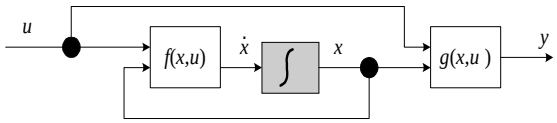

Figure 2. A component-based model for continuous time systems.

- *Discrete Event Model*

In the discrete event model, components communicate by a set of events that have discrete locations on the time line. An event has a time stamp and a value. A component, when executed, can consume input events and produce output events. The output events are required to be no earlier in time than the input events (this property is called *causality*). The simulation of this model utilizes a global event queue. When a component generates an output event, the event is placed in the queue, which sorts events by their time stamps. At each iteration of the simulation, events with the smallest time stamp will be dequeued, and their destination components will be executed.

- *Finite State Machine Model*

In the FSM model, as shown in Figure 3, there is a finite set of states (the bubbles), a finite set of events, an initial state, and transitions from states to states (the arcs). The set of events do not necessarily have a notion of time. A transition is associated with a trigger condition and an action. A trigger condition could be a predicate on input events, and an action might be producing output events. The execution of the system starts from the initial state. For each input event, if the trigger condition on a transition starting

from the current state is true, then the transition is taken and the action associated will be performed. The end state of the transition becomes the new current state.

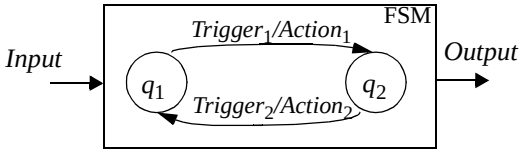

Figure 3. A finite state machine component.

2.2.Signal Conversions and Breakpoint Handling.

A fundamental issue for composing heterogeneous models is how to make a component implemented in one model to expose an interface of another model. Since the two kinds of dynamics have distinct types of signals, the conversion of signals is essential.

- *Event generation*

Event generation is to generate discrete events from piecewise continuous waveforms. We classify two types of events, *time events* and *state events*. The time stamps of time events are known beforehand, while the time stamps of state events depend on the value of the state variables in the CT system. In general, the time stamps of state events cannot be predicted accurately in advance. Special treatment has to be done in the process of ODE solving.

- *Waveform generation*

Waveform generation is to generate piecewise-continuous waveforms from discrete-event signals. This conversion is application dependent. In general, any extrapolations of previous events are reasonable.

- *Breakpoint handling in CT Simulation*

In order to handle event generation and inputs generated from discrete events, we define breakpoints in the continuous-time model. A *breakpoint* is a time point in the CT model when the right-hand side (RHS) of the ODE is not sufficiently smooth, or the output map is not continuous. The numerical ODE solver cannot cross breakpoints in one integration step since either the smooth-RHS assumption is violated or an event need to be produced. According to whether a breakpoint can be predicted in advance, we classify two kinds of breakpoints, *predictable* ones and *unpredictable* ones. For example, time events and unsmoothness in input signals are predictable, while state events and unsmoothness in state variables are unpredictable. Predictable breakpoints can be stored in a table and handled more efficiently. The numerical integration step sizes are now controlled based on three factors:

•*Error control*. This reflects the trade-off between speed and accuracy of a simulation. In general, for a given ODE solving method, a smaller step size means a more accurate result. But it also means more function evaluations and long simulation time.

•*Convergence*. Implicit numerical methods use fixed-point iteration or Newton iteration to solve the induced algebraic equations. Choosing smaller step size may help improve the initial guess.

•*Breakpoints*. Before each integration step, the breakpoint table is queried, and the intent step size (adjusted from the first two factors) may be reduced so that it does not cross a predictable breakpoint. Unpredictable breakpoints are handled by querying components after each integration step. An unpredictable breakpoint is iteratively located within an error tolerance, before the integration continues.

These techniques allow discrete components to expose continuous interface, as shown in Figure 4, and vice versa, as in Figure 5. The breakpoint handling mechanisms built in the continuous-time model suggests that the simulation of discrete dynamics does not have to handle continuous signals.

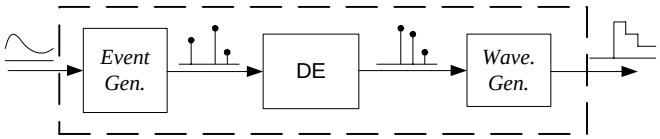

Figure 4. A DE component in a CT model.

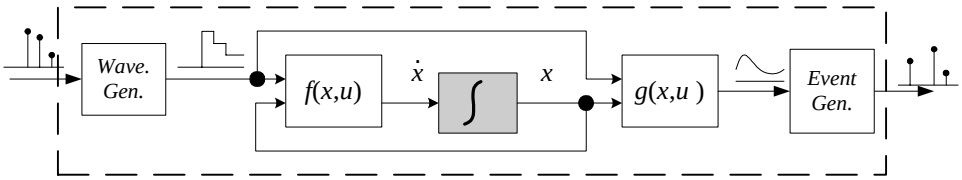

Figure 5. A CT component in a DE model.

2.3.Executing Heterogeneous Components

The execution control of heterogeneous components is critical for a correct and efficient simulation engine. Both CT and DE are timed models. There will be multiple time variables in different components. In our hierarchical composition of models, we call the time at the highest level of hierarchy the "global time," and the time maintained within a component the "local time."

- *DE inside CT*

Since time advances monotonically in CT and events are generated chronologically, the DE component will receive input events monotonically in time. In addition, a composition of causal DE components is causal [11], so the time stamps of output events from a DE component are always greater than or equal to those of the corresponding input events. Thus, from the CT system point of view, the events (breakpoints) produced by a DE component are always predictable.

- *CT inside DE*

When a CT component is contained in a DE system, as shown in Figure 5, the CT component is required to be causal, like all other components in the DE system. This suggests that the local time of the CT component should be greater than or equal to the global time, whenever it is executed [13].

This ahead-of-time execution implies that the CT component should be able to remember its past states and be ready to rollback if the input event is earlier than the local time. The state it needs to remember is the state of the component after it has processed an input event. Consequently, the CT component should not emit detected events to the outside DE system before the global time reaches the event time. Instead, it should request an execution from the DE system at the event time, and wait until its safe to emit it.

- *CT-FSM-CT*

A hierarchical composition of FSM and CT is shown in Figure 6. Although FSM is an untimed model,

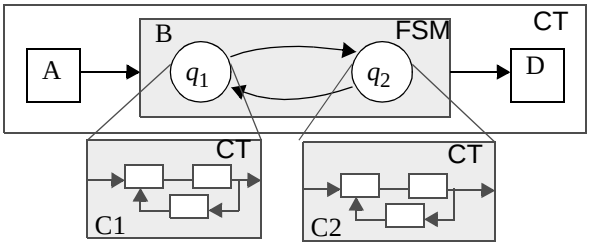

Figure 6. A hierarchical hybrid system.

its composition with a timed model requires it to transfer the notion of time from its external model to its internal model. A CT component, by adopting the event generation technique, can have both continuous and discrete signals as its outputs. The FSM may use predicates on them to build trigger conditions. Actions associated with transitions are usually reset to the initial conditions of integrators in the destination state.

During continuous evolution, the system is simulated as a CT system where the FSM is replaced by the continuous component of its current state. After each time point of CT simulation, the triggers on the transitions starting from the current FSM state are evaluated. If a trigger is enabled, the FSM makes the corresponding transition. The continuous dynamics of the destination state is initialized by the action on the transition. The simulation continues with the transition time treated as a breakpoint.

Currently, the framework for component-based modeling of systems with continuous and discrete dynamics has been built in the heterogeneous modeling and design environment, Ptolemy II [7]. The techniques discussed in the previous sections have been implemented. Several examples have been built to demonstrate the correct simulation results from using these techniques [27, 14].

3. Future Work

I intend to extend the current achievements in two directions, a denotational semantics for the mixed-signal model and a distributable component framework for further scaling up the component model.

3.1.Denotational Semantics for the Mixed-signal Model

Compared to the relative maturity of hybrid system theory, the composition of CT and DE has not been rigorously studied. A denotational semantics [19] defines the behavior of a model by the mathematical relationship on the signals. The study of such semantics for mixed-signal systems may give a deep insight on the properties like determinism and lack of Zeno phenomena (infinitely many events in a finite time interval), as well as a theoretical guide on the implementation of simulators.

The mathematical framework I want to use is the tagged-signal model developed by Lee and Sangiovanni-Vincentelli [12]. The model has been shown to be effective in studying the denotational semantics of DE models [11].

The first question I am trying to answer is related to determinism. Although both DE models and CT models can individually be shown to have a unique behavior under simple conditions (using Cantor metric and L^{∞} space, respectively), the conditions for mixed-signal model may not be trivial. A reason is that the Cantor metric is a metric on time while the $\| \|_{\infty}$ is a metric on values. I would like to come up with conditions that make a composition of continuous-time components (together with event generation and waveform generation processes) to be causal as a discrete-event component, and conditions that make a composition of discrete-event components (together with event generation and waveform generation processes) to be Lipschitz as a continuous-time component.

For the Zeno phenomena, I expect that conditions on the event generation, waveform generation, and system dynamics may help avoid it. The study that Johansson et al. have done on Zeno hybrid automata [9] might be helpful.

3.2.Distributable components

The component-based model has the advantage that there is no restrictions that the components be physically related. It provides a way to protect intellectual property more effectively. In addition, complex embedded systems usually have communication subsystems such that the components interact remotely. A design environment that has the capability to model distributed components is highly desirable.

CORBA [22] and similar middleware techniques [17, 26] provide a promising framework to make the distribution of components transparent to designers. To support them in a design environment, the key questions to be answered are the granularity of distribution, the communication styles among components, and the information to be shared. I intend to embrace the middleware concept in the component model, explore different options of design choices, and provide a practical implementation of it.

4. Conclusion

I expect to contribute a distributable component-based modeling technique for systems with continuous and discrete dynamics, a mathematical framework for it, a correct and efficient simulation strategy, and a practical implementation of it within a component-based multi-model design environment.

References

[1] R. Alur, T. Henzinger, G. Lafferriere, and G.J. Pappas, “Discrete Abstractions of Hybrid Systems,” Submitted to *the Proceedings of the IEEE*, 1999.

[2] P. Antsaklis, W. Kohn, A. Nerode, and S. Sastry (ed.), *Hybrid Systems II*, LNCS 999, Springer 1995.

[3] A. Balluchi, M.D. Di Benedetto, C. Pinello, C. Rossi, and A. Sangiovanni-Vincentelli, “Hybrid control in automotive applications: the cut-off control,” *Automatica*, vol.35, (no.3), March 1999, p.519-535.

[4] S. Bornot, J. Sifakis, and S. Tripakis, “Modeling Urgency in Timed Systems,” *Compositionality: the significant difference*, COMPOS'97, LNCS 1536, Springer 1997.

[5] W.-T. Chang, A. Kalavade and E. A. Lee, “Effective Heterogeneous Design and Cosimulation,” chapter in *Hardware/Software Co-design*, G. DeMicheli and M. Sami, eds., NATO ASI Series Vol. 310, Kluwer Academic Publishers,1996.

[6] C. Cloet, M. Krucinski, R. Horowitz, and M. Tomizuka, “A hybrid control scheme for a copier paperpath,” *Proceedings of the 1999 American Control Conference*, San Diego, CA, June 1999, p.2114-2118.

[7] J. Davis etc., “Ptolemy II: Heterogeneous Concurrent Modeling and Design in Java,” UCB/ERL M99/40, University of California, Berkeley, CA 94720.

[8] T. A. Henzinger, “The theory of hybrid automata,” *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science* (LICS 1996), p.278-292.

[9] K.H. Johansson, M. Egerstedt, J. Lygeros, and S. Sastry, “On the Regularization of Zeno Hybrid Automata,” submitted to *Systems Control Letters, Special Issue on Hybrid Systems.*

[10] P. Kopke, T. Henzinger, A. Puri and P. Varaiya, “What's Decidable About Hybrid Automata?”, *27th Annual ACM Symposium on Theory of Computing* (STOCS), 1995, p.372-382.

[11] E.A. Lee, “Modeling Concurrent Real-time Processes Using Discrete Events,” Invited paper to *Annals of Software Engineering*, Special Volume on Real-Time Software Engineering, to appear.

[12] E. A. Lee and A. Sangiovanni-Vencentelli, “A Framework for Comparing Models of Computation,” *IEEE Transactions on CAD*, Vol. 17, No. 12, Dec. 1998.

[13] J. Liu, *Continuous-Time and Mixed-Signal Simulation in Ptolemy II*, UCB/ERL M98/74, University of California, Berkeley, CA 94720.

[14] J. Liu, X. Liu, T.J. Koo, B. Sinopoli, S. Sastry, and E.A. Lee, “A hierarchical hybrid system model and its simulation,” to appear in *IEEE Conference on Decision and Control* (CDC’99).

[15] N.A. Lynch, R. Segala, F.W. Vaandrager, and H.B. Weinberg, “Hybrid I/O automata,” *Hybrid System III*, LNCS 1066, Springer-Verlag, 1996, p.496-510.

[16] P. J. Mosterman, “An Overview of Hybrid Simulation Phenomena and Their Support by Simulation Packages,” *Hybrid Systems: Computation and Control* (HSCC’99), LNCS 1569, Springer, 1999, p.165-177.

[17] R. Orfali and D. Harkey, *Client/Server Programming with JAVA and CORBA*, Wiley Computer Publishing, 1997.

[18] R. A. Saleh and A. R. Newton, *Mixed-Mode Simulation*, Kluwer Academic Publishers, 1990.

[19] D.A. Schmidt, *Denotational Semantics: A Methodology for Language Development*, Allyn and Bacon, Newton, MA, 1986.

[20] Claire Tomlin, *Hybrid Control of Air Traffic Management Systems*, Ph.D. Thesis, Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, 1998.

[21] P. Varaiya, “Smart cars on smart roads: problems of control,” *IEEE Trans. on Automatic Control*, vol.38, (no.2), Feb. 1993. p.195-207.

[22] T*he Common Object Request Broker: Architecture and Specification* (formal/99-10-07 (CORBA 2.3.1)), Object Management Group, Inc., 1999.

[23] *Simulink 3 User's Guide*, The Mathworks Inc., 1999.

[24] *Verilog-AMS Language Reference Manual*, Version 1.2, Open Verilog International, June 1998.

[25] *VHDL Language Reference Manual* (*Integrated with VHDL-AMS Changes*), IEEE Standard 1076.1-1999.

[26] The Objectspace, Inc. web page, "http://www. objectspace.com"

[27] The Ptolemy II web page, "http://ptolemy. eecs.berkeley.edu/ptII"

Qualifying Examination Proposal

Component-Based Hierarchical Modeling of Systems with Continuous and Discrete Dynamics

Jie Liu
Department of Electrical Engineering and Computer Science
University of California, Berkeley

December 3, 1999

1. Motivation

The increasing complexity of embedded system design often requires the designers to model systems with both continuous and discrete dynamics. Examples include mixed-signal systems that have both continuous and discrete parts and digital systems that interact with a continuous environment. Although each individual model may be relatively well-understood, the integration of heterogeneous models brings additional difficulties and complexities to the design.

This work focuses on the ordinary differential equation (ODE) based *continuous time* (CT) model and two kinds of discrete models, a timed one – the *discrete event* (DE) model, and an untimed one – the *finite state machine* (FSM) model. Following circuit design communities, we call the composition of CT and DE the *mixed-signal* model; following control and computation communities, we call the composition of CT and FSM the *hybrid system* model.

Mixing heterogeneous models to design complex systems is receiving more and more attention from both academic and industrial perspectives. SPLICE [18], a mixed-signal circuit design tool, tries to speed up the simulation by extending the SPICE-like ODE solver to embrace an event-driven simulation engine for digital parts. Unfortunately, it does not accurately detect events, which makes the tight feedback between analog and digital parts hard to handle. Simulink[1], originally a continuous-time control system design tool, has been enhanced to model sample-data systems and, to some extent, discrete-event systems [23]. The hardware description language communities, like VHDL and Verilog, extend and standardize the capability of modeling continuous dynamics in their previous discrete-event based languages [24, 25]. Both of these approaches attempt to come up with a unified model to capture semantically different components, and lack hierarchy for managing the complexity of a design.

One of the most noticeable efforts in mixing continuous and discrete dynamics is the study of hybrid systems [2]. A solid theoretical framework for modeling and analyzing hybrid systems is under rapid development [1, 4, 8, 10, 15], with applications in air traffic management, transportation systems, automotive, manufacturing systems, and electromechanical systems etc. [3, 6, 20, 21]. Nevertheless, a simulation tool with clean semantics and composability that leverages the theory of hybrid system is still under high demand [16].

Component-based models view the building blocks of a system to be components. As shown in Figure 1, components are “black boxes.” The interface of a component and the communication scheme among components are specified, while the contents of a component can be implemented with a different model. The component-based design provides a clean way to integrate different models by hierarchically composing heterogeneous components [5]. This hierarchical composition allows one to manage the complexity of a design by information hiding, and to enhance the possibility of design reuse.

1. A software package from Mathworks Inc.

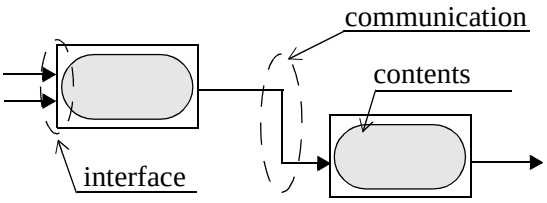

Figure 1. Heterogeneous components

My Thesis

I propose a component-based framework to model systems with both continuous and discrete dynamics. The framework cleanly integrates heterogeneous models by using hierarchical compositions to hide the implementation details of one component from other components, and keeping the components at the same level of hierarchy interacting in the same way. The signal conversions at the boundaries and the execution control among continuous and discrete components are studied. A correct and efficient simulation engine is built. I propose to scale up the framework in order to support modeling of distributable systems. I also expect to contribute via a mathematical understanding of the mixed-signal model.

2. Current Progress

2.1.Component-based model for continuous and discrete dynamics.

- *Continuous Time Model*

We consider an initial value problem of ODE for continuous time systems, and specify it using components, as shown in Figure 2. In this model, components communicate via piecewise continuous waveforms. And the components are continuous maps from input waveforms to output waveforms. The simulation of this model is to solve the ODE numerically with respect to the inputs at a discrete set of time points, and produce outputs at those points. The component-based model imposes no difficulty on numerical ODE solving methods. In fact, the evaluation of the *f* and *g* functions can be achieved by executing the corresponding components in their I/O topological order.

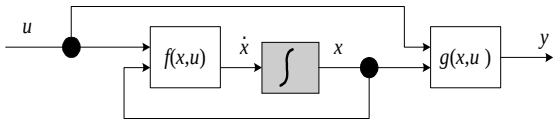

Figure 2. A component-based model for continuous time systems.

- *Discrete Event Model*

In the discrete event model, components communicate by a set of events that have discrete locations on the time line. An event has a time stamp and a value. A component, when executed, can consume input events and produce output events. The output events are required to be no earlier in time than the input events (this property is called *causality*). The simulation of this model utilizes a global event queue. When a component generates an output event, the event is placed in the queue, which sorts events by their time stamps. At each iteration of the simulation, events with the smallest time stamp will be dequeued, and their destination components will be executed.

- *Finite State Machine Model*

In the FSM model, as shown in Figure 3, there is a finite set of states (the bubbles), a finite set of events, an initial state, and transitions from states to states (the arcs). The set of events do not necessarily have a notion of time. A transition is associated with a trigger condition and an action. A trigger condition could be a predicate on input events, and an action might be producing output events. The execution of the system starts from the initial state. For each input event, if the trigger condition on a transition starting

from the current state is true, then the transition is taken and the action associated will be performed. The end state of the transition becomes the new current state.

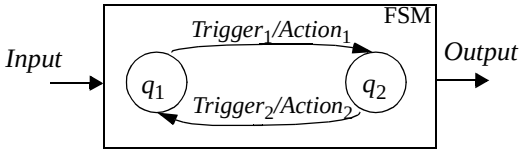

Figure 3. A finite state machine component.

2.2.Signal Conversions and Breakpoint Handling.

A fundamental issue for composing heterogeneous models is how to make a component implemented in one model to expose an interface of another model. Since the two kinds of dynamics have distinct types of signals, the conversion of signals is essential.

- *Event generation*

Event generation is to generate discrete events from piecewise continuous waveforms. We classify two types of events, *time events* and *state events*. The time stamps of time events are known beforehand, while the time stamps of state events depend on the value of the state variables in the CT system. In general, the time stamps of state events cannot be predicted accurately in advance. Special treatment has to be done in the process of ODE solving.

- *Waveform generation*

Waveform generation is to generate piecewise-continuous waveforms from discrete-event signals. This conversion is application dependent. In general, any extrapolations of previous events are reasonable.

- *Breakpoint handling in CT Simulation*

In order to handle event generation and inputs generated from discrete events, we define breakpoints in the continuous-time model. A *breakpoint* is a time point in the CT model when the right-hand side (RHS) of the ODE is not sufficiently smooth, or the output map is not continuous. The numerical ODE solver cannot cross breakpoints in one integration step since either the smooth-RHS assumption is violated or an event need to be produced. According to whether a breakpoint can be predicted in advance, we classify two kinds of breakpoints, *predictable* ones and *unpredictable* ones. For example, time events and unsmoothness in input signals are predictable, while state events and unsmoothness in state variables are unpredictable. Predictable breakpoints can be stored in a table and handled more efficiently. The numerical integration step sizes are now controlled based on three factors:

•*Error control*. This reflects the trade-off between speed and accuracy of a simulation. In general, for a given ODE solving method, a smaller step size means a more accurate result. But it also means more function evaluations and long simulation time.

•*Convergence*. Implicit numerical methods use fixed-point iteration or Newton iteration to solve the induced algebraic equations. Choosing smaller step size may help improve the initial guess.

•*Breakpoints*. Before each integration step, the breakpoint table is queried, and the intent step size (adjusted from the first two factors) may be reduced so that it does not cross a predictable breakpoint. Unpredictable breakpoints are handled by querying components after each integration step. An unpredictable breakpoint is iteratively located within an error tolerance, before the integration continues.

These techniques allow discrete components to expose continuous interface, as shown in Figure 4, and vice versa, as in Figure 5. The breakpoint handling mechanisms built in the continuous-time model suggests that the simulation of discrete dynamics does not have to handle continuous signals.

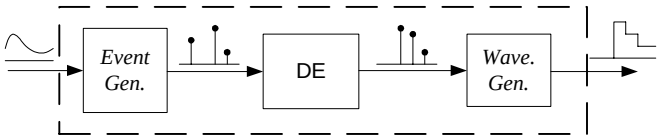

Figure 4. A DE component in a CT model.

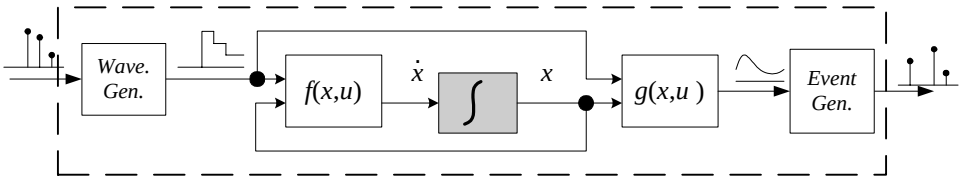

Figure 5. A CT component in a DE model.

2.3.Executing Heterogeneous Components

The execution control of heterogeneous components is critical for a correct and efficient simulation engine. Both CT and DE are timed models. There will be multiple time variables in different components. In our hierarchical composition of models, we call the time at the highest level of hierarchy the "global time," and the time maintained within a component the "local time."

- *DE inside CT*

Since time advances monotonically in CT and events are generated chronologically, the DE component will receive input events monotonically in time. In addition, a composition of causal DE components is causal [11], so the time stamps of output events from a DE component are always greater than or equal to those of the corresponding input events. Thus, from the CT system point of view, the events (breakpoints) produced by a DE component are always predictable.

- *CT inside DE*

When a CT component is contained in a DE system, as shown in Figure 5, the CT component is required to be causal, like all other components in the DE system. This suggests that the local time of the CT component should be greater than or equal to the global time, whenever it is executed [13].

This ahead-of-time execution implies that the CT component should be able to remember its past states and be ready to rollback if the input event is earlier than the local time. The state it needs to remember is the state of the component after it has processed an input event. Consequently, the CT component should not emit detected events to the outside DE system before the global time reaches the event time. Instead, it should request an execution from the DE system at the event time, and wait until its safe to emit it.

- *CT-FSM-CT*

A hierarchical composition of FSM and CT is shown in Figure 6. Although FSM is an untimed model,

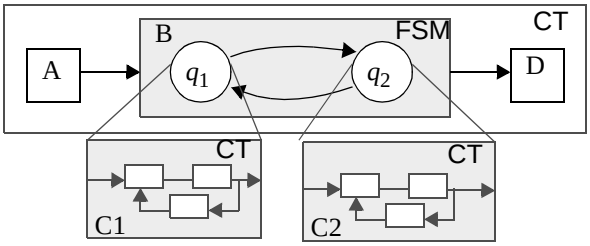

Figure 6. A hierarchical hybrid system.

its composition with a timed model requires it to transfer the notion of time from its external model to its internal model. A CT component, by adopting the event generation technique, can have both continuous and discrete signals as its outputs. The FSM may use predicates on them to build trigger conditions. Actions associated with transitions are usually reset to the initial conditions of integrators in the destination state.

During continuous evolution, the system is simulated as a CT system where the FSM is replaced by the continuous component of its current state. After each time point of CT simulation, the triggers on the transitions starting from the current FSM state are evaluated. If a trigger is enabled, the FSM makes the corresponding transition. The continuous dynamics of the destination state is initialized by the action on the transition. The simulation continues with the transition time treated as a breakpoint.

Currently, the framework for component-based modeling of systems with continuous and discrete dynamics has been built in the heterogeneous modeling and design environment, Ptolemy II [7]. The techniques discussed in the previous sections have been implemented. Several examples have been built to demonstrate the correct simulation results from using these techniques [27, 14].

3. Future Work

I intend to extend the current achievements in two directions, a denotational semantics for the mixed-signal model and a distributable component framework for further scaling up the component model.

3.1.Denotational Semantics for the Mixed-signal Model

Compared to the relative maturity of hybrid system theory, the composition of CT and DE has not been rigorously studied. A denotational semantics [19] defines the behavior of a model by the mathematical relationship on the signals. The study of such semantics for mixed-signal systems may give a deep insight on the properties like determinism and lack of Zeno phenomena (infinitely many events in a finite time interval), as well as a theoretical guide on the implementation of simulators.

The mathematical framework I want to use is the tagged-signal model developed by Lee and Sangiovanni-Vincentelli [12]. The model has been shown to be effective in studying the denotational semantics of DE models [11].

The first question I am trying to answer is related to determinism. Although both DE models and CT models can individually be shown to have a unique behavior under simple conditions (using Cantor metric and L^{∞} space, respectively), the conditions for mixed-signal model may not be trivial. A reason is that the Cantor metric is a metric on time while the $\| \|_{\infty}$ is a metric on values. I would like to come up with conditions that make a composition of continuous-time components (together with event generation and waveform generation processes) to be causal as a discrete-event component, and conditions that make a composition of discrete-event components (together with event generation and waveform generation processes) to be Lipschitz as a continuous-time component.

For the Zeno phenomena, I expect that conditions on the event generation, waveform generation, and system dynamics may help avoid it. The study that Johansson et al. have done on Zeno hybrid automata [9] might be helpful.

3.2.Distributable components

The component-based model has the advantage that there is no restrictions that the components be physically related. It provides a way to protect intellectual property more effectively. In addition, complex embedded systems usually have communication subsystems such that the components interact remotely. A design environment that has the capability to model distributed components is highly desirable.

CORBA [22] and similar middleware techniques [17, 26] provide a promising framework to make the distribution of components transparent to designers. To support them in a design environment, the key questions to be answered are the granularity of distribution, the communication styles among components, and the information to be shared. I intend to embrace the middleware concept in the component model, explore different options of design choices, and provide a practical implementation of it.

4. Conclusion

I expect to contribute a distributable component-based modeling technique for systems with continuous and discrete dynamics, a mathematical framework for it, a correct and efficient simulation strategy, and a practical implementation of it within a component-based multi-model design environment.

References

[1] R. Alur, T. Henzinger, G. Lafferriere, and G.J. Pappas, “Discrete Abstractions of Hybrid Systems,” Submitted to *the Proceedings of the IEEE*, 1999.

[2] P. Antsaklis, W. Kohn, A. Nerode, and S. Sastry (ed.), *Hybrid Systems II*, LNCS 999, Springer 1995.

[3] A. Balluchi, M.D. Di Benedetto, C. Pinello, C. Rossi, and A. Sangiovanni-Vincentelli, “Hybrid control in automotive applications: the cut-off control,” *Automatica*, vol.35, (no.3), March 1999, p.519-535.

[4] S. Bornot, J. Sifakis, and S. Tripakis, “Modeling Urgency in Timed Systems,” *Compositionality: the significant difference*, COMPOS'97, LNCS 1536, Springer 1997.

[5] W.-T. Chang, A. Kalavade and E. A. Lee, “Effective Heterogeneous Design and Cosimulation,” chapter in *Hardware/Software Co-design*, G. DeMicheli and M. Sami, eds., NATO ASI Series Vol. 310, Kluwer Academic Publishers,1996.

[6] C. Cloet, M. Krucinski, R. Horowitz, and M. Tomizuka, “A hybrid control scheme for a copier paperpath,” *Proceedings of the 1999 American Control Conference*, San Diego, CA, June 1999, p.2114-2118.

[7] J. Davis etc., “Ptolemy II: Heterogeneous Concurrent Modeling and Design in Java,” UCB/ERL M99/40, University of California, Berkeley, CA 94720.

[8] T. A. Henzinger, “The theory of hybrid automata,” *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science* (LICS 1996), p.278-292.

[9] K.H. Johansson, M. Egerstedt, J. Lygeros, and S. Sastry, “On the Regularization of Zeno Hybrid Automata,” submitted to *Systems Control Letters, Special Issue on Hybrid Systems.*

[10] P. Kopke, T. Henzinger, A. Puri and P. Varaiya, “What's Decidable About Hybrid Automata?”, *27th Annual ACM Symposium on Theory of Computing* (STOCS), 1995, p.372-382.

[11] E.A. Lee, “Modeling Concurrent Real-time Processes Using Discrete Events,” Invited paper to *Annals of Software Engineering*, Special Volume on Real-Time Software Engineering, to appear.

[12] E. A. Lee and A. Sangiovanni-Vencentelli, “A Framework for Comparing Models of Computation,” *IEEE Transactions on CAD*, Vol. 17, No. 12, Dec. 1998.

[13] J. Liu, *Continuous-Time and Mixed-Signal Simulation in Ptolemy II*, UCB/ERL M98/74, University of California, Berkeley, CA 94720.

[14] J. Liu, X. Liu, T.J. Koo, B. Sinopoli, S. Sastry, and E.A. Lee, “A hierarchical hybrid system model and its simulation,” to appear in *IEEE Conference on Decision and Control* (CDC’99).

[15] N.A. Lynch, R. Segala, F.W. Vaandrager, and H.B. Weinberg, “Hybrid I/O automata,” *Hybrid System III*, LNCS 1066, Springer-Verlag, 1996, p.496-510.

[16] P. J. Mosterman, “An Overview of Hybrid Simulation Phenomena and Their Support by Simulation Packages,” *Hybrid Systems: Computation and Control* (HSCC’99), LNCS 1569, Springer, 1999, p.165-177.

[17] R. Orfali and D. Harkey, *Client/Server Programming with JAVA and CORBA*, Wiley Computer Publishing, 1997.

[18] R. A. Saleh and A. R. Newton, *Mixed-Mode Simulation*, Kluwer Academic Publishers, 1990.

[19] D.A. Schmidt, *Denotational Semantics: A Methodology for Language Development*, Allyn and Bacon, Newton, MA, 1986.

[20] Claire Tomlin, *Hybrid Control of Air Traffic Management Systems*, Ph.D. Thesis, Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, 1998.

[21] P. Varaiya, “Smart cars on smart roads: problems of control,” *IEEE Trans. on Automatic Control*, vol.38, (no.2), Feb. 1993. p.195-207.

[22] T*he Common Object Request Broker: Architecture and Specification* (formal/99-10-07 (CORBA 2.3.1)), Object Management Group, Inc., 1999.

[23] *Simulink 3 User's Guide*, The Mathworks Inc., 1999.

[24] *Verilog-AMS Language Reference Manual*, Version 1.2, Open Verilog International, June 1998.

[25] *VHDL Language Reference Manual* (*Integrated with VHDL-AMS Changes*), IEEE Standard 1076.1-1999.

[26] The Objectspace, Inc. web page, "http://www. objectspace.com"

[27] The Ptolemy II web page, "http://ptolemy. eecs.berkeley.edu/ptII"