Proceedings of the 2000 IEEE International
Symposium on Computer-Aided Control System Design
Anchorage, Alaska, USA ,, September 25-27, 2000, pp 95-100

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

Jie Liu and Edward A. Lee

*Department of Electrical Engineering and Computer Science*
*University of California, Berkeley*
{liuj, eal}@eecs.berkeley.edu

## Abstract[*]

This paper presents a component-based modeling technique for systems with continuous and discrete dynamics. It uses hierarchical composition to hide the implementation details of one component from other components, and keeps the components at the same level of hierarchy interacting under a well-defined model of computation. Continuous time, discrete event, and finite state machine models are considered. The signal conversions at the boundaries and the execution control among the components are studied. The modeling technique is implemented in Ptolemy II, a heterogeneous modeling and design environment. A hierarchical helicopter control system is modeled as an example.

## 1. Introduction

Large-scale control systems are intrinsically heterogeneous. The physical plants to be controlled are usually continuous but controllers are implemented using digital circuits or real-time software, which are discrete. While the closed-loop regulators interact with the plant at a fast sampling rate, a high-level planner may interact with the regulators at much more sparse time points. There are a variety of models that match different parts of a system, for example, ordinary-differential equations for the plant dynamics, a sampled-data model for the regulator, a finite-state-machine model for operation modes, and a discrete-event model for high-level planners. Although each individual model is relatively well-understood, the integration of heterogeneous models brings additional difficulties and complexities to the design of such systems.

Component-based modeling techniques view the building blocks of a system to be “black boxes,” as shown in Figure 1.The interface of a component and the communication scheme among components are specified, while the contents of a component can be implemented using a different model. A 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 component reuse.

This paper 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 world. Simulink[†], originally a continuous-time control system design tool, has been enhanced to deal with sampled-data systems and, to some extent, discrete-event systems [18]. Hardware description languages, like VHDL and Verilog, extend and standardize the capability of modeling continuous-time systems (e.g. analog circuits) in discrete-event based languages. Both of the approaches attempt to come up with a unified model to capture semantically different components, but lack arbitrary nesting of heterogeneous models.

One of the most remarkable efforts in mixing continuous and discrete dynamics is the study of hybrid systems [2]. Solid theoretical frameworks for modeling and analyzing hybrid systems are under rapid development [1, 4, 7, 10, 14], with applications in air traffic control, transportation systems, automotive, manufacturing systems, and electro-mechanical systems etc. [3, 16, 17]. Nevertheless, a simulation tool with clean semantics and composability that leverages the theory of hybrid system is still under high demand [15].

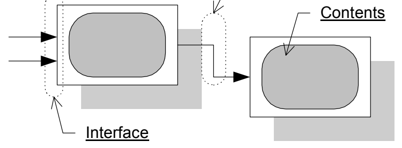

Figure 1. Heterogeneous components

*. This work is part of the Ptolemy project, which is supported by DARPA/ITO, the State of California MICRO program, and the following companies: Cadence Design Systems, Hewlett Packard, Hitachi, Hughes Space and Communications, Motorola, NEC, and Philips.

†. A software package from the MathWorks Inc.

In this paper, a component-based framework is proposed to model systems with both continuous and discrete dynamics. It uses hierarchical compositions to hide the implementation details of one component from other components, and keeps 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. The framework is implemented in Ptolemy II, a heterogeneous modeling and design environment.

In the following sections, we discuss the component-based models (section 2), the signal conversion techniques and the requirements of numerical ODE solvers for handling them (section 3), and the hierarchical composition and execution among heterogeneous models (section 4). Section 5 gives an example of a three-layered helicopter control system modeled in Ptolemy II.

## 2. Component-based model for continuous and discrete dynamics.

This section describes how individual models of interest can be built in a component-based framework, emphasizing what the components are, how they interact, and how the execution is done.

### 2.1. Continuous-Time Model

Consider an initial value problem of ODEs for a continuous-time (CT) system:

$$\begin{aligned} \dot{x} &= f(x, u, t), x(t_0) = x_0 \\ y &= g(x, u, t), \end{aligned} \tag{1}$$

where,

- $t \in \Re$, $t \geq t_0$, a real number, is time;
- $x(t) \in \Re^n$ is the $n$-dimensional state of the system at $t$;
- $u(t) \in \Re^m$ is the $m$-dimensional input;
- $y(t) \in \Re^l$ is the $l$-dimensional output;
- $\dot{x} = dx/dt \in \Re^n$ is the derivative of $x$ w.r.t. time $t$;
- $x_0 \in \Re^n$ is the initial condition.

Using components, an ODE system (1) can be modeled using a block diagram 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. A special component, the integrator, makes a feedback loop an ODE. The functions *f* and *g* can consist of a feedforward composition of CT components.

The simulation of this model involves solving the ODE numerically with respect to the inputs at a discrete set of time points, and to produce outputs at those points. The

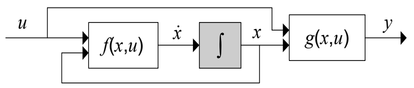

Figure 2. A component-based model for ODE

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 [12].

### 2.2. Discrete-Event Model

A discrete-event model is a timed model, where time is global to all components. An event has a value and a time stamp. Components in the model communicate via a set of events located discretely on the time line. A component, when executing, consumes input events and produces output events. The output events are required to be no earlier in time than the input events that trigger them (this property is called *causality*).

The simulation of this model uses 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.

A component can schedule itself to be executed at a particular future time by placing a *pure event* (an event without a value) on the event queue, with the component itself as the destination. When this pure event is dequeued, the component can be executed. Pure events are useful for source components (components without input ports) to produce events at a reasonable rate, and they are also essential for interacting with other models.

### 2.3. Finite-State-Machine Models

In a 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 does not necessarily have the 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 associated action will be performed. The end state of the transition will be the new current state.

In this paper, we assume the FSM model to be non-blocking, meaning that there will always be a transition enabled for all input events and all states. This is not a restrictive assumption, since we can add an implicit self-loop transition on all states such that if no other trigger condition is true for a given input, the self-loop transition will be taken.

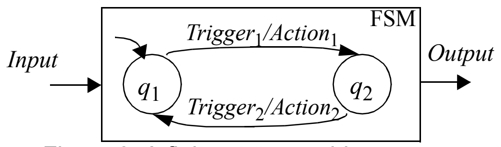

Figure 3. A finite-state-machine component.

## 3. Signal Conversions and Requirements of CT Simulation.

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

### 3.1. Event Generation

Event generation is to generate discrete events from piecewise continuous waveforms. A key for event generation is to find event time stamps. We classify two kinds of events:

- *time events*. These are events whose time stamps are known beforehand. A typical case is the sampling events in a sampled-data system.
- *state events*. The time of this type of event depends on the value of the state variables of the CT system. An example is threshold crossing detection.

In general, the time stamp of state events cannot be predicted accurately in advance. Special treatment has to be done in the process of ODE solving.

### 3.2. Waveform Generation

Waveform generation is the conversion of discrete events into piecewise continuous waveforms. A key is to provide values in the waveform between successive event time stamps. This conversion is usually application dependent. One popular way is the zero-order hold, which is consistent with common D/A converters. In general, any extrapolations of previous event values is reasonable.

### 3.3. Requirements of CT Simulation

Traditional ODE solvers assume sufficient smoothness of the right-hand side functions. This may not be valid when the inputs of a CT component are generated from discrete events. Furthermore, the event generation process requires the solvers to be able to find the state of the CT system at any time when an event occurs. For this purpose, 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 smooth, or the output map is not continuous. The numerical ODE solvers cannot cross breakpoints in one integration step since either the smooth-RHS assumption is violated or an event needs to be produced.

According to whether a breakpoint can be predicted before an integration step is taken, we classify two kinds of breakpoints – *predictable* ones and *unpredictable* ones. For example, time events and some unsmoothness in input signals are predictable, while state events and unsmoothness depending on state variables are unpredictable. Predictable breakpoints can be stored in a table and handled efficiently.

The introduction of breakpoints adds more requirements for ODE solvers to adjust numerical integration step sizes. There are three factors to be considered:

1.) *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.
2.) *Convergence*. Implicit numerical methods use fixed-point iteration or Newton iteration to solve the induced algebraic equations. Choosing a smaller step size may help improve the initial guess and help convergence.
3.) *Breakpoints*. Before each integration step, the breakpoint table is queried, and the intended 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.

## 4. Hierarchical Composition and Execution Control

The execution control of heterogeneous components is critical for a correct and efficient simulation engine. One issue is how time is managed. There can be multiple time variables in different components, and time advances in various ways. For example, time in a DE model jumps, while in a CT model it evolves continuously. In our hierarchical composition, we call the time at the highest level of hierarchy the "global time," and the time maintained within a component the "local time." The management of time, as well as the execution flow, depends on the composition of models.

### 4.1. Mixed-signal Systems

#### 4.1.1. DE inside CT

Figure 4 shows a DE component wrapped by an event generator and a waveform generator. Since time advances monotonically in CT and events are generated chronologically, the DE component receives input events monotonically in time. In addition, a composition of causal DE components is causal [11], so the time stamps of the output events from a DE component are always greater than or equal to the global time. From the view point of the CT system, the events (i.e. breakpoints) produced by a DE component are predictable.

Note that in the CT model, finding the numerical solution of the ODE at a particular time is semantically an instantaneous behavior. During this process, the behavior of all

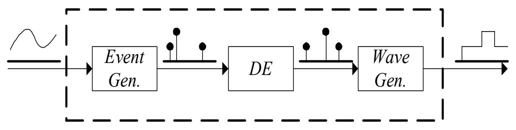

Figure 4. A DE component in a CT model.

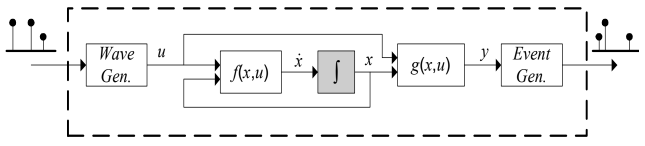

Figure 5. A CT component in a DE model.

components, including those implemented in a DE model, should keep unchanged. This implies that the DE components should not be executed during one integration step of CT, but only between two successive CT integration steps.

#### 4.1.2. 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. Let the CT component have local time $t$, when it receives an input event with time stamp $\tau$. Since time is continuous in the CT model, it will execute from its local time, and may generate events at any time greater or equal to $t$. Thus we need

$$t \geq \tau \tag{2}$$

to ensure the causality. This means that the local time of the CT component should be greater than or equal to the global time whenever it is executed.

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 time is smaller than its current 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 send a pure event to the DE system at the event time, and wait until it is safe to emit it.

### 4.2. Hybrid Systems

A hierarchical composition of FSM and CT is shown in Figure 6. Although FSM is an untimed model, 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 output. The FSM can use predicates on them, as well as the input signals, to build trigger conditions. The actions associated with transitions are usually setting parameters in the destination state, including the initial conditions of integrators.

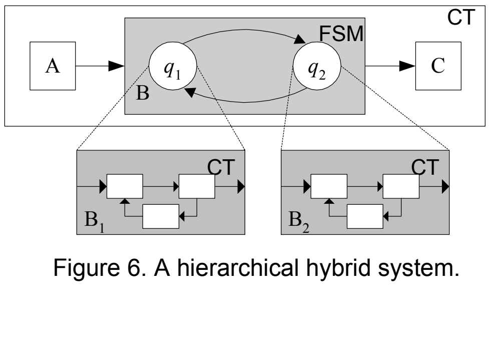

Figure 6. A hierarchical hybrid system.

During continuous evolution, the system is simulated as a CT system where the FSM is replaced by the continuous component refining the current FSM 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 actions on the transition. The simulation continues with the transition time treated as a breakpoint.

## 5. Modeling a Helicopter Control System

As an example, we model a helicopter control system with three layers of hierarchy, the helicopter dynamics layer, the regulation layer, and the trajectory planning layer, as shown in Figure 7. The goal is to demonstrate the mixing and match of modeling techniques, rather than optimal control algorithms. The architecture of the system is a simplification of the one presented in [9].

### 5.1. Helicopter Dynamics

We consider the 2-D motion of the helicopter along the longitudinal ($x$) and vertical ($z$) axes, which are pointing north and down, respectively. The motion of the helicopter is controlled by $T_M$, the main rotor thrust, and $\beta$, the longitudinal tilt path angle. Variables $p_x$, $p_z$, and $\theta$ are the position on the $x$-axis, $z$-axis, and the pitch angle. The equations of the motion can be expressed as [13]:

$$\begin{bmatrix} \ddot{p}_x \\ \ddot{p}_z \end{bmatrix} = -\frac{1}{m}\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}\begin{bmatrix} T_M \sin\beta \\ T_M \cos\beta \end{bmatrix} + \begin{bmatrix} 0 \\ g \end{bmatrix} \tag{3}$$

$$\ddot{\theta} = \frac{1}{I_y}(M_M\beta + h_M T_M \sin\beta) \tag{4}$$

where $m$ is the mass of the helicopter, $I_y$ is the moment of inertia about body $y$-axis, $M_M$ is the hub pitching moment stiffness, and $h_M$ is the vertical distance between the main rotor and the center of gravity. The state vector is $\boldsymbol{x} = [p_x, \dot{p}_x, p_z, \dot{p}_z, \theta, \dot{\theta}]^T$ and the input vector is $\boldsymbol{u} = [T_M, \beta]^T$.

### 5.2. Regulation Layer

The regulation is based on *flight modes* [8], which represent different modes of operation of the helicopter. The closed-loop regulation laws are designed for each individual mode to optimize the performance. A complex motion is

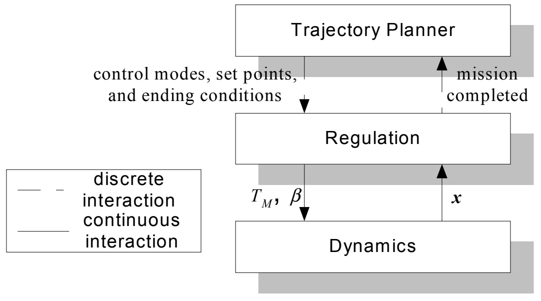

Figure 7. A hierarchical hybrid control architecture.

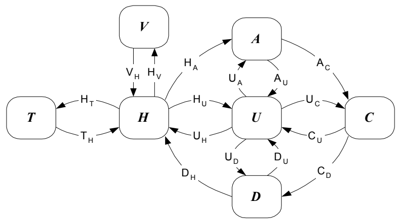

Figure 8. Possible flight mode switching.

achieved by switching among the flight modes. The flight modes – hover (***H***), turn (***T***), vertical up/down (***V***), cruise (***U***), accelerate (***A***) or decelerate (***D***) with altitude hold, and climb/descend (***C***). The possible switching among them are shown as an FSM in Figure 8.

In each mode, the regulation uses approximate feedback linearization. Assuming all states are accessible for control purpose, the linearizer has the following form:

$$\begin{bmatrix} T_M^{(3)} \\ \dot{\beta} \end{bmatrix} = \mathbf{A}^{-1}(x)\left[-\boldsymbol{b}(x) + \boldsymbol{r}\right] \quad . \tag{5}$$

where $\boldsymbol{r} = [r_x, r_z]^T$ is the input of the feedback linearizer. The resulting closed-loop system has a linear dynamics:

$$\begin{bmatrix} p_x^{(5)} \\ p_z^{(5)} \end{bmatrix} = \begin{bmatrix} r_x \\ r_z \end{bmatrix} . \tag{6}$$

The linearization rule is shared by all modes, but ***r*** can be different variables under different modes, depending on the goal of the regulation. A uniform pole placement control law is used to generate ***r***. The controllers take the form:

$$\begin{aligned} r_i^p &= -\alpha_0(p_i - p_i^s) - \alpha_1\dot{p}_i - \ldots - \alpha_4 p_i^{(4)} \\ r_i^v &= -\alpha_1(\dot{p}_i - \dot{p}_i^s) - \alpha_2\ddot{p}_i - \ldots - \alpha_4 p_i^{(4)} \\ r_i^a &= -\alpha_2(\ddot{p}_i - \ddot{p}_i^s) - \alpha_3 p_i^{(3)} - \alpha_4 p_i^{(4)} \end{aligned} \tag{7}$$

for $i = x, z$, set points $p_i^s, \dot{p}_i^s, \ddot{p}_i^s$, and constant $\alpha$'s. For each mode, the mapping of ***r*** and the corresponding control laws are summarized in the following table.

| Mode | Outputs | Control laws |
|---|---|---|
| ***H/T*** | $\boldsymbol{r} = [p_x, p_z]^T$ | $r_x = r_x^p; r_z = r_z^p$ |
| ***U*** | $\boldsymbol{r} = [\dot{p}_x, p_z]^T$ | $r_x = r_x^v;\ r_z = r_z^p$ |
| ***A/D*** | $\boldsymbol{r} = [\ddot{p}_x, p_z]^T$ | $r_x = r_x^a;\ r_z = r_z^p$ |
| ***C*** | $\boldsymbol{r} = [\dot{p}_x, \dot{p}_z]^T$ | $r_x = r_x^v;\ r_z = r_z^v$ |
| ***V*** | $\boldsymbol{r} = [p_x, \dot{p}_z]^T$ | $r_x = r_x^p;\ r_z = r_z^v$ |

## 5.3. Trajectory Planning Layer

The trajectory planner, having the knowledge about possible mode switching, accepts pilot's commands like "fly from point *A* to point *B*," finds optimal paths, decomposes them into sequences of flight modes, and controls the switching of regulation layer controllers. For each flight

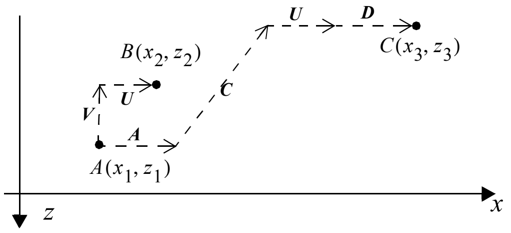

Figure 9. Examples of trajectory planning.

mode, the planner also gives the set points and ending conditions to the regulator. When the ending condition is achieved, a "mission completed" message is sent back to the planner, and the planner will issue the next flight mode.

For example, as shown in Figure 9, if the helicopter, hovering at point *A* and heading north, wants to fly to a close point *B*, the trajectory planner can send the following operations to the regulation layer sequentially:

| No. | Mode | Set Points | Ending Condition |
|---|---|---|---|
| **0** | ***H*** | $p_x^s = x_1;\ p_z^s = z_1$ | – |
| **1** | ***V*** | $p_x^s = x_1;\ \dot{p}_z^s = v_z$ | $p_z = z_2$ |
| **2** | ***H*** | $p_x^s = x_1;\ p_z^s = z_2$ | $\dot{p}_z = 0$ |
| **3** | ***U*** | $\dot{p}_x^s = v_x;\ p_z^s = z_2$ | $p_x = x_2$ |
| **4** | ***H*** | $p_x^s = x_2;\ p_z^s = z_2$ | – |

In another case, a high-altitude take-off flight from *A* to *C* may go through modes: ***H***, ***A***, ***C***, ***U***, ***D***, and ***H***, with corresponding set points and ending conditions.

## 5.4. Modeling the Helicopter System in Ptolemy II

The component-based model of the helicopter system in Ptolemy II has several levels of hierarchy. The top level is a DE model for the trajectory planner, which interacts with a discrete-event abstraction of the regulators, as shown in Figure 10. The DE model at this level captures the discrete interaction between the planner and the regulation layer, which is irregular in time, and it is possible to model computation and communication latencies so that the performance of the system is easy to analyze.

The regulation layer is modeled in the CT domain, as shown in Figure 11. The controller is internally implemented as an FSM, as shown in Figure 8. The mode transitions, set points, and ending condition parameters are controlled by the inputs. Each mode is further refined by a CT subsystem, implementing the control law and detecting ending conditions. For example, the cruise mode controller

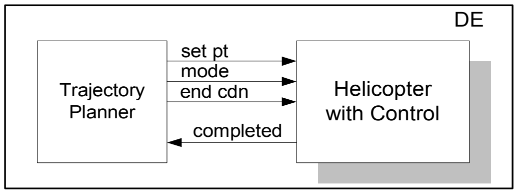

Figure 10. The top-level model.

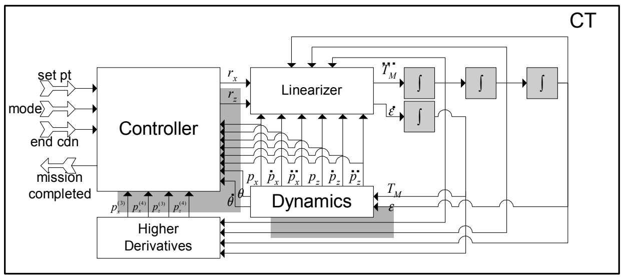

Figure 11. Model of the regulation layer.

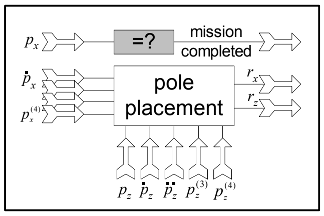

Figure 12. Cruise mode controller.

is shown in Figure 12. Ending condition detectors are event generators, such that when the condition is satisfied, a "mission completed" event is sent to the trajectory planner. Thus, the "Controller" is internally a hybrid system, which exposes a discrete interface to the trajectory planner and a continuous interface to the linearizer and the helicopter dynamics.

In Figure 11, the "Dynamics" block is further refined by a CT subsystem, as the one in Figure 13, which implements the equations (3) and (4).

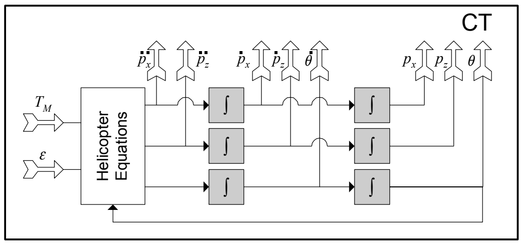
CT
$\ddot{p}_x$ $\ddot{p}_z$ $\dot{p}_x$ $\dot{p}_z$ $\dot{\theta}$ $p_x$ $p_z$ $\theta$
$T_M$
$\varepsilon$
Helicopter Equations

Figure 13. Model of the helicopter dynamics.

In this example, we clearly separate the implementation details of the helicopter model from the control parts, and separate the modal regulators from the high level planner. At the same time, the components are systematically and modularly integrated by formal models of computations. This separation and integration provide the composability of components, the interoperability of the models, and the possibility of design reuse.

## 6. Conclusion

This paper presents a component-based framework for modeling systems with continuous and discrete dynamics. The framework systematically and modularly integrates different models by hierarchically composing heterogeneous components. Information hiding, signal conversions, and execution control under this framework are studied. A hierarchical helicopter control system is modeled as an example in Ptolemy II.

## 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-Verlag, 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] JDavis et al., "Ptolemy II: Heterogeneous Concurrent Modeling and Design in Java," UCB/ERL M99/40, University of California, Berkeley, CA 94720.
[7] 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.
[8] T.J. Koo, F. Hoffmann, H. Shim, B. Sinopoli, and S. Sastry, "Hybrid Control of Model Helicopter," *Proc. of IFAC Workshop on Motion Control*, Grenoble, France, Oct. 1999, pp 285-290.
[9] T.J. Koo, B. Sinopoli, A. Sangiovanni-Vincentelli, S. Sastry, "A Formal Approach to Reactive System Design: A UAV Flight Management System Design Example," in *Proc. of IEEE International Symposium on Computer-Aided Control System Design*, Kohala Coast, Hawaii, Aug. 1999.
[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," *Annals of Software Engineering*, Special Volume on Real-Time Software Engineering, vol. 7 (1999), p.25-45.
[12] J. Liu, *Continuous-Time and Mixed-Signal Simulation in Ptolemy II*, UCB/ERL M98/74, University of California, Berkeley, CA 94720.
[13] J. Liu, X. Liu, T.J. Koo, B. Sinopoli, S. Sastry, and E.A. Lee, "A hierarchical hybrid system model and its simulation," *Proc. of the 38th IEEE Conference on Decision and Control* (CDC'99), Phoenix, AZ, Dec. 1999, p.3508-3513.
[14] 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.
[15] P. J. Mosterman, "An Overview of Hybrid Simulation Phenomena and Their Support by Simulation Packages," *Hybrid Systems: Computation and Control* (HSCC'99), LNCS1569, Springer, 1999, p.165 - 177.
[16] 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.
[17] P. Varaiya, "Smart cars on smart roads: problems of control," *IEEE Trans. on Automatic Control*, vol.38, (no.2), Feb. 1993. p.195-207.
[18] *Simulink 3 User's Guide* (1999), The MathWorks Inc.