Software-Enabled Control: Information Technology for Dynamical Systems
Tariq Samad and Gary Balas (eds.), Wiley-IEEE Press, April 2003

Heterogeneous Modeling and Design of Control Systems

Xiaojun Liu, Jie Liu, Johan Eker, Edward A. Lee
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley
Berkeley, CA 94720, USA
{liuxj, liuj, johane, eal}@eecs.berkeley.edu

Editors' Summary

Complex control systems integrate a variety of functions and capabilities, which will in general rely on different computational mechanisms. The plant model may be represented as a set of ordinary differential equations, the mode switching logic may be expressed as a finite state machine, and dataflow models may be used to capture the architecture of a sensor processing subsystem, for example. Design tools are needed that can support these heterogeneous models of computation—and their integration within a single control system.

This chapter describes Ptolemy II, a component-based design environment that allows different models of computation to be hierarchically composed. Individual components are called actors; these can include simple operators such as an AND gate and, through compositionality in the form of arbitrarily nested actor hierarchies, complex functions such as a Kalman filter. Different models of computation can then be realized by imposing a (possibly partial) execution order and a communication mechanism on the actors that comprise a composite actor.

Ptolemy II can be particularly useful for the design of hybrid dynamical systems—systems that combine discrete event mode logic and continuous time dynamics within each mode. By using

Ptolemy II to model a hybrid system, we can choose whichever model of computation is most appropriate for a particular task while ensuring the consistency and integrity of the overall model.

An application of Ptolemy II to the control of the Furuta inverted pendulum is also described in this chapter, with an emphasis on implementation issues relevant to real-time operating systems. As noted in Chapter 4, Ptolemy II has been interfaced with the OCP and used to model a nonlinear actuator as part of an overall vehicle simulation. Chapter 5 also notes an OCP-related application of the Ptolemy project, in this case to support the validation of reconfiguration strategies and other hybrid control aspects.

1 Introduction

Computer control is now the standard technique for implementing control systems, mainly for two reasons. First is the exponential reduction in the cost of computing; second is the versatility of implementing control laws in software. Many developments in control systems are only practical with computer control, e.g. to implement the nonlinear and time varying control laws associated with adaptive control. Complicated computations can be incorporated into the control loop, for example when computer vision is used to guide a robot.

Designing the software for such control systems is hard because the systems are usually *heterogeneous*. They may include subsystems with very different characteristics, such as hydraulic actuators and an inertial navigation system. On the software side the situation is similar. The controller may have several operational modes. The control law in each mode can be specified by difference equations; the mode switching logic can be specified by a state machine. For vision guidance, complex image processing algorithms need to be programmed.

For each of these subsystems and aspects of the software, formal models that support its modeling, analysis, or programming have been developed. For example, image processing algorithms can be programmed in various dataflow models [16], [17]. Each formal model employs a computational mechanism that dictates what are the components in the model, and how they communicate and execute. Such a mechanism is called a *model of computation*.

Working with heterogeneous systems requires more than one model of computation. This is evident from the trend of adding extensions to existing tools and formal models. For example, both VHDL and Verilog, originally designed for digital circuits and based on the discrete-event model, have been extended to handle analog components [12], [25]. Simulink, a continuous-time environment, has been extended with Stateflow [22] for modeling and designing event-driven systems. Ideal switching elements, controlled by finite state machines, are added to bond graphs for modeling hybrid systems [27]. However, most of these tools and formal models support just a few models of computation and few choices in the way they can be combined. Further extensions may be awkward or impossible due to the semantic mismatch between the new model of computation and the existing infrastructure.

Ptolemy II [7] is a system-level design environment that supports component-based heterogeneous modeling and design. Its model structure allows a variety of models of computation to be implemented, and to be hierarchically composed in heterogeneous models. This paper presents Ptolemy II and illustrates its application to control system design. We use several case studies in section 2 to elaborate the challenges in designing complex control software. The Ptolemy II model structure is discussed next. Section 4 gives an overview of the models of computation that are useful in control system design. The Ptolemy II modal model structure is presented in section 5. An inverted pendulum controller is used to demonstrate how Ptolemy II can be used in the modeling and design exploration of control systems. In the last section we present conclusions and discuss our ongoing work.

2 Software Complexity in Control Systems

The use of computers in control systems started in the 1950's. In the 1980's computer control became the standard technique for implementing new control systems, from simple single-loop controllers to large distributed control systems [2]. The versatility of implementing control laws in software brings many opportunities to control system design. For example, a proportional-integral-derivative (PID) controller can come equipped with automatic tuning and gain scheduling. As another example, to achieve better performance, a controller can be designed to switch among a set of candidate control laws according to the operating region of the controlled process. Such developments bring about increased complexity in control software. As the following cases from the theory and applications of control systems will demonstrate, the capability to build complex and reliable software has become a key enabler of further developments in computer-controlled systems.

- Vision-guided landing of unmanned aerial vehicles (UAVs) [26]

  A UAV equipped with a video camera is to land on a moving landing platform (e.g. the landing pad on a ship). The UAV control system uses computer vision as a sensor in the feedback control loop. The images captured by the camera are processed and relevant features in the field of vision are extracted. The extracted features are further processed by a computer vision algorithm to estimate the motion of the UAV relative to the landing platform. The control software has to perform complex image transformations and analyses in real-time.

- Model-based fault diagnosis [24]

The goal of fault diagnosis is to detect and isolate faults in physical processes. In the model-based approach to fault diagnosis, a process model is used to predict normal process behavior. Faults are detected when observed process behavior deviates from normal behavior. Based on the deviation, one or more hypothesized faults can be generated for fault isolation. The hypothesized faults are injected into the process model to predict future behavior. The result of fault isolation consists of those faults whose predictions are consistent with the observations. Elaborate process models are needed to achieve greater resolution and coverage in fault diagnosis. When we build a control system that uses model-based fault diagnosis, the software for process modeling and simulation is not only an essential design-time tool, but also a crucial component in the deployed system. A desirable feature in such software is the support of dynamic model modification for fault injection.

- Multi-modal control [15]

  Many controlled systems have multiple modes of operation. Consider the flight of a helicopter. Each possible maneuver - hover, turn, vertical climb, etc. - corresponds to a mode of operation. To optimize performance, each mode has its own closed-loop feedback controller. The helicopter flight management system can be structured in layers, for example a trajectory planner layer and a regulation layer. Given a flight task, for example to fly to a certain location, drop the load, and fly back, the trajectory planner comes up with a sequence of flight modes (maneuvers), the set points for the controller of each mode, and ending conditions. For example, one maneuver in the sequence may be to accelerate horizontally, with the ending condition that the horizontal velocity reaches 150 km/h. The regulation layer switches controllers according to the flight sequence and ending conditions. The software for the regulation layer can be very well structured using the hybrid system formalism [20]. The operation modes and switching among modes are captured by a finite state machine (FSM). Each mode is represented by a state that contains the controller of that mode.

Using FSMs in a hierarchical model was first made popular by Harel. He proposed *Statecharts* [11], which combine hierarchical FSMs and concurrency. The proposal stimulated many developments in both theory and applications. A recent development, **charts* (pronounced star-charts) [10], generalizes and unifies Statecharts and hybrid systems.

- Embedded control systems

Similar to what happened in control engineering, computer technology has been extensively applied to many application domains and opens up many exciting opportunities. For example, the concept of "real-time" enterprises has been recently proposed [13]. In such an enterprise, all the information that is relevant to business decision-making, from inventory to cash flow, is made available at the click of a mouse, not just on a weekly or monthly basis. The enterprise can adapt better to the rapidly changing marketplace. This is enabled by the use of computer and internet technologies in every aspect of enterprise management. Many computer-control systems will be integrated into a larger context. In this vision, the process control system of a petrochemical plant will become a component of the production management system that also manages the inventory of raw materials and end products and schedules production according to supply and demand. Such integration requires a sound strategy to interface design and abstraction. It is already a hard problem to integrate software systems in the same application domain but from different vendors. Integration across application domains can only be more challenging.

From this brief survey, how to manage what we call heterogeneity emerges as the key question to be answered when designing complex control software. To further illustrate the notion of heterogeneity, let us consider a flight management system of an unmanned helicopter. The system employs multi-modal control, fault diagnosis based on a model of the helicopter dynamics, and vision-guided landing. The controllers of some flight modes may be described by difference equations. In the landing mode, the controller needs to perform image transformations that are best programmed using dataflow languages and models. The dynamics model is simulated, possibly by numerically solving ordinary differential equations, to predict the state of the helicopter. An FSM captures the mode switching logic. Multi-modal control and fault diagnosis are assigned to different tasks in a real-time operating system (RTOS). Such a control system is heterogeneous in that its subsystems have very different characteristics. Their components may interact by synchronous rendezvous or asynchronous event notification, may execute sequentially or in parallel, may communicate via continuous-time signals or streams of data. A disciplined approach is indispensable when composing heterogeneous systems from diverse subsystems.

3 The Ptolemy II Model Structure

The Ptolemy II modeling and design environment [7] uses a component-based design methodology that is consistent with component-based techniques used in object-oriented design [29]. The components in a Ptolemy II model are called *actors*. A model is a hierarchical composition of actors, as shown in figure 7.1. The *atomic* actors, such as A1, only appear at the bottom of the hierarchy. Actors that contain other actors, such as A2, are *composite*. A composite actor can be contained by another composite actor, so the hierarchy can be arbitrarily nested.

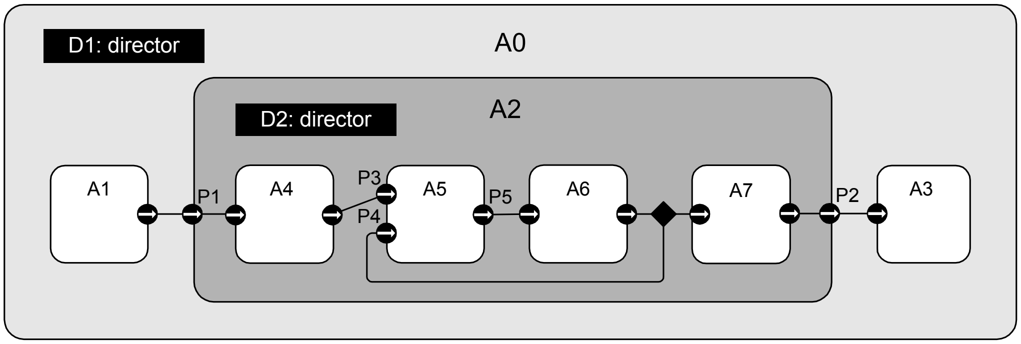


FIGURE 7.1 The schematic of a hierarchical Ptolemy II model.

Atomic actors encapsulate basic computation, from as simple as an AND gate to more complex such as a fast Fourier transform (FFT). Through composition, actors that perform even more complex functions can be built. Actors have *ports*, which are their communication interfaces. For example, in figure 7.1, A5 receives data from input ports P3 and P4, performs its computation, and sends the result to output port P5. A port can be both an input and an output. Communication channels among actors are established by connecting ports. A port of a composite actor, such as P1, can have connections both to the inside and to the outside, thus linking inside actors to outside actors.

There is one last part in figure 7.1 that we have yet to explain: the *directors*. As the names actor and director suggest, a director controls the execution order of the actors in a composite, and mediates their communication. In figure 7.1, D1 may choose to execute A1, A2, and A3 sequentially. Whenever A2 is executed, D2 takes over and executes A4~A7 accordingly. A director uses receivers to mediate actor communication. As shown in figure 7.2, one receiver is created for each communication channel; it is situated at the input ports, although this makes little difference. When the producer actor sends a piece of data, called a *token* in Ptolemy II, to the output port, the receiver decides how the transaction is completed. It may put the token into a first-in first-out (FIFO) buffer, from which the consumer actor will get data. It may tag the token as an event, and put the event in a global event queue. The token will be made available to the consumer when time comes for the consumer to process the event. Or it may stall the producer to wait for the consumer to be ready.

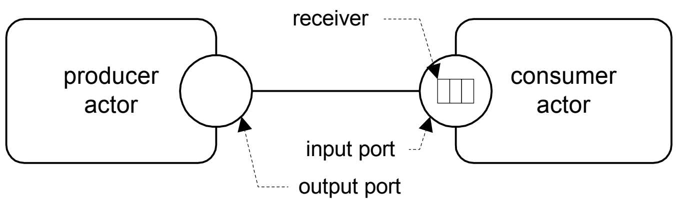
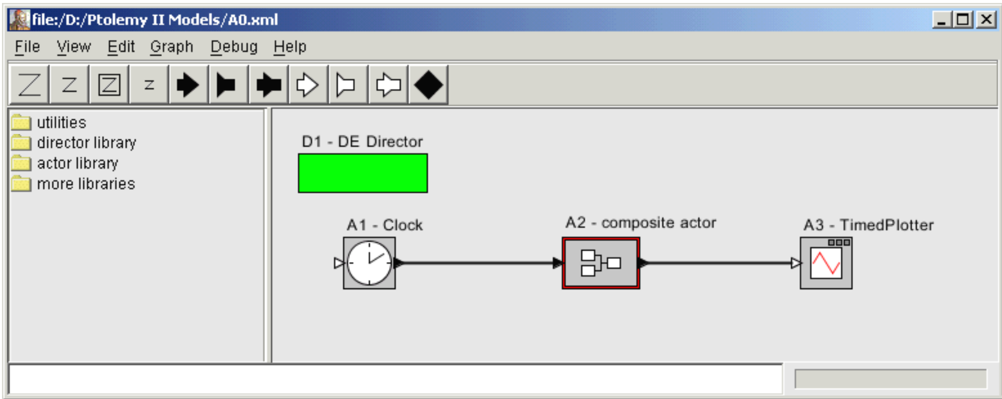


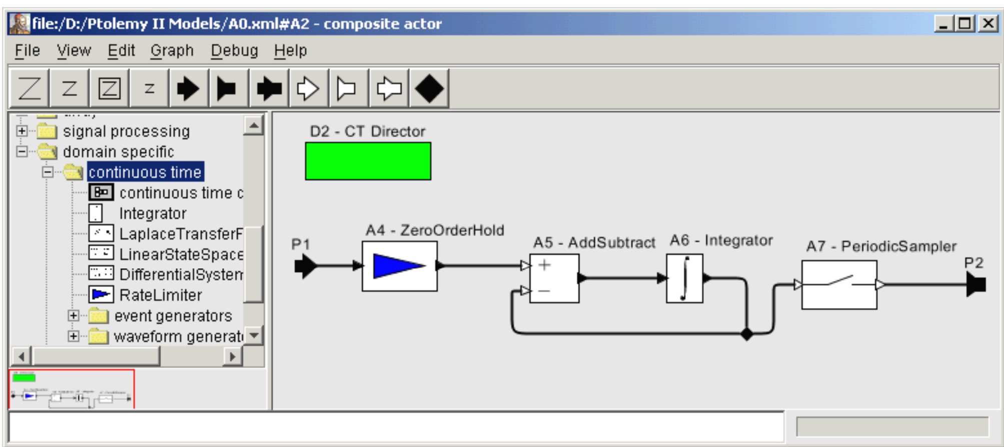
FIGURE 7.2 A receiver is used to mediate communication between actors.

By choosing an ordering strategy and a communication mechanism, a director implements a model of computation. Within a composite actor, the actors under the immediate control of a director interact homogeneously. Properties of the director's model of computation can be used to reason about the interaction. A heterogeneous system is modeled by using multiple directors in different places in the hierarchy. A concrete example, complete with the graphical user interface to Ptolemy II, is shown in figure 7.3. The directors are carefully designed so that they provide a polymorphic execution interface to the director one level up in the hierarchy. This ensures that the model of computation at each level in the hierarchy is respected.

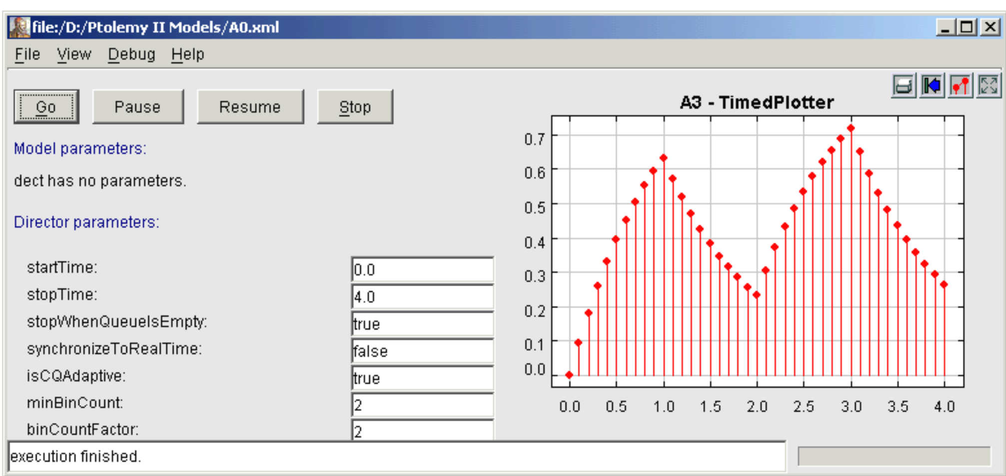
In Ptolemy II, the realization of a model of computation is called a *domain*, so directors are associated with domains. Most actors, however, are not. Such actors are agnostic about how their inputs are received and outputs sent. They can be reused in different domains, and are called *domain-polymorphic*.



a) Top level, a discrete event model.



b) A continuous time model embedded in the above discrete event model.



c) The window that controls the simulation of the model and displays the result.

FIGURE 7.3 A heterogeneous model realizing the schematic shown in figure 7.1.

4 Concurrent Models of Computation for Control Systems

A diverse set of models of computation have been implemented in Ptolemy II. Here we will discuss a subset of them, those that are most useful in the modeling and design of control systems.

4.1 Continuous time

The continuous time (CT) domain [19] models ordinary differential equations (ODEs), extended to allow the handling of discrete events. Special actors that represent *integrators* are connected in feedback loops in order to represent the ODEs. Each connection in this domain carries a continuous-time signal. The actors denote the relations among these signals. Event generators, e.g. periodic samplers, triggered samplers, and zero-crossing detectors, and waveform generators, such as a zero-order hold, are implemented to convert between continuous-time signals and discrete events. A CT model is shown in figure 7.3b.

The execution of a CT model involves the computation of a numerical solution to the ODEs at a discrete set of time points. In order to support the detection of discrete events and the interaction with discrete models of computation, the time progression and the execution order of a CT model are carefully controlled [19]. The CT domain can be used to model physical processes whose dynamics are described by ODEs, or continuous control laws.

4.2 Discrete event

In the discrete event (DE) domain [3] of Ptolemy II, actors share a global notion of time and communicate through events that are placed on a (continuous) time line. Each event has a value and a time stamp. Actors process events in chronological order. The output events produced by an actor are required to be no earlier in time than the input events that were consumed. In other words, DE models are *causal*.

Discrete event models, having the continuous notion of time and the discrete notion of events, are widely used in modeling hardware and software timing properties, communication networks, and queuing systems.

4.3 Dataflow models

In dataflow models [17], connections represent data streams, and actors are processes that compute their output data streams from input streams. In such models, the order of execution for the processes are only constrained by the data dependency among them. This makes dataflow models amenable to optimized execution, for example to minimize buffer sizes, or to achieve a higher degree of parallelism. Dataflow models are very useful in designing signal processing algorithms and sampled control laws.

There are many variants of dataflow models, of which synchronous dataflow (SDF) [16] is a particularly restricted special case. In SDF, when an actor executes, it consumes a fixed number of tokens from each input port, and produces a fixed number of tokens to each output port. For a consistent SDF model, a static schedule can be computed, such that the actors always have sufficient data before execution. For algorithms with a fixed structure, SDF is very efficient and predictable.

4.4 Timed multitasking

The timed multitasking (TM) domain in Ptolemy II allows designers to explore priority-based scheduling policies such as those found in an RTOS and their effects on real-time software. In this domain, actors are software tasks with priorities. The director of this domain implements a prioritized event dispatching mechanism and invokes tasks according to their feasibility and priority. Both preemptive and nonpreemptive scheduling, as well as static and dynamic priority assignment, can be modeled.

4.5 Synchronous/Reactive

In the synchronous/reactive (SR) model of computation [9], the connections represent signals whose values are aligned with global clock ticks. Thus, they are discrete signals, but need not have a value at every clock tick. The actors represent relations between input and output values at each tick, and are usually partial functions with certain technical restrictions to ensure determinacy. Examples of languages that use the SR model of computation include Esterel [6] and Signal [5]. SR models are excellent for discrete control applications with multiple, tightly-coupled, and concurrent tasks. Because of the tight synchronization, safety-critical real-time applications are a good match.

4.6 Finite state machines

An FSM has a set of states and transitions among states. An FSM reacts to input by taking a transition from its current state. (The transition may be an implicit transition back to the current state.) Output may be produced by actions associated with the transition. FSMs are very intuitive models of sequential control logic and the discrete evolution of physical processes. FSM models are amenable to in-depth formal analysis and verification. Applications of FSM models include datapath controllers in microprocessors, and communication protocols.

In Ptolemy II, the FSM domain provides two modeling mechanisms. One allows designers to create actors whose behaviors are specified by FSMs. We can think of this as a graphical scripting language for writing new actors. The other one applies to *modal models*, which are hierarchical composition of FSMs with other models of computation.

5 Modal Models

Many engineering systems exhibit modes of operation. We call such systems *modal systems*. Defining operational modes is a very useful instrument of abstraction, which helps us to gain a high level understanding of system operation. In control engineering, a modal system operates with continuous dynamics in each mode. When mode changes, the continuous dynamics change abruptly [23]. Such modal systems are more specifically called *hybrid systems*. One example is a multi-tank system, a common experimental platform in control engineering. A modal controller, discussed in section 2, is also an example.

A number of approaches to the modeling and simulation of hybrid systems have been proposed [23]. Discrete variables can be introduced into dynamic equations to model the changes in system dynamics. Another possibility is to use discrete components in an otherwise continuous model, e.g. the ideal switch in switched bond graphs [27]. These approaches merge the discrete mode changes and continuous dynamics into one form (one set of equations or one monolithic model). The results are usually compact, but capturing all possible system configurations in one form may make it hard to understand. These approaches often do not give explicit representations to operational modes.

In Ptolemy II, the model structure naturally dictates how a hybrid system is to be modeled [20]. The continuous dynamics of each mode is captured by a CT model (a composite actor with a CT director). The discrete mode changes are modeled by an FSM. A modal model actor contains the CT models of all modes and the FSM. Each state of the FSM represents a mode and has as refinement the CT model of that mode. This modal model matches very well the hybrid I/O automata [21]. The schematic of a modal model in Ptolemy II is shown in figure 7.4. A process is controlled by a modal controller. There are two actors in the top-level CT model. One actor models the process dynamics; the other is a modal model actor. Inside this actor is a two-state FSM. From this FSM, we know that the modal controller employs two alternative control laws, which are modeled by controllers A and B. The conditions of mode changes are annotated on the transitions between states. The conditions are expressed as predicates *p* and *q* on the process state *y*.

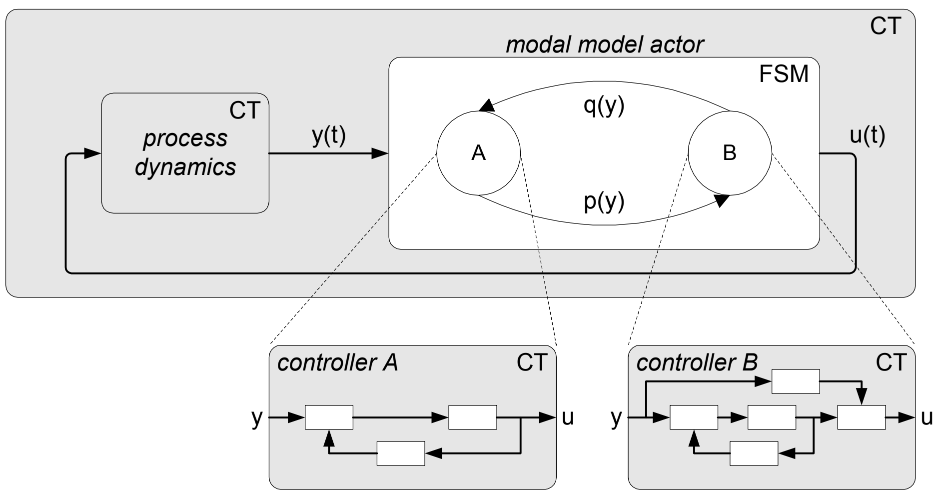


FIGURE 7.4 The schematic of a modal model in Ptolemy II.

The model in figure 7.4 clearly demonstrates the benefits of "orthogonalizing concerns." Modeling discrete mode changes with FSMs yields an easy to understand summary of system operation. In each mode, we can deal with pure continuous dynamics. This is possible in Ptolemy II because of the variety of models of computation supported. When modeling a heterogeneous system, we can choose whichever model of computation that best fits with the aspect or subsystem we are working on. Because of the disciplined hierarchical composition of different models of computation, their properties are preserved in a heterogeneous model.

Given the Ptolemy II infrastructure, it is natural to generalize modal models from hybrid system modeling to *charts [10], which allow the heterogeneous hierarchical composition of FSM and other models of computation. For example, combining FSMs with the synchronous/reactive model of computation yields Statecharts-like models, which are widely used in designing complex discrete control logic. If we combine FSMs with process networks [17], the semantics is similar to that of the Specification and Description Language (SDL) [4], which is widely used in the telecommunications field.

Modal systems from many application domains can be modeled cleanly with *charts. A particularly useful special case of *charts is the composition of FSM and SDF, which we call *heterochronous dataflow* (HDF) [10]. As discussed in section 4.3, SDF models have very desirable properties but only apply to algorithms with a fixed structure. Many components in signal processing systems need to adapt their algorithms to changing environments. For example, in a wireless handset, a simple channel equalization algorithm may suffice when the interference level is low. A more sophisticated algorithm will be used when the interference level becomes high. Such switching of algorithms can be captured by an HDF model. In each mode, the whole signal processing system can still be treated as a hierarchical SDF model.

6 Application - Inverted Pendulum Controller

In this section, we illustrate how the Ptolemy II environment supports the modeling, simulation, and design exploration of control systems, using an inverted pendulum controller as an example. (A more detailed model, a helicopter controller, is described in [19].) Controlling the inverted pendulum is a classical problem in control laboratories because the pendulum dynamics is both nonlinear and unstable. The experimental setup is shown in figure 7.5. The pendulum consists of two moving parts, the arm that rotates in the horizontal plane, and the pendulum that moves in the vertical plane. The states of the pendulum process are the angle and angular velocity of the pendulum, and those of the arm. The process is controlled by the acceleration of arm rotation. Complete equations describing the process dynamics can be found in [1].

FIGURE 7.5 The Furuta inverted pendulum.

A modal controller can be designed to swing up the pendulum and stabilize it in the upright position. The controller has three modes of operation: a swing-up mode, a catch mode, and a stabilize mode. The swing-up controller moves the pendulum from its initial downward position towards the upright position, using a nonlinear energy-based algorithm. When the pendulum comes close enough to the upright equilibrium, the catch controller takes over. The task of the catch mode is to reduce the speed of the pendulum and the arm before the stabilize mode is entered. The catch and stabilize controllers use linear state feedback.

The Ptolemy II model of the modal controller is shown in figure 7.6. The mode switching is modeled by the FSM. (The *init* state of the FSM serves no other purpose than to produce information about the initial mode of the controller.) On the left are the inputs to the FSM, which are the measured states of the pendulum process. For each mode, the computation required by the control law is fixed, and is modeled with SDF.

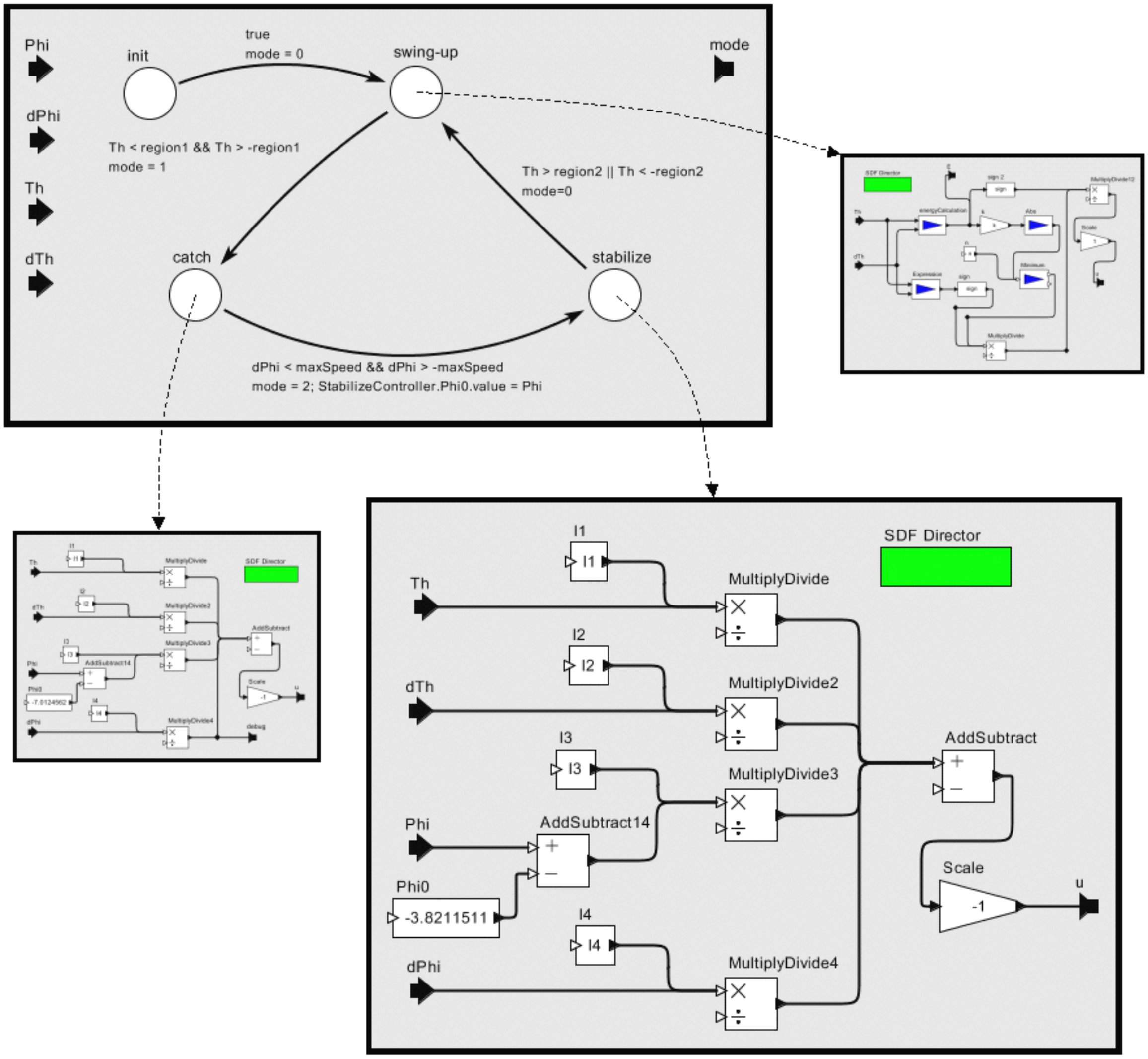


FIGURE 7.6 The modal controller of the inverted pendulum.

When the modal controller is implemented on a computer that controls the pendulum, the whole system becomes a sampled-data system. The Ptolemy II model of such a system is shown in figure 7.7. The *PendulumDynamics* actor is an instance of the *DifferentialSystem* actor in the continuous-time actor library, which can model dynamic equations of the form:

$$\frac{d\vec{x}}{dt} = f(\vec{x}, \vec{u}, t) \qquad \vec{y} = g(\vec{x}, \vec{u}, t) \qquad \vec{x}(0) = \vec{x}_0$$

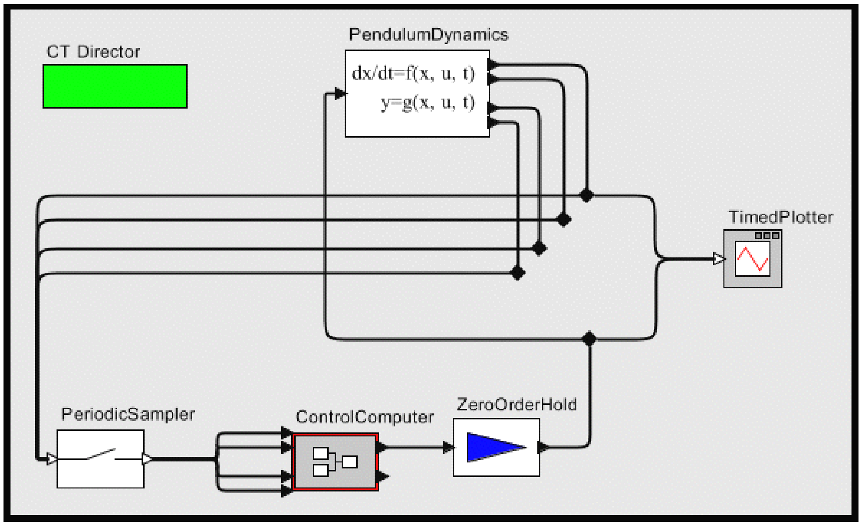


FIGURE 7.7 The sampled-data model of inverted pendulum control.

The states of the pendulum process are periodically sampled. The *ControlComputer* actor contains the modal controller, and computes a control output at each sampling time. The *ZeroOrderHold* actor converts the sequence of control outputs to a continuous-time signal that is fed back to the pendulum process. The *TimedPlotter* actor plots the control signal and the angle of the pendulum.

The sampled-data model in figure 7.7 helps us to verify with simulation that the control laws are effective in a sampled-data system. This is only a first step in the design process from control law specifications to the final implementation, because the model does not capture many issues present in an actual implementation. For example, the model treats the computation in the control computer as taking zero time, and there is no communication delay in the control loop. In Ptolemy II, we can elaborate this model to explore a number of such issues, as shown in figure 7.8.

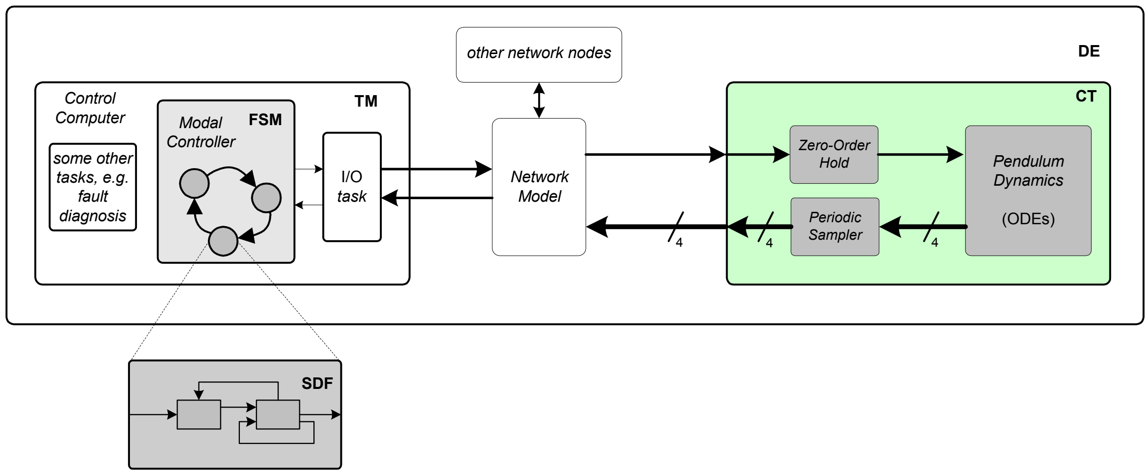


FIGURE 7.8 An elaborate Ptolemy II model that incorporates many implementation issues.

Control software today is often implemented on top of an RTOS. In such a realization, the controller runs as a task in the RTOS and competes with other tasks for resources, e.g. CPU time. This may cause jitter in the input-to-output delay and even changes in the sampling period. We can model these effects with the TM domain in Ptolemy II. As shown in figure 7.8, the control computer is modeled as a composite actor with a TM director. The modal controller actor is treated as a task by the TM director. Some other tasks, such as one for fault diagnosis and one for I/O, may be added to reflect the dynamics of concurrent tasks running in the same RTOS. With such a model, we can evaluate various scheduling mechanisms in terms of control performance.

The computers, sensors and actuators in a control system are often networked. We can model this by using the discrete event domain at the top level. The pendulum model and the control computer are both connected to a network model actor, along with other network nodes to reflect the contention of network resources. The network model simulates the communication among the nodes in the control system, taking into account contention, transmission delay and loss, etc. Such a model helps the designers to evaluate different network architectures and protocols.

7 Conclusions

The modeling and design of complex control systems require tools that support multiple models of computation, and a software architecture to compose heterogeneous systems. The Ptolemy II environment has a component-based hierarchical structure that meets these requirements. Components in Ptolemy II have a fine-grained interface for execution control, which allows directors to compose the execution of components according to various models of computation. All directors provide the same execution interface to the outside domain, so that when different models of computation are composed in a heterogeneous model, their properties are preserved.

Ptolemy II includes a number of domains that are useful in control system design. Other pertinent models of computation are being implemented, such as Petri nets and port-based objects [28]. With modal models, Ptolemy II provides a clean structure for studying modal systems that are common in control engineering.

A framework for studying the dynamic interaction among actors, receivers, and directors is being developed. The framework [18] extends the concept of type systems in programming languages. Interface automata [8] are used to capture the dynamic behavior of components, and the communication protocols between components. Type checking, which checks the compatibility of a component with a certain domain, is conducted through automata composition. When reusable components are used in a model, we can use type checking to study whether the model will generate undesirable behavior, such as deadlock. The framework also helps us to verify that hierarchically composing different models of computation preserves their properties.

Another active development in Ptolemy II is code generation [30]. The structure of Ptolemy II models provides a good foundation for the code generator. At each hierarchical level of the model, the components interact homogeneously according to a specific model of computation. The properties of the model of computation can be used to analyze and optimize the model at that level for efficient code generation. For example, in an SDF model, the actors can be statically scheduled. It is possible to merge the functions of a cluster of actors in the schedule, so as to reduce the number of function calls in the generated code. Different models of computation offer different optimizations, which can be applied orthogonally in a heterogeneous model.

The URL of the Ptolemy project homepage is:

http://ptolemy.eecs.berkeley.edu/

The Ptolemy II software can be downloaded from there, complete with source code and design documentation. Online demonstrations can be viewed with an applet-enabled Internet browser. Most publications from the Ptolemy group are made available in electronic form.

Acknowledgements

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.

References

[1] J. Akesson, “Safe Manual Control of Unstable Systems,” *Master Thesis ISRN LUTFD2/TFRT--5646--SE*, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden, September 2000.

[2] K. J. Astrom and B. Wittenmark, *Computer-Controlled Systems*, 2nd ed., Prentice Hall, 1990.

[3] J. Banks, J. S. Carson, B. L. Nelson, and D. M. Nicol, *Discrete Event System Simulation*, Prentice Hall, August 15, 2000.

[4] F. Belina, D. Hogrefe, and A. Sarma, *SDL With Applications From Protocol Specification*, Prentice Hall, 1991.

[5] A. Benveniste and P. Le Guernic, "Hybrid Dynamical Systems Theory and the SIGNAL Language," *IEEE Tr. on Automatic Control*, Vol. 35, No. 5, pp. 525-546, May 1990.

[6] G. Berry and G. Gonthier, "The Esterel synchronous programming language: Design, semantics, implementation," *Science of Computer Programming*, 19(2):87-152, 1992.

[7] J.Davis II, C. Hylands, B. Kienhuis, E.A. Lee, J. Liu, X. Liu, L.Muliadi, S. Neuendorffer, J. Tsay, B. Vogel, and Y. Xiong, "Heterogeneous Concurrent Modeling and Design in Java," *Technical Memorandum UCB/ERL M01/12*, EECS, University of California, Berkeley, March 15, 2001. (http://ptolemy.eecs.berkeley.edu/publications/papers/01/HMAD/)

[8] L. de Alfaro and T. A. Henzinger, "Interface Automata," to appear in *Proc. of the Joint 8th European Software Engineering Conference and 9th ACM SIGSOFT International Symposium on the Foundations of Software Engineering (ESEC/FSE 01)*, Austria, 2001.

[9] S. A. Edwards, "The Specification and Execution of Heterogeneous Synchronous Reactive Systems," Ph.D. thesis, University of California, Berkeley, May 1997. Available as UCB/ERL M97/31. (http://ptolemy.eecs.berkeley.edu/papers/97/sedwardsThesis/)

[10] A. Girault, B. Lee, and E. A. Lee, “Hierarchical Finite State Machines with Multiple Concurrency Models,” *IEEE Transactions On Computer-aided Design Of Integrated Circuits And Systems*, Vol. 18, No. 6, June 1999.

[11] D. Harel, “Statecharts: A Visual Formalism for Complex Systems,” *Sci. Comput. Program.*, vol 8, pp. 231-274, 1987.

[12] IEEE Computer Society, *IEEE Draft Standard VHDL-AMS Language Reference Manual*, 1997.

[13] KnowNow Inc., “Vision: Powering the Real-Time Enterprise,” (http://www.knownow.com/products/whitepapers/powering.html) 2001.

[14] 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.

[15] T. J. Koo, G. J. Pappas, and S. Sastry, “Multi-Modal Control of Systems with Constraints,” *IEEE Conference on Decision and Control*. Orlando, FL, December 2001.

[16] E. A. Lee and D. G. Messerschmitt, “Synchronous Data Flow,” *Proc. of the IEEE*, vol. 75, no. 9, pp. 1235-1245, September, 1987.

[17] E. A. Lee and T. M. Parks, “Dataflow Process Networks,” *Proc. of the IEEE*, vol. 83, no. 5, pp. 773-801, May, 1995.

[18] E. A. Lee and Y. Xiong, “System-Level Types for Component-Based Design,” *First Workshop on Embedded Software, EMSOFT2001*, Springer-Verlag, Lake Tahoe, CA, USA, Oct. 8-10, 2001.

[19] Jie Liu and Edward A. Lee, “Component-based Hierarchical Modeling of Systems with Continuous and Discrete Dynamics,” *IEEE Symposium on Computer-Aided Control System Design (CACSD'00)*, Anchorage, Alaska, USA, Sep. 2000.

[20] J. Liu, X. Liu, T. J. Koo, B. Sinopoli, S. Sastry, and E. A. Lee, “A Hierarchical Hybrid System Model and Its Simulation,” *38th IEEE Conference on Decision and Control (CDC'99)*, Phoenix, Arizona, Dec. 1999.

[21] N. Lynch, R. Segala, F. Vaandrager, and H.B. Weinberg, “Hybrid I/O automata”, *Hybrid Systems III*, number 1066 in LNCS, pp. 496-510, Springer Verlag, 1996.

[22] The MathWorks, Inc., *Stateflow User’s Guide*, May 1997.

[23] P. J. Mosterman, “An overview of hybrid simulation phenomena and their support by simulation packages,” *Hybrid Systems: Computation and Control '99*, volume 1569 of Lecture Notes in Computer Science, pages 165-177. 1999.

[24] S. Narasimhan, F. Zhao, G. Biswas, and E. Hung, “Fault Isolation in Hybrid Systems Combining Model-Based Diagnosis and Signal Processing,” *Proc. of IFAC 4th Symposium on Fault Detection, Supervision, and Safety for Technical Processes*, Budapest, Hungary, 2000.

[25] OVI, *Verilog-AMS Language Reference Manual, Analog & Mixed-Signal Extensions to Verilog HDL*, Version 1.1, March 13, 1998.

[26] C. S. Sharp, O. Shakernia, and S. Sastry, “A Vision System for Landing an Unmanned Aerial Vehicle,” *International Conference on Robotics and Automation*, Seoul, Korea, May 2001.

[27] U. Soderman and J.-E. Stromberg, “Switched bond graphs: Towards systematic composition of computational models,” *International Conference on Bond Graph Modeling and Simulation (ICBGM'95)*, Volume 27 of Simulation Series, pp. 73-79, SCS Publishing.

[28] D. B. Stewart and P. Khosla, “Chimera methodology: designing dynamically reconfigurable real-time software using port-based objects,” *Proceedings of the Workshop on Object-Oriented Real-Time Dependable Systems (WORDS)*, pp. 46-53, 1995.

[29] C. Szyperski, *Component Software: Beyond Object-Oriented Programming*, Addison-Wesley Pub. Co., January 1998.

[30] J. Tsay, C. Hylands, and E. A. Lee, “A Code Generation Framework for Java Component-Based Designs,” *CASES '00*, San Jose, CA, November 17-19, 2000.