Transition Management for Reconfigurable Hybrid Control Systems

©JENE NELSON AT CN GRAPHICS INC.

By Murat Guler, Scott Clements, Linda M. Wills, Bonnie S. Heck, and George J. Vachtsevanos

Complex control systems often contain numerous controllers (or control laws) for a given plant, where the controller having authority at any given time depends on the current operating condition of the plant. For example, a plant operating in the nominal condition generally uses one controller, while a plant with a fault uses a different controller. Even in nominal operation, there may be several controllers that are designed for different steady-state operating points (for example, a helicopter may have one controller for hover and a different one for takeoff). These types of systems can be modeled as hybrid systems; that is, systems that have both continuous and discrete states. In this case, the plant and the controllers can be modeled using differential or difference equations, which have continuously varying states. The higher-level logic that determines which controller to use can be modeled using discrete states that evolve according to a finite state automaton.

The implementation of hybrid controllers can be facilitated by using component-based software architectures [1], [2], which reduce software development and validation costs. Component-based architectures encourage code reuse across applications. For example, there are many algorithmic methods that are applicable to many different systems: neural networks, Kalman filters, and even PID controllers. These can all be made into software components by standardizing the interfaces of the different modules and by making sure that the components can be composed together to form specific hybrid control systems. Built in this manner, a hybrid control system would require dynamic reconfiguration of the software components (that is, switching and adapting components at run time), with the discrete logic determining which controller component is selected at any given time. Component-based designs are also more adaptable and easier to evolve as the plant changes or as the control requirements for the plant change over time. This type of reconfiguration of the hybrid controller might be done offline.

Wills (linda.wills@ece.gatech.edu), Guler, Clements, Heck, and Vachtsevanos are with the School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30332-0250, U.S.A.

0272-1708/03/$17.00©2003IEEE

Further reduction in code development and validation costs can be achieved by taking advantage of generic patterns [3] in software architectures. With hybrid systems, for example, designers tend to use a select few ways of organizing and reconfiguring their systems. It is beneficial to identify generic ways of integrating and dynamically reconfiguring these systems and to use these patterns to create implementation code that can be reused in different applications. Capturing valid strategies for dynamic reconfiguration leads to robust, highly reliable computing systems. Moreover, it is desirable to design the code so that it can be modified and updated easily whenever new controllers are added; that is, the code implementing the reconfiguration decision logic should be explicitly evolvable and adaptable.

Transition management (TM) is an example of a common practice in hybrid system implementation for which generic patterns can be identified. In particular, practicing engineers often introduce a strategy to smooth transitions during reconfigurations. An abrupt transition (that is, a discrete switch between controllers) can excite high-frequency dynamics in a system, which causes undesired responses and can stress the actuators. A number of transition management strategies that smooth the transition are performed in practice, but they are usually incorporated into the system on an ad hoc basis after the overall system has been designed, rather than early in the process. This often leads to an implementation that does not explicitly preserve the transition logic, making it difficult to change later.

This article discusses generic aspects of transition management that have been identified in hybrid system reconfiguration and that have been derived from the literature, as well as from our experience in building hybrid systems. It focuses on one particular generic pattern that recurs in hybrid system designs to smooth transitions when switching from one configuration to another. It shows how the pattern is supported on the open control platform (OCP), a real-time middleware-based substrate for integrating and reconfiguring control systems [4]-[6]. The article illustrates how this pattern can be instantiated in multiple different contexts that require configuration transitioning.

This transition management pattern encapsulates the reconfiguration decision logic, the structural configuration specifications, and the blending strategies as separate modular entities that can be specified and easily changed by the control system designer. Our goal is to provide reconfiguration flexibility in hybrid control system design and implementation. Most importantly, we would like to maintain the connection between the underlying transitioning model specified when the system was designed (e.g., in state diagrams or Petri net models) and the code that implements the decision logic. By making the decision logic and reconfiguration strategies explicit in encapsulated software components, changes and upgrades to the strategies are made easier, as is validation and modeling of the hybrid system.

To facilitate rapid prototyping of the software implementation, we are using hybrid modeling and simulation techniques from the Ptolemy group [7], [8] at the University of California at Berkeley to specify the transition management strategies. Our goals are to model and enforce valid instantiations and compositions of the generic reconfiguration and transition mechanisms early in the development process rather than patching transition management code into the system in later development stages, which is costly and error prone. We are using Ptolemy II as a rapid prototyping tool with which to express these strategies and validate them through simulation.

Transition management is an example of a common practice in hybrid system implementation for which generic patterns can be identified.

This article first discusses generic aspects of hybrid system reconfiguration, which provide the basis and justification for the transition management support we have developed. Our transition management pattern is then presented, along with background on the OCP on which it is built. The hybrid modeling and simulation of the transition management strategies using Ptolemy are also discussed, showing results that validate the strategies. We describe how the transition management pattern has been applied to gain scheduling and state initialization transition strategies. Finally, conclusions and future directions are given.

Generic Reconfiguration Transition Strategies

We began the process of identifying generic patterns for modeling hybrid systems and for implementing hybrid controllers by reviewing a large number of recent publications (more than 400) on hybrid systems. These papers ranged from theoretical hybrid modeling techniques to applications of hybrid control to real world systems [9]-[19]. A brief compilation of the generic aspects of hybrid systems found from the literature search is given next.

Hybrid systems contain both continuous- and discrete-state variables. Within a given discrete state, the continuous variables evolve according to a set of differential (or difference) equations. Transitions between discrete states are generally modeled with a finite automaton. Of particular interest to the controls engineer is how to design these discrete transitions.

Traditional approaches to the modeling of hybrid control systems concealed the continuous + discrete nature of the

systems by converting them into either purely discrete or purely continuous systems. Recently, efforts have been made to capture the true hybrid nature of a system. A hybrid system is, essentially, an indexed set of dynamical systems along with some means of "switching" between them. These switches are initiated whenever the state satisfies certain conditions, described by the state's inclusion in a specified subspace of the state space. Also, when a transition occurs, there is some means of determining the initial state of the new dynamical system.

One paper that is particularly useful for identifying generic patterns in hybrid systems is [10], in which Branicky proposes a unified framework that encapsulates both the important continuous and discrete dynamics of the system and their interactions. This framework, which is consistent with many different modeling approaches, identifies four transition phenomena common to hybrid systems:

- *autonomous switching*, where the continuous vector field changes discontinuously when the state hits a prescribed boundary
- *autonomous impulses*, where the continuous state changes impulsively on hitting a prescribed region of the state space
- *controlled switching*, where the continuous vector field switches in response to a control command
- *controlled impulses*, where the continuous state jumps in response to a control command.

Thus, a generic framework for hybrid systems should include the above transitions.

Most of the theoretical papers on hybrid systems say little about the actual transition dynamics; that is, the behavior of the continuous-time system while it is transitioning between the different discrete states. It is often assumed that these dynamics take place on a much faster time scale so that the system spends relatively little time in the transition.

On the other hand, many papers that employ some sort of switched control for practical systems are concerned with the transition. One approach pursued by Oishi and Tomlin [19] creates a new discrete state for the transition that incorporates the transition dynamics into it. This approach can create a large number of extra "transition states" if the original hybrid system has a large number of discrete states originally (consider all the combinations of discrete states and the corresponding transitions between them). Another approach to handling transition dynamics is to find a means of smoothing the transition between discrete states without deviating from the original set of discrete states. A common method of achieving this goal is to smooth the control action. For example, in many gain-scheduled control algorithms, controller parameters are switched based on the state's inclusion in regions about local operating points [20]. When the state nears the boundary of two regions, the parameters are "blended" to smooth the transition from one region to the next. Another example of control smoothing is used regularly in sliding mode control, which is a switching control law where a switching surface is defined in the state space. To reduce chatter, the discontinuous part of the control is smoothed in a region around the switching surface.

Toward developing software support for generic hybrid control system mechanisms, we have identified several transition strategies for hybrid controllers: discrete switches, output blending, parameter blending, transient compensation, and initialization.

- *Discrete switch:* The simplest of these strategies is a discrete switch from one controller to another, as shown in Figure 1. This transition strategy can lead to discontinuous compensation signals that excite high-frequency dynamics in the plant.
- *Output blending:* In the output blending strategy, the system begins and ends in the configurations shown in Figure 1(a) and (b), respectively. Between these two stages, however, the system configuration is as shown in Figure 2. An example of the use of this strategy is gain scheduling of static compensators, where scheduling of the outputs is equivalent to scheduling of the controller parameters. A more sophisticated output blending transition strategy is described in [21], where the blender itself has internal states and dynamics.
- *Parameter blending:* When two controllers have a very similar structure, the parameters of the controllers can be blended during the transition. For this case, an additional parameter-setting schedule block must be added to the transition configuration, as shown in Figure 3. The parameters (as well as the input, potentially) may be scheduled based on time or on the plant's state. Gain scheduling [20] of dynamic compensators is a simple, yet common, example of this strategy.
- *Transient management:* Another strategy is transient management [22], where a signal may be added to the compensation signal to try to cancel out unwanted transient characteristics. This configuration, shown in Figure 4, includes a transient compensator block. The reference signal, compensator signal, plant output, and plant state may be used by this block to determine the appropriate signal that should be added to the compensator signal.
- *State initialization:* Finally, compensators might be initialized in a way that alleviates transients during the transition. In particular, the state of dynamic compensators must be initialized before the compensators can be implemented. There are several approaches to state initialization. The simplest one is to start with an initial state of zero. This may not give a smooth transition, but it is easy to implement and does ensure that the actuator does not start in saturation. The transient response can be improved if more thought is put into the state initialization. If the controllers before

and after the switch have the same structure, an approach known as "state preserving" initializes the state of the second controller to the final state of the first controller, thus preserving continuity of the controller state. By extending this idea to systems that do not necessarily have the same structure, it is sometimes possible to match controller outputs, as well as one or more derivatives of the output signal [23], [24].

Having identified some of the generic aspects of hybrid system reconfiguration, the next step is to address the issue of supporting the transition in software. Transitions from one discrete state to another may be abrupt (such as a component failure) or gradual (such as the blending of gains in a gain-scheduled controller). Moreover, in the implementation of a hybrid controller, the transition from one discrete state to another (and potentially, therefore, one control algorithm to a different control algorithm) may require some change in which inputs must be sent to the controller and/or what outputs are generated. In the next section, we describe a generic transition management pattern that addresses these issues.

Transition Management Pattern

We are developing generic reconfiguration support for the common transition management strategies that we have identified in hybrid control systems. This support is in the form of an architectural pattern, called the TM pattern, that captures standard reconfiguration plans, specifying how one mode-related software configuration transitions into another. These patterns provide a generic support structure for transition management strategies (e.g., for managing the collation and blending of signals during gradual component transitions), which can then be instantiated with application-specific functions.

To understand how the TM pattern works, consider a simple example in which two sensor components are being interchanged and their outputs need to be blended during the transition from one to the other. For example, an unmanned aerial vehicle might use GPS to measure altitude at high altitudes, but it might use sonar at low altitudes where sonar is more effective, as shown in Figure 5. As the vehicle descends from a high to low altitude, it is not appropriate to abruptly stop using data from the GPS before starting to rely on the sonar altimeter. Rather, it is better to use a mixed GPS/sonar measurement during the transition. The blending of the GPS and sonar data is more accurate at the in-between altitude range. At these altitudes, the onboard computer should be able to perform sensor fusion by blending these data together to come up with a logical measurement. The TM pattern can be used to coordinate how the sonar and GPS components are connected and disconnected to/from consumers of the altitude data and to control the application of blending functions to maintain a smooth output profile during the transition.

In particular, the pattern encapsulates the following as separate modular entities that can be specified and easily changed by the system designer:

1) signal transition/blender functions, which specify how to collate and/or blend signals during a gradual transition between controller configurations or what transient compensation function to apply
2) signal transition coordinator, which includes the reconfiguration decision logic and determines when transition functions should be applied (e.g., when to start and end blending of signals)

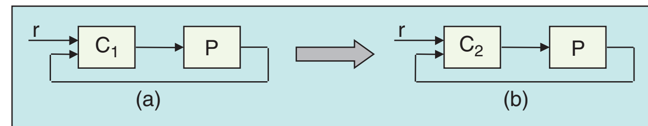

Figure 1. *A discrete switch from (a) controller 1 to (b) controller 2.*

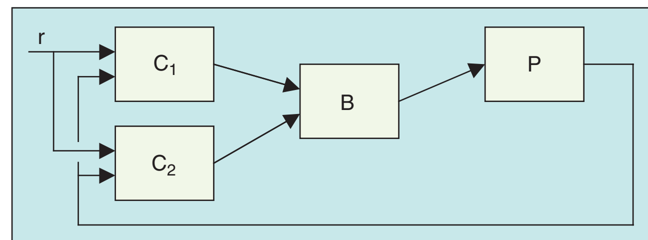

Figure 2. *Blending controller outputs to smooth a transition.*

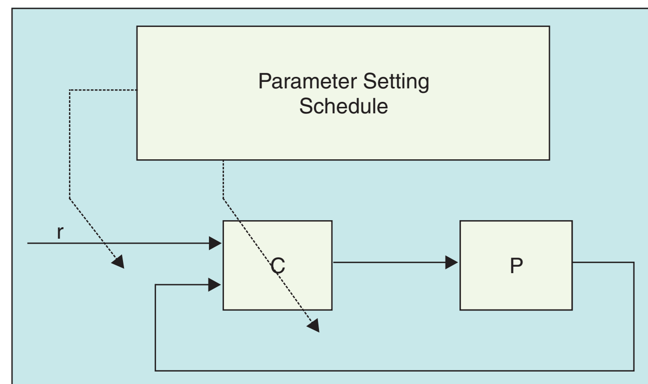

Figure 3. *Gain scheduling of dynamic compensators.*

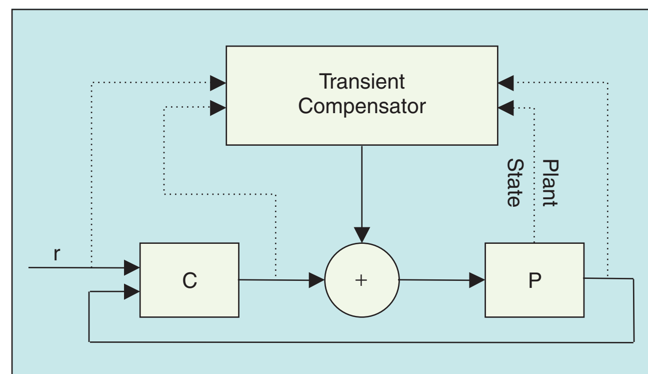

Figure 4. *Using a transient compensation signal to smooth transient response.*

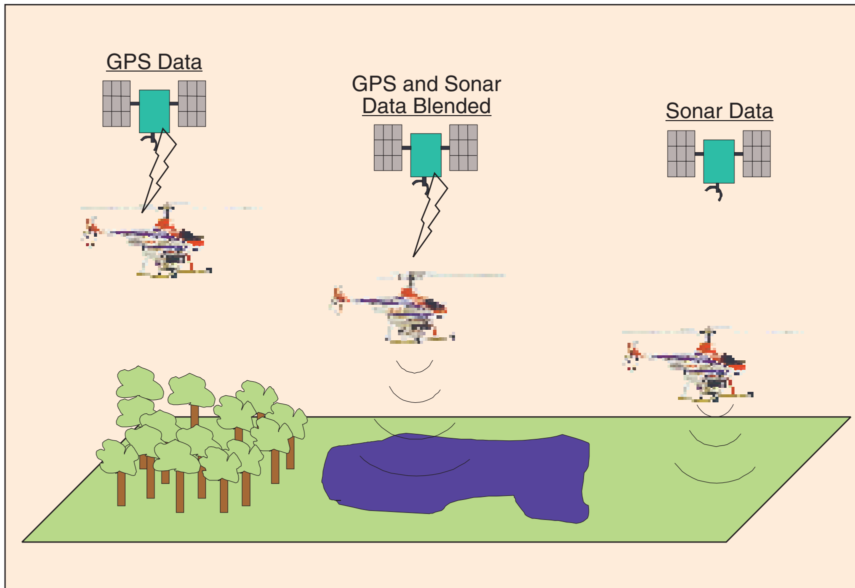

Figure 5. *GPS/sonar transition.*

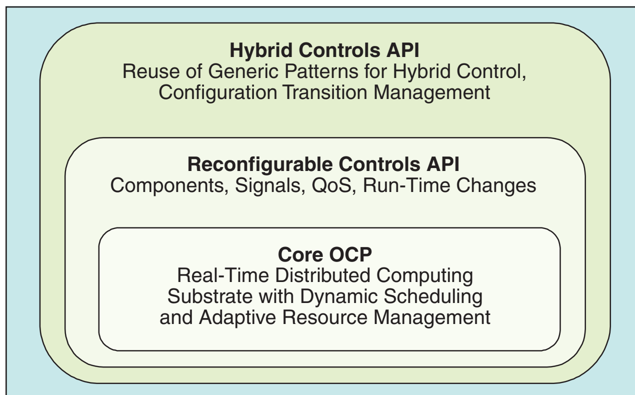

Figure 6. *Layers of the OCP.*

3) signal configurations, which specify alternative ways of structurally connecting components.

Open Control Platform

This generic support for TM is being developed at Georgia Tech to support hybrid control systems as part of a larger effort (in collaboration with Boeing, Honeywell, and the University of California at Berkeley) to create an OCP [4]-[6] based on object-oriented middleware and distributed object computing. It coordinates distributed interaction among diverse control system components and supports dynamic reconfiguration and customization of the components in real time. It specifically provides more comprehensive support than existing tools for integrating distributed components while hiding details of distributed computing from the control system developer. It also moves beyond development-only support to enabling the rapid run-time adaptation and dynamic reconfiguration of control systems. The OCP is being applied to the autonomous control of unmanned aerial vehicles (UAVs) [25]-[28] at Georgia Tech.

The OCP consists of multiple layers of application programmer interfaces (APIs) that increase in abstraction and become more domain specific at the higher layers, as shown in Figure 6. At each level, the abstract interfaces are defined to provide access to the underlying functionality while hiding details of how that functionality is implemented. Each layer builds on the components defined in lower layers. The layers of the OCP are intended to form a bridge from the controls domain to distributed computing and reconfigurability technologies so that controls engineers can exploit these technologies without being experts in computer science.

In the bottommost "core" layer, the OCP leverages from and extends new advances in real-time distributed object computing that allow distributed components to communicate asynchronously in real time [29]-[33]. It also supports highly decoupled interaction among the distributed components of the system, which tends to localize architectural or configuration changes so that they can be made quickly and with high reliability.

The middle "reconfigurable controls" layer provides abstractions for integrating and reconfiguring control system components; the abstractions bridge the gap between the controls domain and the core distribution substrate [28]. The abstract interface is based on familiar control engineering concepts, such as block diagram components, input and output ports, and measurement and command signals. It allows real-time properties to be specified on signals that translate to quality-of-service (QoS) constraints in the core real-time distribution substrate. It also allows run-time changes to be made to these signal properties, which are then handled by lower-level dynamic scheduling and resource management mechanisms [34], [35]. This layer raises the conceptual level at which the controls engineer integrates and reconfigures complex, distributed control systems.

The third "hybrid controls" layer supports reconfiguration management by making reconfiguration strategies and rationale for reconfiguration decisions explicit and reusable. It contains generic patterns of integration and reconfiguration that are found in hybrid, reconfigurable control systems. The TM pattern, which is the focus of this article, is found in the hybrid controls layer. It can be specialized with logic for choosing reconfigurations as well as signal blending strategies for smoothly transitioning from one configuration to another. This is critical to hybrid systems in which continuous dynamics must be maintained between discrete reconfiguration events and where multiple control and blending strategies are applicable.

Transition Management Pattern Implementation

The three parts of the TM pattern (signal transition/blending functions, transition coordinator, and configurations) all exist and run within an OCP component, as shown in the center box of Figure 7. (Boxes labeled C1 through C6 represent components of a control system, such as controllers, plants, sensors, and models.)

The configuration is an important TM constituent that is a composition of low-level user-defined objects, such as controllers, plant models, sensors, and actuators, as shown in Figure 8. As its name implies, each configuration defines a specific layout of a hybrid control system. For example, if we implement the system represented in Figure 1(a) and (b), one configuration could implement what is represented in (a) and another could implement the layout in (b). Configurations have references to the low-level components (e.g., C1, C2, and P) so that components that occur in more than one configuration (such as P) are not duplicated in the system code. Configurations not only specify low-level components but also the dataflow relationships between them. For instance, the fact that C2 sends its output to P has to be specified explicitly in that configuration.

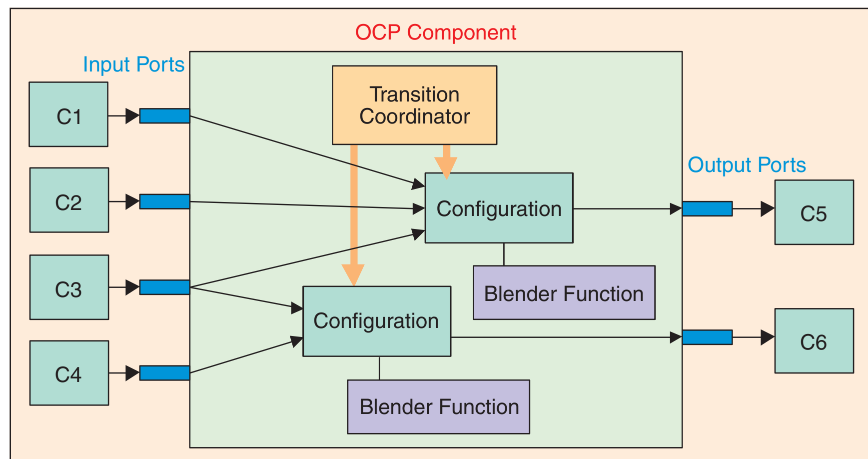

Figure 7. *Structure of a transition manager.*

Signal transition or blending functions are applied to two or more signals and produce an output signal that is some mathematical function of the inputs. For example, a signal transition function that is responsible for blending the sonar and GPS data might take a weighted average of the sonar and GPS inputs as in the following pseudocode:

```
void SignalTransitionFunction()
{Read GPS_Signal and Sonar_Signal;
GPS_weight = function(Signal values);
Sonar_weight = function(Signal values);
Output_Signal = GPS_Weight * GPS_Signal +
   Sonar_Weight * Sonar_Signal;
Return Output_Signal;}
```

The transition coordinator is a user-specified entity that decides when to start or end blending. It oversees the activations of different configurations. A coordinator typically keeps one configuration active at a time, and, depending on its state and the state of the input received, it may decide to switch to another configuration. The coordinator pseudocode for our sonar/GPS example is:

```
Update () {
Determine_Input_Signal_State();
Switch (Blending_State) {
Case CHANNEL 1:
  if (Input_Signal = MEDIUM) then {
    Switch_To_Configuration(CONFIG_BLEND);
    Set_Blending_State(BLEND); }
Case BLEND:
   if (Input_Signal = LOW) then {
    Switch_To_Configuration
    (CONFIG_CHANNEL_1);
    Set_Blending_State(CHANNEL 1);
  else if (Input_Signal = HIGH) then
    Switch_To_Configuration
    (CONFIG_CHANNEL_2);
    Set_Blending_State(CHANNEL 2);
Case CHANNEL 2:
  if (Input_Signal = MEDIUM) then
    Switch_To_Configuration(CONFIG_BLEND);
    Set_Blending_State(BLEND);
}}
```

The transition coordinator can be viewed as a finite state machine. The signal state is a way of discretizing the input data. The coordinator is responsible for assigning a signal state to its own inputs. In our sonar/GPS example, the input (from the GPS) can be low, high, or medium. The blending state is the state of the coordinator, an indication of what the hybrid control system is doing at a given time. In this example, the hybrid system can be either transmitting from Channel 1 (Sonar), Channel 2 (GPS), or blending data from both channels. In each blender state, a different configuration is active. Channel 1 and 2 configurations are just one-input, one-output conduits. The blender configuration is a two-input, one-output configuration that is responsible for blending sonar and GPS data. Depending on its blending state and the state of its input signals at the time of update, the transition coordinator switches to another blending state and activates the necessary configurations.

The TM pattern is implemented as a set of abstract classes that are reused across applications by user-defined specializations of methods for coordinating transitions and

blending data. The relationships among the classes of the TM are shown in the unified modeling language (UML) [36], [37] class diagram in Figure 9. In this diagram, the rectangles signify the classes, some of which may be instantiated as objects in the C++ program. The edges with triangles represent inheritance relationships, where the classes located by the triangle are the parent or abstract classes and the classes on the other end of the line are the derived or implementation classes. The edges with diamonds represent ownership relationships, where the classes next to the diamonds own or contain the classes connected by the edge. For example, a configuration is composed of a set of interconnected controllers, plant models, sensors, etc. The numbers by the class diagrams on the edge lines indicate the numbers of class instances that participate in that relationship. Star (*) means "zero or more" and (1..*) means "one or more." For instance, each coordinator object could have reference to one or more configuration objects, but each configuration object is referenced by only one coordinator object. The colors in our diagram also have meanings, although coloring is not a feature of UML class diagrams. The blue indicates classes that are already defined by the TM system and already exist in the TM library. The green indicates classes that must be created or modified by the user.

As can be seen from Figure 9, the user need only define the user-defined coordinator to plug in the reconfiguration decision logic. The user also has to define the configurations that serve as compositions of user-defined objects (such as low-level controllers) and that also contain user-defined signal transition functions.

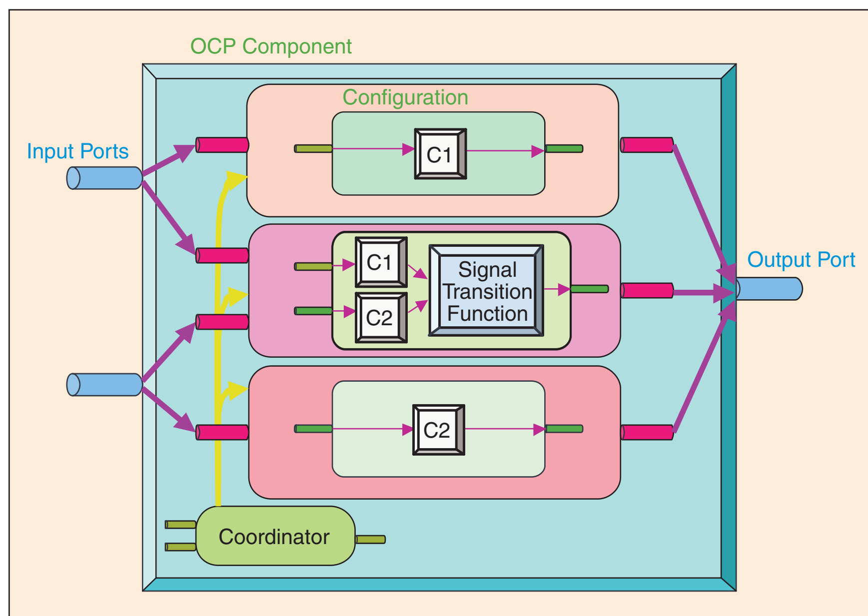

Figure 8. *An example with several configurations.*

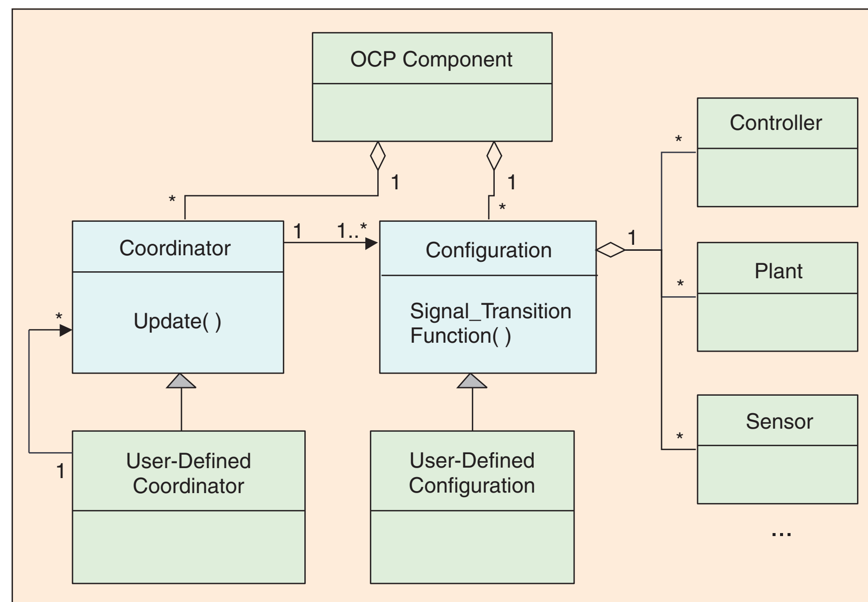

Figure 9. *UML class diagram of transition manager.*

The transition coordinator and the configurations themselves may contain state variables. This enables TM application code to have internal states and dynamics. For instance, the coordinator in the sonar/GPS example keeps track of information about the current hybrid state of the system, which it incorporates into decisions about when to initiate or end a transition. The state transition function within the configuration may also draw on internal states, encapsulated in the configuration object. For example, the blending may be a mathematical function of output parameters of two controllers, weighted by a factor that changes over time and that is maintained as a state variable of the configuration [38].

Distributed Transition Management

A transition coordinator may sometimes oversee other coordinators, even though they may be in separate OCP processes. Such coordinators communicate with each other using their discrete state values. This is a key feature of the TM system: it may be distributed across different processes in support of distributed hybrid control systems. This also makes it possible to implement TM in a hierarchy of coordinators. Such a hierarchy would enable coordinators to read the signal and hybrid states of lower-level coor-

dinators and adjust the hybrid states of themselves and their subordinates. Distributing TM is beneficial to the controls engineer in terms of reducing the cognitive burden of the design. If the system design is more modular, it is much easier to deal with complexity. Some systems themselves can be distributed in their nature. For example, an array of robot arms in a manufacturing plant would require an implementation of a distributed hybrid control system. Each robot arm would have its own branch TM components. Higher-level coordinators can then be implemented to coordinate the overall system by supervising the branch coordinators in each robot arm.

Hybrid Controls API to Transition Management

The hybrid controls API provides a high-level interface to the TM pattern that allows a controls engineer to define the signal transition/blending functions and the transition coordination logic as a finite state machine (FSM). An example is shown in Figure 10.

The first portion of Figure 10 specifies which signal transition functions will be used and gives their signatures (the type information for their inputs and outputs). The signal transition functions that are named refer to procedures defined by the user. The second portion of the figure gives the transition coordinator as a textual description of a finite state machine that specifies the coordination logic. Using the hybrid controls API, a user specifies TM strategies that can be translated directly into OCP code instantiating the TM pattern.

Hybrid Modeling and Simulation of TM Strategies

The TM pattern can be used by directly encoding transition strategies in the hybrid controls API (as in the excerpt shown in Figure 10). However, we also developed a higher-level, rapid prototyping approach [38] that uses a graphical hybrid modeling front end (Ptolemy II) for specifying transition strategies. This enables TM strategies to be specified using familiar graphical modeling representations, such as FSMs, that controls engineers typically use in describing hybrid systems behavior. Its primary advantage is that the specified TM strategies can be validated early in the design process rather than waiting for the full system implementation. Using hybrid modeling and simulation to model TM can iron out interactions between the behaviors of transition strategies in the early stages of development, rather than discovering them and trying to resolve them in the implementation, testing, or later stages, where it is more difficult and costly. Furthermore, we are developing code generation techniques [38] to translate hybrid models of the transition strategies to their implementation in the hybrid controls API (i.e., directly to TM OCP code). This allows reconfiguration strategies to be directly transferred into the control system software without a separate, manual reimplementation step. Thus, model continuity [39], [40] is maintained from hybrid transitioning models to code that preserves the decision logic and reconfiguration strategies. This increases the likelihood that the hybrid system implementation will behave as expected and will avoid unwanted transients and instabilities.

The modeling front end we are using to specify TM strategies is Ptolemy II [7], [8]. This is a heterogeneous modeling and simulation environment developed by the University of California, Berkeley, for modeling systems that embody multiple models of computation (MoC), organized hierarchically. A model of computation describes how a set of components execute and interact with each other. Example MoCs are dataflow and discrete-event models. A hybrid system can be described using a continuous-time (CT) MoC that represents the low-level controllers and an FSM MoC

```
<TransitionManagement OCP_Component="HybridController">
...
<TM_Configuration Name="BlenderFunction">
    <InputPort Name="InPort1" Type="double"/>
    <InputPort Name="InPort2" Type="double"/>
    <OutputPort Name="OutPort1" Type="double"/>
    <Signal_Transition_Function>
        "factor = (((1-factor)*InPort1)+(factor*InPort2)-0.27)/0.6;
        OutPort1 = ((1-factor)*InPort1)+(factor*InPort2);" </ Signal_Transition_Function >
</TM_Configuration>

<TM_Coordinator Name="PlantCoordinator" InitialState="INIT">
    <InputPort Name="InPort1" Type="double"/>
    <InputPort Name="InPort2" Type="double"/>
    <OutputPort Name="OutPort1" Type="double"/>
    ...
    <TM_State Name="C1_REF4" Configuration="Channel1Function">
        <Transition NewState="BLEND">
            <Guard_Condition> "OutPort1&gt;0.27" </Guard_Condition>
            <Set_Action>
                "Channel2Function.Controller2.CopyState(Channel1Function.Controller1);"
            </Set_Action>
        </Transition>
    </TM_State>
    <TM_State Name="BLEND" Configuration="BlenderFunction">
        <Transition NewState="C2_REF4">
            <Guard_Condition> "OutPort1&gt;0.33" </Guard_Condition>
        </Transition>
    </TM_State>
    ...
</TM_Coordinator>
</TransitionManagement>
```

Figure 10. *Hybrid controls API example.*

that represents the high-level discrete coordination logic governing the activation of the controllers.

An example of a hybrid control system designed using Ptolemy II is shown in Figure 11. The upper left window in the figure shows the hybrid FSM model consisting of the continuous controller and the discrete FSM blocks. The reconfiguration behavior is specified in the FSM diagram in the upper right. It models a gain scheduling transition strategy, which is described further in the next section. The controller is specified in the CT domain, as shown in the lower left of Figure 11. The lower right portion of the figure shows a plot of the simulation results generated by Ptolemy for this example. The overall output of the system as one controller is replaced by another is smoothed during the transition between controllers, as described in a later section.

Each of the states in the FSM diagram (upper right of Figure 11) corresponds to a controller in the CT domain by a relationship, called a "refinement," that is explicitly given in the configuration of each finite state. The transition arcs between the finite states specify when and how controller transitions will take place. The guard expression on each transition specifies the conditions that must be true to enable a transition from one state to the next. The set actions on the arcs specify actions to be taken during the transition, such as the initialization of a controller. Usually, the output of the FSM (and the output of the overall hybrid control system) is simply the output from one of the active low-level controllers; however, some transitions require the outputs of several controllers to be blended together. This kind of signal blending can be specified in the output actions of certain finite states, which we call blending states. Such blending states must be designed to have multiple active controllers designated as refinements. (Extensions were made by the Ptolemy group to allow multiple overlapping refinements of states, which made this possible.) Thus, the FSM specifies not only the coordination logic that determines when transitions occur but also the application of signal transition functions during the transitions.

Example Uses of TM Pattern

The TM provides a generic pattern that can be applied to the implementation of a wide range of control techniques, including sensor fusion (as shown in the GPS/sonar example given earlier), simple switching controllers (such as sliding mode control), and the more general class of hybrid controllers. In this section, the utility of the TM is demonstrated by applying it to the classical case of a gain-scheduled controller and to a state-zeroing state initialization transition. It is also applicable to the nonlinear control of a VSTOL aircraft [41] and to a more sophisticated class of mode transition controllers given in [21]. In [38], we also demonstrated its application to

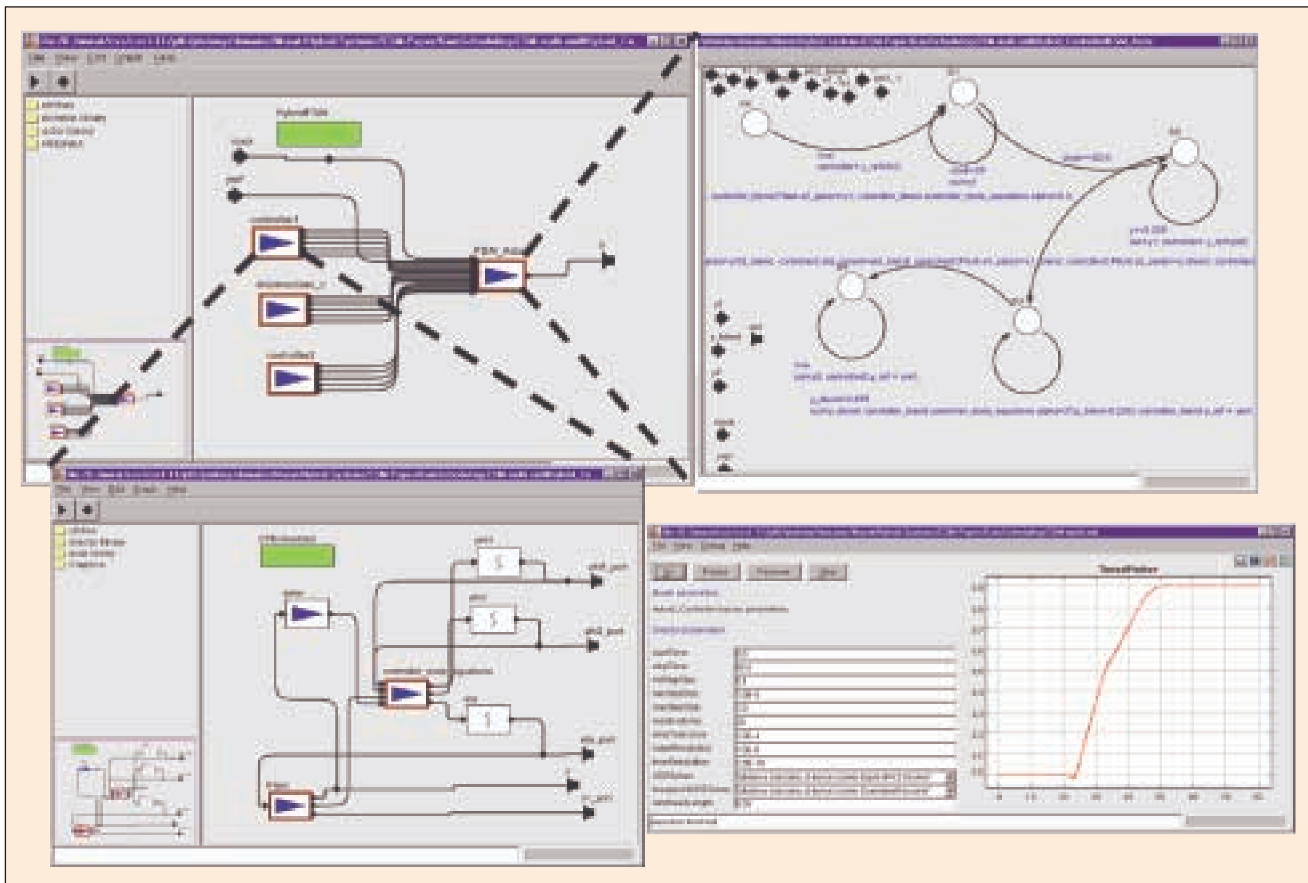

Figure 11. *Modeling a gain scheduling transition strategy in Ptolemy II.*

the composition of a state-preserving transition strategy with an output-signal-blending transition strategy.

Gain-Scheduled Control

Consider a simple gain-scheduled controller, as shown in Figure 12. A nonlinear system, given by $\dot{x} = f(x,u)$, has been linearized about the operating points x_1 and x_2. For each of these points, a corresponding state feedback control law, $u_i = K_i x$, is assumed known. If the state is within a prescribed region, R_i, about an operating point, x_1, the corresponding feedback control law is used. When the state is near the boundary of two operating regions, the gains of the two controllers are scheduled to smooth the transition from one operating region to the next.

In our model, the controller is parametrized by a variable, α, depending on which hybrid state the plant of the system is in. In this example, the discrete states correspond to linearized regions corresponding to $\alpha = 0$ and $\alpha = 0.9$, respectively. When the system is between these two linearized regions, the parameter α is linearly scheduled from 0 to 0.9, as shown in Figure 12. The intent of the scheduling is to smooth the transition and protect the system from step inputs resulting from a sudden change in controllers.

The transition coordination logic, specified in the FSM in Figure 13, chooses the active mechanism based on the proximity of the current state to the operating regions.The signal transition function applied in the transition region is also specified in the FSM of Figure 13 in the output action of blending state S3. The FSM description provides a graphical modeling front end for conveniently specifying TM coordinators and signal transition functions. Using Ptolemy to capture these descriptions allows the designer to simulate and validate the transition strategy before generating OCP TM code implementing it [38].

This gain scheduling example provides a context in which to see how the TM pattern helps to reduce the complexity of managing transitions. In complex applications, it may be necessary to switch between many alternative controllers and use gain scheduling to smooth each transition. It would be unrealistic to set up a separate transition configuration specific to each pair of controllers. Instead, we can define a single generic transition configuration that corresponds to a gain scheduling transition and then dynamically instantiate this configuration at run time for a given specific pair of controllers that are involved in the transition. This collapses multiple transition states having the same coordination logic into a single state in the coordinator and a single transition configuration in the actual implementation. This also helps in reusing system designs in the future; since gain scheduling is a common strategy, the generic configurations created for one application can be directly transferred to another, and the coordination logic is encapsulated so that it can be easily modified for a new application.

Validation

The results of running our gain scheduling example with the TM OCP code are shown in Figure 14 (black curve). These closely match the simulation results predicted by Ptolemy II (see lower right screen in Figure 11). In contrast to the smooth transitioning behavior shown in the black curve, the red curve in Figure 14 shows the behavior of the hybrid system when the controllers are transitioned using a discrete switch rather than the gain scheduling TM strat-

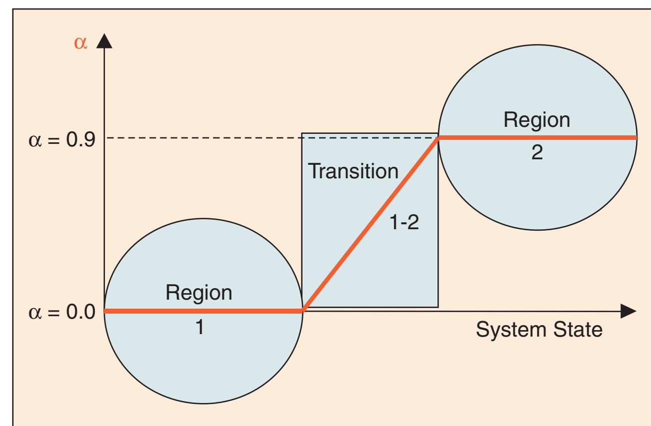

Figure 12. *Linearized regions and controller parameter (α).*

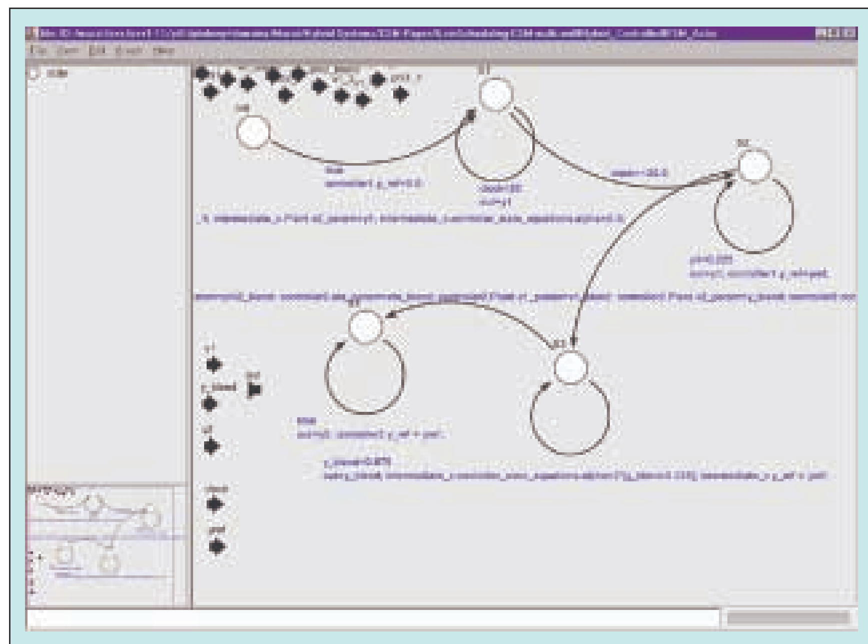

Figure 13. *Transition coordination and blending logic for gain scheduling.*

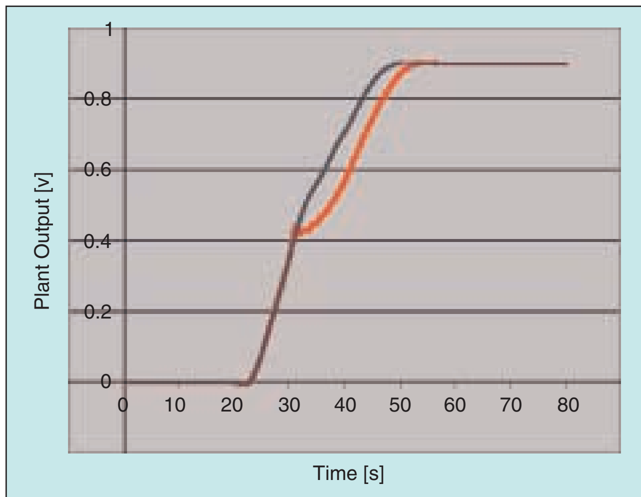

Figure 14. *OCP results of using gain scheduling transition strategy (black curve); and without TM (red curve).*

egy. When the TM strategy is not used, there is a damped ringing in the plant output due to the sudden change of controller parameters. The results using the transition strategy display no ringing and a smoother output, as well as improved rise time.

State Initialization

A second example using the TM pattern involves state initialization. In this scenario, there is a discrete switch between two different controllers. However, instead of trying to blend the outputs in a transition state, the state of the second controller is initialized in a manner that attempts to minimize the impact of the sudden switch. State initialization is a useful transition technique in certain hybrid control applications. Some implementations of low-level controllers may have their integrators saturated during run time. It is advisable that when the system transitions to a new state under such a condition, the new low-level controller starts operating with a brand-new initial state, with all its integrators at a preset initial level (e.g., a state-zeroing initialization strategy would set them to zero).

Figure 15 shows the transition coordinator FSM diagram for the state-zeroing state initialization strategy. In this example, there is no intermediate or "blend" state. Instead, there is a set action on the transition arc that initializes the state of the second controller to zero. The transitions in the state initialization strategies are triggered by either the current time of operation of the overall system or the output level of the plant that is controlled. In Figure 15, the transition occurs when the plant state variable X_2 exceeds 0.1.

The coordinator is responsible for specifying a signal transition function to use and when to use it. Each signal transition function contains the code for initializing the integrators of the low-level controller. The output action in the transition shown in the FSM in Figure 15 gives an example of a function for state zeroing.

Related Work

Recently, a variety of commercial tools have appeared on the market for implementing real-time control systems, including ControlShell/NDDS from Real-Time Innovations, Inc. [42], MATLAB/Simulink from The MathWorks, Inc. [43], and ARCSware from Advanced Realtime Control Systems, Inc. [44]. Most provide simulation and code-generation capabilities that support primarily the development of control systems. They make available to the developer repositories of prebuilt and preverified reusable components (such as common types of controllers, filters, and matrix computations) for efficient construction of systems. The OCP complements the component-level reuse provided by commercial products with reuse of generic integration patterns and reconfiguration strategies found in dynamically reconfigurable hybrid systems.

Simulink/Stateflow [43] is a popular commercial tool suite for prototyping and generating hybrid control systems software. These tools use block diagrams and FSMs to model hybrid systems. They share similarities with related MoCs in Ptolemy II. Both use FSMs to model discrete transitions, where the active state of the FSM enables (or refines) a specified block (or actor) in the model. However, the ability of the FSM chart to change parameters of other blocks is not inherently available using Simulink/Stateflow (as it is with Ptolemy II). This ability is vital in some TM strategies, particularly state initialization, where the state of the newly active controller must be initialized based on the state of the previously active controller.

Real-Time Workshop, an add-on product to Simulink, generates code for deployment on processors or embedded systems. However, the code is monolithic and not meant for modification at the code level. In the context of our UAV application, it is critical that the generated code explicitly preserve the TM strategies and decision logic in a modular, readable form so that they can be modified or replaced (and the change more easily validated) after the code is deployed. These complex systems are composed of many components (newly created and legacy), some of which have been generated using other simulation and modeling platforms. It is important to integrate and reconfigure these in a way that preserves TM and reconfiguration decision logic in the code. TM code by its nature cannot be treated as a monolithic black box in systems into which it is integrated.

Software architecture-oriented approaches have been proposed to manage run-time software evolution [45]-[47]. Oriezy et al. [46], [47] refer to standard reconfiguration strategies as "change application policies," which dictate how to make changes without violating reliability, safety, and consistency constraints. These policies are defined, but no automated support is developed to enforce them.

Kramer and Magee [48], [49] performed seminal work on managing change in distributed software architectures. This work has been augmented by Taentzer, Goedicke, and

Meyer [50], [51], who formalize change management using distributed graph transformation rules. Our work differs from the existing work in that these approaches typically assume either discrete switches, in which the switching time between one configuration and another is not significant to the application, or they assume the components involved in the architectural change are in a quiescent (or silent) state during the reconfiguration. In hybrid systems applications, the transition between configurations is often critical (e.g., in avoiding sharp discontinuities in signals which could disrupt stability). Therefore, we do not make these assumptions but focus on the transition period explicitly. The existing work also assumes a clean separation between the interaction semantics of components in one configuration versus another; there are no hybrid semantics defined for transition periods (because the components are assumed not to be interacting). If we lift the quiescence assumption, hybrid interaction semantics are needed.

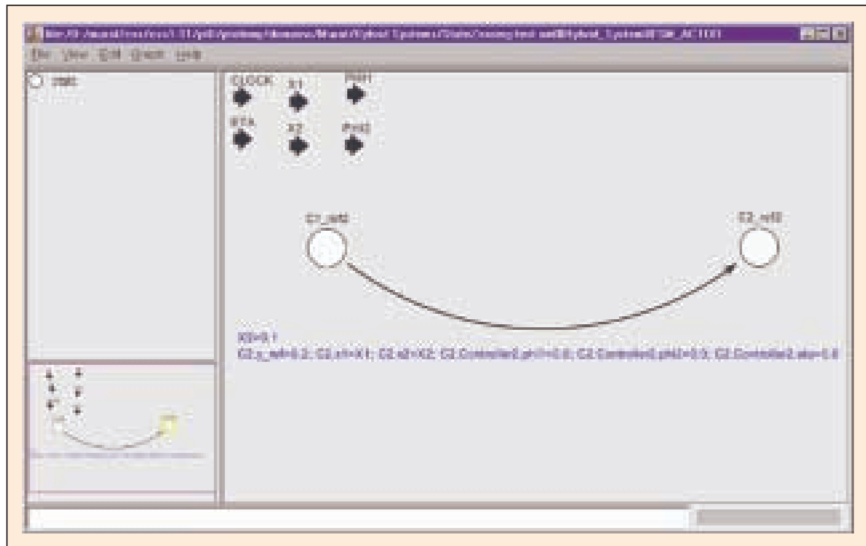

Figure 15. *FSM diagram for state initialization.*

Conclusion and Future Work

We have applied the TM pattern to a wide range of hybrid system reconfiguration examples involving discrete component switches, collated-blending reconfigurations, and transitions requiring component state initialization [41], [52], [53]. Reconfiguration strategies involving transient compensation transitions and transitions requiring component warm-up periods are currently being developed as well. This section discusses additional open areas of future research.

Broadening Modeling Capabilities

FSMs in Stateflow/Simulink and in Ptolemy II allow us to express certain types of common transition strategies in hybrid systems. We would like to broaden the class of transition and reconfiguration strategies that can be modeled and validated. One of Ptolemy's strengths is that it composes heterogeneous models. An FSM model is just one of the types of computation models that can be employed to direct the behavior of controller components and how signals can be routed, rerouted, or blended. We would like to express other types of strategies, e.g., involving real-time QoS parameters and fault-tolerance constraints. These will require using additional models of computation to specify and validate the strategies (e.g., the newly developed timed multitasking domain) [55].

Increasing Robustness

Efforts are ongoing to make the OCP TM software more robust to broken transmissions and time-outs at the inputs. This work will benefit from integration with a new transition service being developed by Boeing [4] for monitoring the status of configuration changes and globally coordinating them. To handle unexpected conditions that may arise during a transition, it must also be possible to interrupt and safely back out of a transition.

Additional Modeling Front Ends

In addition to Ptolemy II, there are other modeling front ends that might be helpful in specifying TM behavior and dynamic reconfigurations. For example, like Ptolemy II, GME's [54] actor-based semantics fit very well with the component-based OCP semantics. Stateflow/Simulink is also a candidate for certain types of strategies, as previously discussed. Supporting multiple front ends would increase the applicability and portability of the OCP's hybrid controls support, allowing controls engineers to work in a modeling and simulation environment that is familiar to them.

Acknowledgments

We are grateful to Edward Lee, Jie Liu, and the members of the Ptolemy group for their adaptations to Ptolemy II that made it possible to model blending transitions. We would also like to acknowledge the contributions of Cameron Craddock, Eric Johnson, Suresh Kannan, Nidhi Kejriwal, Freeman Rufus, J.V.R. Prasad, and Daniel Schrage of Georgia Tech and Brian Mendel at Boeing. We also appreciate the insightful comments from the anonymous reviewers of this article. This work is supported by the DARPA Software-Enabled Control (SEC) program under contracts 33615-98-C-1341 and 33615-99-C-1500 and by the NSF Embedded and Hybrid Systems program under contract

CCR-0209179. We gratefully acknowledge DARPA, NSF, AFRL, and Boeing Phantom Works for their continued support. We have benefited greatly from the guidance of John Bay (DARPA), Helen Gill (NSF), and Bill Koenig (AFRL).

References

[1] G.T. Heineman and W.T. Councill, *Component-Based Software Engineering: Putting the Pieces Together*. Reading, MA: Addison-Wesley, 2001.

[2] C. Szyperski, *Component Software: Beyond Object-Oriented Programming*. Reading, MA: Addison-Wesley, 1998.

[3] R. Johnson, E. Gamma, R. Helm, and J. Vlissides, *Design Patterns, Elements of Reusable Object-Oriented Software*. Reading, MA: Addison-Wesley, Oct. 1998.

[4] J. Paunicka, D. Corman, and B. Mendel, "A CORBA-based middleware solution for UAVs," in *Proc. 4th IEEE Int. Symp. on Object-Oriented Real-Time Distributed Computing (ISORC 2001)*, Madgeburg, Germany, 2001.

[5] J. Paunicka, B. Mendel, and David Corman, "The OCP—An open middleware solution for embedded systems," in *Proc. American Control Conf.*, Arlington, VA, 2001, pp. 3345-3350.

[6] L. Wills, S. Kannan, S. Sander, M. Guler, B. Heck, J. Prasad, D. Schrage, and G. Vachtsevanos, "An open platform for reconfigurable control," *IEEE Contr. Syst. Mag.*, pp. 49-64, June 2001.

[7] J. Liu, X. Liu, and E. Lee, "Modeling distributed hybrid systems in Ptolemy II," in *Proc. 2001 American Control Conf.*, Arlington, VA, June 2001, pp. 4984-4985.

[8] X. Liu, J. Liu, J. Eker, and E. Lee, "Heterogeneous modeling and design of control systems," *Software-Enabled Control: Information Technology for Dynamical Systems,* T. Samad and G. Balas, Eds., New York: IEEE, 2003.

[9] P. Antsaklis and A. Nerode, "Hybrid control systems: An introductory discussion to the special issue," *IEEE Trans. Automat. Contr.*, vol. 43, pp. 457-460, Apr. 1998.

[10] M. Branicky, V. Borkar, and S. Mitter, "A unified framework for hybrid control: Model and optimal control theory." *IEEE Trans. Automat. Contr.*, vol. 32, pp. 31-45, Jan. 1998.

[11] R. Fierro, F. Lewis, and A. Lowe, "Hybrid control for a class of underactuated mechanical systems," *IEEE Trans. Syst., Man, Cybern. A.*, vol. 29, pp. 649-654, Nov. 1999.

[12] E. Frazzoli, M. Dahleh, and E. Feron, "A hybrid control architecture for aggressive maneuvering of autonomous helicopters," in *Proc. 38th Conf. Decision and Control*, Phoenix, AZ, Dec. 1999, pp. 2471-2476.

[13] H. Garcia, A. Ray, and R. Edwards, "A reconfigurable hybrid system and its application to power plant control," *IEEE Trans. Contr. Syst. Technol.*, vol. 3, pp. 157-170, June 1995.

[14] X. Koutsoukos, "Supervisory control of hybrid systems," *Proc. IEEE*, vol. 88, pp. 1026-1049, July 2000.

[15] Y.-H. Liu, K. Kitagaki, T. Ogasawara, and S. Arimoto, "Model-based adaptive hybrid control for manipulators under multiple geometric constraints," *IEEE Trans. Contr. Syst. Technol.*, vol. 7, pp. 97-109, Jan. 1999.

[16] A. Nerode and W. Kohn, "Models for hybrid systems: Automata, topologies, controllability, observability,"in *Hybrid Systems* (Lecture Notes in Computer Science, vol. 736), R. Grossman, A. Nerode, A. Ravn, and H. Rischel, Eds. New York: Springer-Verlag, 1993, pp. 317-356.

[17] A. Arehart and W. Wolovich, "Bumpless switching in hybrid systems," in *Hybrid Systems IV* (Lecture Notes in Computer Science, vol. 1273), P. Antsaklis, W. Kohn, A. Nerode, and S. Sastry, Eds. New York: Springer, 1997, pp. 1-17.

[18] D. Mignone, A. Bemporad, and M. Morari, "A framework for control, fault detection, state estimation, and verification of hybrid systems," in *Proc. American Control Conf.*, San Diego, CA, June 1999, pp.134-138.

[19] M. Oishi and C. Tomlin, "Switched nonlinear control of a VSTOL aircraft,"in *Proc. 38th Conf. Decision Contr.*, Phoenix, AZ, Dec. 1999, pp. 2685-2690.

[20] R. Nichols and R. Reichert, "Gain scheduling for H-infinity controllers: A flight control example," *IEEE Trans. Contr. Syst. Technol.,* vol. 1, pp. 69-79, June 1993.

[21] F. Rufus and G. Vachtsevanos, "Design of mode-to-mode fuzzy controllers," *Int. J. Intell. Syst.*, vol. 15, no. 7, pp. 657-685, 2000.

[22] G. Simon, T. Kovacshazy, and G. Peceli, "Transient reduction in control loops in case of joint plant-controller reconfiguration," in *Proc. IEEE Instrument. Measure. Technol. Conf.,* Budapest, Hungary, May 21-23, 2001, pp. 1172-1176.

[23] G. Simon, T. Kovacshazy, and G. Peceli, "Transients in reconfigurable control loops," in *Proc. IEEE Instrument. Measure. Technol. Conf.*, IMTC/2000, Baltimore, MD, vol. 3, May 1-4, 2000, pp. 1333-1337.

[24] G. Simon, T. Kovacshazy, and G. Peceli, "Transient management in reconfigurable systems," in *Proc. IWSAS 2000* (Lecture Notes in Computer Science, vol. 1936), P. Robertson, H. Shrobe, and R. Laddaga, Eds. New York: Springer-Verlag, 2000, pp. 90-98.

[25] S. Kannan, C. Restrepo, I. Yavrucuk, L. Wills, D. Schrage, and J.V.R. Prasad, "Control algorithm and flight simulation integration using the open control platform for unmanned aerial vehicles," in *Proc. 18th Digital Avionics Systems Conference (DASC),* St. Louis, MO, Oct. 1999, pp. 6.A.3-1 - 6.A.3-10.

[26] F. Rufus, S. Clements, S. Sander, B. Heck, L. Wills, and G. Vachtsevanos, "Software-enabled control technologies for autonomous aerial vehicles," in *Proc. 18th Digital Avionics Systems Conf.*, St. Louis, Oct. 1999, pp. 6.A.5:1-8.

[27] D. Schrage and G. Vachtsevanos, "Software-enabled control for intelligent UAV's," in *Proc. 1999 Int. Conf. Contr. Applicat.*, HI, Aug. 22-27, 1999, pp. 528-532.

[28] L. Wills, S. Sander, S. Kannan, A. Kahn, J.V.R. Prasad, and D. Schrage, "An open control platform for reconfigurable, distributed, hierarchical control systems," in *Proc. 19th Digital Avionics Systems Conf. (DASC-2000)*, Philadelphia, PA, Oct. 2000, pp. 4.D.2-1 - 4.D.2-8.

[29] D. Levine, S. Mungee, and D. Schmidt, "The design and performance of real-time object request brokers," *Comput. Commun.,* vol. 21, pp. 294-324, Apr. 1998.

[30] D. Levine, C. Gill, and D. Schmidt, "Dynamic scheduling strategies for avionics mission computing," in *Proc. 17th Digital Avionics Syst. Conf.,* vol. 1, pp. C15/1-8, 1998.

[31] D. Schmidt, D. Levine, and T. Harrison, "The design and performance of a real-time CORBA event service," in *Proc. OOPSLA'97*, Atlanta, GA, 1997, pp. 184-200.

[32] D. Schmidt and F. Kuhns, "An overview of the Real-Time CORBA specification," *IEEE Computer*, vol.33, pp. 56-63, June 2000.

[33] Object Management Group, "CORBA 2.2 common object services specification," [Online]. Available: http://www.omg.org

[34] M. Cardei, I. Cardei, R. Jha, and A. Pavan, "Hierarchical feedback adaptation for real time sensor-based distributed applications," in *Proc. 3rd IEEE Int. Symp. Object-Oriented Real-Time Distributed Comput., (ISORC)*, 2000, pp.181-188.

[35] B. Doerr, T. Venturella, R. Jha, C. Gill, and D. Schmidt, "Adaptive scheduling for real-time, embedded information systems," in *Proc. 18th Digital Avionics Syst. Conf.*, St. Louis, MO, Oct. 1999, p.2.D.5/9.

[36] H. Eriksson and M. Penker, *UML Toolkit*. Somerset, NJ: Wiley, 1997.

[37] G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide*. Reading, MA: Addison-Wesley, 1999.

[38] M. Guler, S. Clements, N. Kejriwal, L. Wills, B. Heck, and G. Vachtsevanos, "Rapid prototyping of transition management code for reconfigurable control systems," in *Proc. 13th IEEE Int. Workshop on Rapid Systems Prototyping (RSP)*, Darmstadt, Germany, July 2002, pp. 76-83.

[39] R. Janka, *Specification and Design Methodology for Real-Time Embedded Systems.* Boston, MA: Kluwer Academic, 2001.

[40] R. Janka, L. Wills, and L. Baumstark, "Virtual benchmarking and model continuity in prototyping embedded multiprocessor signal processing systems," *IEEE Trans. Software Eng.*, vol. 28, no. 9, pp. 832-846, Sept. 2002.

[41] M. Guler, S. Clements, L. Wills, B. Heck, and G. Vachtsevanos, "Generic transition management for reconfigurable hybrid control systems," in *Proc. 20th Amer. Contr. Conf. (ACC-2001)*, Arlington, VA, June 2001 [CD-ROM].

[42] G. Pardo-Castellote and S. Schneider, "The network data delivery service: Real-time data connectivity for distributed control applications," in *Proc. IEEE Int. Conf. Robotics and Automation*, vol. 4, pp. 2870-2876, 1994.

[43] The MathWorks: Real-Time Workshop [Online]. Available: http://www.mathworks.com/products/rtw/

[44] Advanced Realtime Control Systems, Inc. Available: [Online], Sept. 29, 2000. http://www.arcsinc.com

[45] P. Feiler and J. Li, "Consistency in dynamic reconfiguration," in *Proc. 4th Int. Conf. Configurable Distributed Systems (ICCDS)*, Annapolis, MD, 1998, pp. 189-196.

[46] P. Oreizy, N. Medvidovic, and R. Taylor, "Architecture-based runtime software evolution," in *Proc. Int. Conf. Software Eng. (ICSE)*, Kyoto, Japan, 1998, pp. 117-186.

[47] P. Oreizy, M. Gorlick, R.N. Taylor, D. Heimbigner, G. Johnson, N. Medvidovic, A. Quilici, D. Rosenblum, and A. Wolf, "An architecture-based approach to self-adaptive software," *IEEE Intell. Syst.*, vol. 14, pp. 54-62, May/June 1999.

[48] J. Kramer and J. Magee, "The evolving philosophers problem: Dynamic change management," *IEEE Trans. Software Eng.*, vol. SE-16:11, pp. 1293-1306, 1990.

[49] J. Kramer and J. Magee, "Analysing dynamic change in software architectures: A case study," *Proc. 4th Int. Conf. Configurable Distributed Systems (ICCDS)*, Annapolis, MD, 1998, pp. 91-100.

[50] G. Taentzer, M. Goedicke, and T. Meyer, "Dynamic change management by distributed graph transformation: Towards configurable distributed systems, in *Proc. TAGT*, Paderborn, Germany, 1998, pp. 179-193.

[51] G. Taentzer, M. Goedicke, and T. Meyer, "Dynamic accommodation of change: Automated architecture configuration of distributed systems," in *Proc. Automated Software Engineering (ASE99)*, Cocoa Beach, FL, Oct. 1999, pp. 287-290.

[52] M. Guler, L. Wills, S. Clements, B. Heck, and G. Vachtsevanos, "Support for dynamic reconfiguration of hybrid systems for UAV control," in *Proc. OOPSLA 2001 Workshop on Engineering Complex Object-Oriented Systems for Evolution*, Tampa, FL, Oct. 2001 [Online]. Available: http://www.dsg.cs.tcd.ie/ecoose/oopsla2001/papers.shtml

[53] M. Guler, L. Wills, S. Clements, B. Heck, and G. Vachtsevanos, "A pattern for gradual transitioning during dynamic component replacement in extreme performance UAV hybrid control systems," in *Proc. OOPSLA 2001 Workshop on Patterns and Pattern Languages for Object-Oriented Distributed Real-Time and Embedded Systems*, Tampa, FL, 2001 [Online]. Available: http://www.cs.wustl.edu/~mk1/realtimepatterns/oopsla2001/submissions/muralguler.pdf

[54] A. Ledeczi, M. Maroti, A. Bakay, G. Karsai, J. Garrett, C. Thomason, G. Nordstrom, J. Sprinkle, and P. Volgyesi, "The generic modeling environment," in *Proc. Workshop Intelligent Signal Processing*, Budapest, Hungary, May 2001 [CD-ROM].

[55] J. Liu and E.A. Lee, "Timed multitasking for real-time embedded software," *IEEE Contr. Syst. Mag.*, vol, 23, pp. 65-75, Feb. 2003.

Murat Guler is a Ph.D. candidate in the School of Electrical and Computer Engineering at the Georgia Institute of Technology. He received a B.S. degree in electrical and computer engineering from the Georgia Institute of Technology in 2000, with highest honors. His research area is reconfigurable software for real-time embedded and autonomous systems.

Scott Clements is a Ph.D. candidate in the School of Electrical and Computer Engineering at the Georgia Institute of Technology. He received an M.S. degree in electrical engineering from the Georgia Institute of Technology in 1998 and a B.S. degree in electrical engineering from Mississippi State University in 1996. He is a member of the IEEE and IEEE Control Systems Society. His research interests are in fault-tolerant control and intelligent systems.

Linda M. Wills is an assistant professor of electrical and computer engineering at the Georgia Institute of Technology, where she holds the Demetrius T. Paris Professorship. She received her S.B. (1985), S.M. (1986), and Ph.D. (1992) degrees from the Massachusetts Institute of Technology. She is general chair of the IEEE International Workshop on Rapid System Prototyping (RSP2003) and served as program chair of RSP2001. She has also served as general chair and program chair of the Working Conference on Reverse Engineering. She is a member of IEEE, ACM, and the IEEE Computer Society. Her research interests are in automated software reengineering and reuse, reconfigurable embedded software, and software parallelization for portable multimedia systems.

Bonnie S. Heck is professor of electrical and computer engineering at the Georgia Institute of Technology. She received her bachelor's degree in electrical engineering from the University of Notre Dame in 1981. She received a master's degree in mechanical and aerospace engineering from Princeton University in 1984 and a Ph.D. in electrical engineering from Georgia Tech in 1988. She was an engineer at Honeywell Inc. from 1983 to 1985. She has been a member of the IEEE Control Systems Society since 1988 and has served in several organizational capacities since that time, including serving on the Board of Governors. Her research interests include industrial applications, power electronics, real-time computing and control, and nonlinear control design.

George J. Vachtsevanos is a professor of electrical and computer engineering at the Georgia Institute of Technology. He received a B.E.E. degree from the City College of New York, an M.E.E. degree from New York University, and a Ph.D. degree from the City University of New York. He directs the Intelligent Control Systems Laboratory at Georgia Tech, where faculty and students are conducting research in intelligent control, diagnostics and prognostics for conditioned-based maintenance, and vision-based inspection and control of industrial and bioengineering systems and manufacturing systems. He has published more than 250 papers in his area of expertise. He is a Senior Member of IEEE. He serves as the associate editor of the *International Journal of Intelligent and Robotic Systems* and is a consultant to government agencies and industrial organizations.