

A Pattern for Gradual Transitioning during Dynamic Component Replacement in Extreme Performance UAV Hybrid Control Systems

*M. Guler, L. Wills, S. Clements, B. Heck, G. Vachtsevanos
School of Electrical and Computer Engineering
Georgia Institute of Technology, Atlanta, GA 30332*

Introduction: Transition Strategies in Hybrid Control Systems

A common strategy for control systems is to design a set of controllers for a given plant (i.e., the system to be controlled). The controller that is chosen to be implemented at any given time depends on the current operating condition of the plant. For example, a plant operating in nominal condition generally uses one controller, while a plant with a fault uses a different controller. The predefined controllers are usually modeled using differential or difference equations. The states corresponding to these controllers are continuous in nature. The higher level logic that determines which controller to use can be modeled using discrete states that evolve according to a finite state automaton. Such control systems that have both continuous and discrete states are commonly termed *Hybrid Control Systems*.

Transitions between the discrete states correspond to switching between the continuous controllers. Of particular interest to the controls engineer is how to design these transitions. An abrupt transition (that is, a discrete switch) can excite high frequency dynamics in a system, which causes undesired responses and can stress the actuators [2]. In some cases, it is assumed that these dynamics take place on a much faster time scale so that the system spends relatively little time in the transition. However, in practice, most control engineers attempt to alleviate these perturbations to the system whenever possible. One approach pursued by Oishi and Tomlin [3] creates a new discrete state for the transition, which 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). Other approaches to handling transition dynamics are to find a means to smooth 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 by blending the parameters of the controllers during the transition between them (such as done with scheduling the gains of the controllers). Another example of control smoothing is initializing the new controller with the parameters of the old one before facilitating the switch [4].

As a result of these smoothing strategies, the transition from one discrete state to another may be done gradually. In other cases, the transition may be necessarily abrupt (such as due to a sudden component failure). 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 this paper, we present a generic transition management pattern that addresses these issues.

Transition Management Pattern

Intent

To capture a generic set of transition strategies commonly used by controls engineers in order to ensure a graceful hybrid controls system transition. To enable smooth transition and expression of transition strategy in any kind of online software reconfiguration activity.

Also Known As

Data Blender

Motivation

An extreme performance Unmanned Aerial Vehicle (UAV) system is a practical example of a Hybrid Control System, where different controllers with varying dynamical characteristics are switched among

each other at run-time, without introducing instabilities to the overall system. In the case of the UAV System, the controllers work in a hard real-time environment. The UAV in question may engage in extreme performance maneuvers, where the system must be able to perform rapid online reconfiguration and react promptly to environmental changes and unpredictable events during flight, such as avoiding a moving obstacle or recovering from vehicle equipment failures. System components run on embedded processors placed in the various parts of the UAV and ground station system, hence having a distributed nature [1]. A key challenge these systems face is being able to transition between different controller configurations rapidly while maintaining stability. The Transition Management pattern captures a generic set of transition strategies commonly used by controls engineers to ensure a graceful transition during reconfiguration [2].

Applicability

- Use the TM pattern when you have a component based system where you're doing online reconfiguration of components, and want to express the semantics of the transition operation, as well as ensuring system stability during the transition.
- You have a reconfigurable hybrid controls system with dynamic transition characteristics that you wish to define.
- You have to implement various voting algorithms on data coming from an array of sensors [10].

Structure

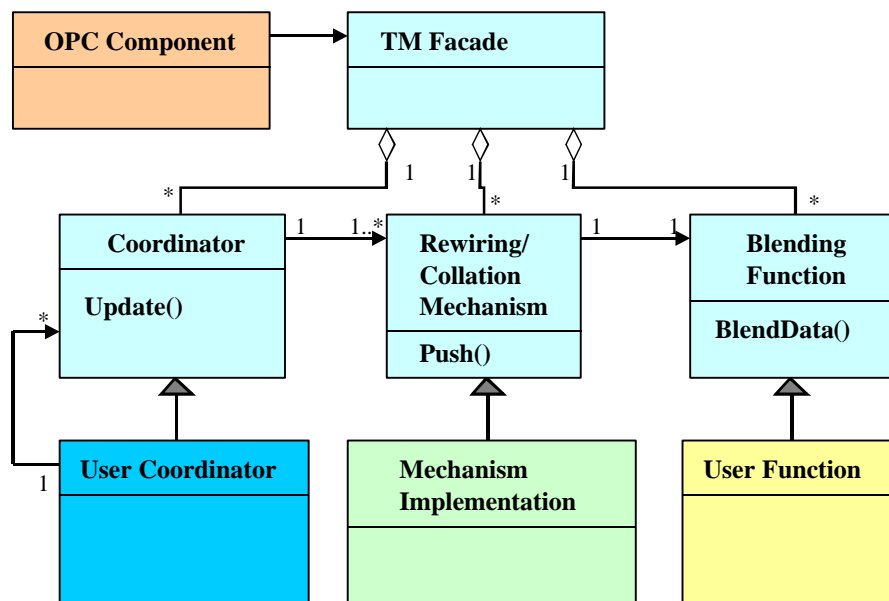


Figure 1: Structure of the Transition Management Pattern.

Participants

- **OCP Component**
 - Connection-oriented programming based Component with input and output ports that run on a CORBA Event Service. Contains the generic Hybrid Controls System Logic [9].
- **TM Façade**
 - Contains all the other TM components.
- **Rewiring/Collation Mechanism**
 - Is responsible for taking care of input and output connections of the Transition Manager.
- **Blender Function**

- Processes the controls data that goes through the TM System.
- **TM Coordinator**
 - Oversees the operation of other TM Constituents, namely the Rewiring Mechanisms and Blender Functions.
- **User Coordinator and User Function**
 - User implementations of the TM Coordinator and Blender Function classes.
- **Mechanism Implementation**
 - Implementation of the abstract Rewiring Mechanism class. The input and output ports as well as the data types being transmitted are specified in this implementation class.

Collaborations

- The OCP Component requests the TM Façade to create the Rewiring Mechanism Implementation class while passing in the user defined TM Coordinator and Blender Function classes.

Implementation

The Transition Management (TM) pattern uses a connection-oriented component-based approach [9]. The control algorithms are encapsulated within what we call the hybrid systems components, or OCP Components, which in fact are object instantiations of CORBA Event-Service based middleware components that receive data through the input ports, and send processed data through the output ports. Thus, the pattern must have the capability to denote the change in input and output connections as well as the change in the encapsulated control algorithms during a transition within a hybrid control systems component [2].

The Transition Manager is made up of three primary constituents as shown in Figure 1: the Rewiring Mechanism, the Blender Function Container, and the Transition Coordinator. All these constituent exist and run within a hybrid systems component. The Rewiring Mechanism is responsible for taking care of input and output connections of the Transition Manager. It makes use of the existing component ports, which the user must create and supply to the Transition Manager. An example Rewiring Mechanism has two or more input ports and one or more output port. The Rewiring Mechanism in the current Transition Management software is automatically generated from the user's specification of input/output ports for a particular Blending Function.

The Blending Function is a user-defined object which includes a method for getting the data from the specified hybrid system component input ports, processing it, and sending the result to the specified output ports. The function to process the data can be a simple differential equation, or a more complicated fuzzy neural network. That is determined by the controls engineer. In the TM pattern, each Blending Function is paired up with a specific Rewiring Mechanism, which in turn is responsible for the timing and activation of its Blending Function.

The Transition Coordinator is a user-specified entity that decides which hybrid system controller to implement, and whether or not to apply data blending between the switching of controllers. It oversees the operations of several Rewiring Mechanisms. A Coordinator typically keeps one mechanism – and therefore one Blending Function - active at a time, and depending on its state and the state of the input received, it may decide to activate another mechanism. The Coordinator can be modeled as a finite state machine. It has its blending state, which is an indication of which Rewiring Mechanism is active at a given time. Depending on the discrete state of its input, also called the signal state, the Coordinator may switch to a different blending state, during which the Coordinator deactivates the current Rewiring Mechanism and activates the next designated one.

The operation of the Transition Management System is as follows. When the Rewiring Mechanism receives data from all its inputs, it sends an update to its Transition Coordinator. The Coordinator, by inspecting the input data and its own state, decides whether to keep the current Rewiring Mechanism active or to activate another one in the next period. After updating the Coordinator, the Rewiring Mechanism calls the Blender Function Container, which in turn blends the input data. Finally, the Rewiring Mechanism sends the resulting data from its output port.

The TM Constituents exist within a class named TM Façade. All the communication between the Hybrid System Component and the TM Constituents are handled by the TM Façade. The TM Constituent classes are in fact created by the TM Façade, as the user passes in the references to the Blender Function and Transition Coordinator implementation classes.

A Transition Coordinator may sometimes oversee other Coordinators, even though they may be located on separate Hybrid System Components. Such coordinators communicate with each other using their discrete state values. This is a key feature of the Transition Management: it may be distributed across different processes in support of distributed hybrid control systems. Distributing the Transition Management System 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 Controls system. Each robot arm would have its own Transition Manager constituents. But to coordinate the overall system, a Main Coordinator would have to be implemented to supervise the Local Coordinators in each robot arm [2].

Known Uses

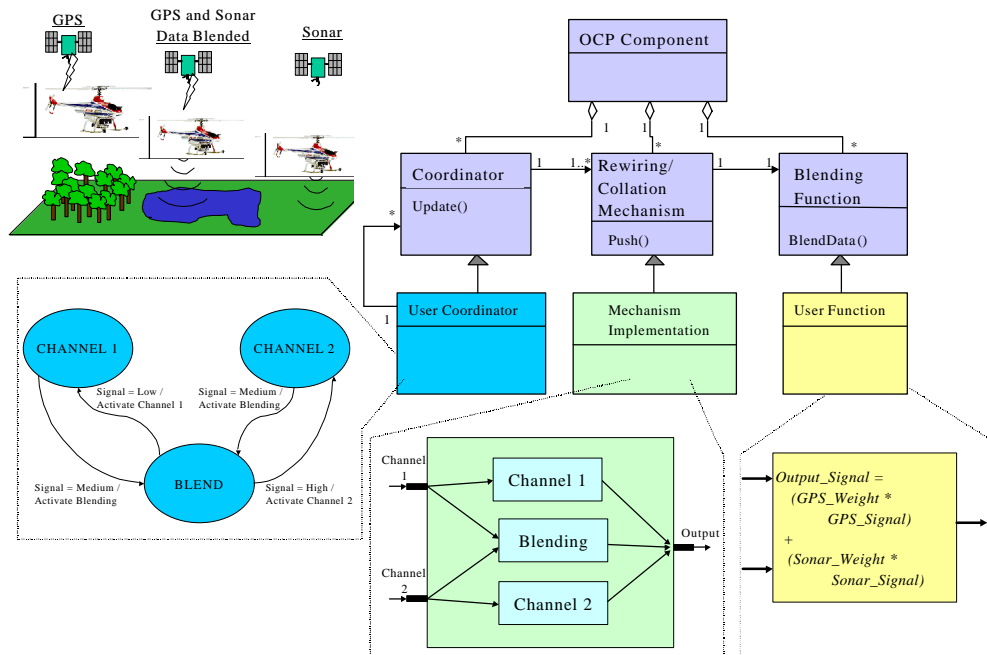


Figure 2: Sonar/GPS Example

The utility of the Transition Manager can be demonstrated through 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 aerial vehicle might use GPS to measure altitude at high altitudes and sonar at low altitudes. 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 this 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 transition management pattern can be used to organize how the sonar and GPS components are connected and disconnected to/from consumers and to control the application of blending functions to maintain a smooth output profile during the transition. In this example, the controls engineer would need to design a Blending Function to

specify how data from the two different sensors would be blended together, and also would need to design the Coordinator to specify when to start and stop applying the data blending [2]. See Figure 2.

Besides being utilized in sensor fusion, the TM can be used in online transition of Hybrid System Controllers, which was its original design purpose. When an Unmanned Aerial Vehicle is executing its direct autonomous flight, and then suddenly intends to make a turn, it needs to switch from its direct flight controller to its right-turn controller. Each of these controllers can be implemented as a Blending Function, while the high level decision system that's responsible for initiating the turn can be implemented as the TM Coordinator. As in the previous example, it is up to the Controls Engineer to design and code what goes inside the Blending Function and TM Coordinator.

Transition Management Pattern can be applied to a myriad of online dynamic reconfiguration cases, where a software component (or its runtime object instantiation) should be replaced by another at runtime. A good example to this would be the Fault Tolerance System of the UAV, where if a mission-critical software sub-component of the UAV stops functioning at run-time, it must be replaced by another sub-component. The various sub-components can be implemented as Blending Functions, while the Fault Detection Algorithm can be embedded in the TM Coordinator. It is important to mention at this point that such a Fault Tolerance system can be implemented under any kind of real-time system, not just a UAV.

Related Patterns

The TM defines the abstract Rewiring Mechanism, Blending Function and Transition Coordinator classes that has references to each other. The implementations of these classes are extended from the abstract ones, either by the user specification as in the case of the Blending Function and the Transition Coordinator, or by the TM pattern itself as in the case of the Rewiring Mechanism. At the run-time of the software, the abstract class references point to the actual implementation classes, thus enabling the use of polymorphism. Since these references can be changed at run-time to other implementation, this also facilitates the use of run-time reconfiguration. This is in fact realization of a well-known software pattern known as "the Strategy Pattern" [5]. In addition to that, the TM pattern is also related to the Façade and Factory patterns. The TM Constituents exist within a class named TM Façade. This is an example to the Façade Pattern [5], where a collection of different classes are accessed through only one class named Façade that contains them. Since the TM Constituent classes are created by the TM Façade, as the user passes in the references to the Blender Function and Transition Coordinator implementation classes, we can say the TM Façade also serves as a BuilderFactory Pattern [5].

Status and Conclusion

The TM pattern is currently implemented upon a software infrastructure known as the "Open Control Platform" (OCP) [6]. OCP is based on object-oriented middleware and distributed object computing and it's being developed as a collaboration effort of Georgia Tech, Boeing, Honeywell, and University of California at Berkeley. OCP is being created to aid the development of complex control systems that coordinates distributed interaction among diverse hybrid control systems components and supports dynamic reconfiguration and customization of these components in real time. Its primary goals are to accommodate rapidly changing application requirements, easily incorporate new technology (such as new hardware platforms or sensor technology), interoperate in heterogeneous environments, and maintain viability in unpredictable and changing environments. 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 3. In this way, OCP forms a bridge from the controls domain to distributed computing and reconfigurability technology so that controls engineers can exploit these technologies without being an expert in computer science, particularly object-oriented or network programming.

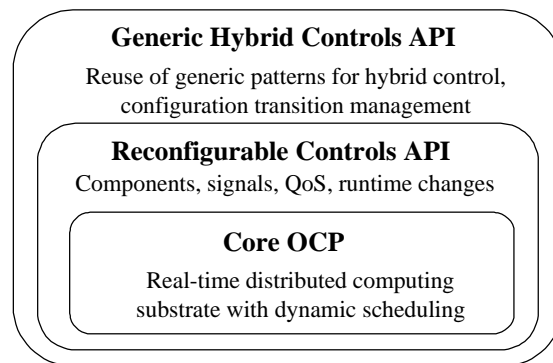


Figure 3: The Structure of the Open Control Platform

At its lowest layer, OCP utilizes the Real Time CORBA technology, also known as TAO, developed by Washington University [7]. TAO is a real-time distributed object computing system which allows distributed, heterogeneous components to communicate asynchronously in real time, and also supports highly decoupled interaction between the distributed components of the system. The middle “reconfigurable controls” layer of the OCP provides basic abstractions for integrating and reconfiguring control system components. The abstract interface is based on familiar controls 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 constraints in the core real-time distribution substrate. It also allows runtime changes to be made to these signal properties, which are then handled by lower level dynamic scheduling and resource management mechanisms. The third “hybrid controls” layer of the OCP supports reconfiguration management by making reconfiguration strategies and rationale for reconfiguration decisions explicit and reusable. It is this third layer that contains generic patterns of integration and reconfiguration that are found in hybrid, reconfigurable control systems, such as the Transition Management pattern.

Currently the ongoing efforts of upgrading the Transition Management System involve designing a more robust implementation to handle broken transmissions and time-outs at the inputs. This work will benefit from integration with a new transition service being developed by Boeing for monitoring the status of configuration changes and globally coordinating them.

Also, as we identify and capture more generic patterns and reconfiguration strategies, a number of challenging issues must be addressed. One is that the patterns are applicable at multiple levels of the controls hierarchy and support must be given for the hierarchical composition of these patterns. This composition may be greatly facilitated by leveraging from existing hybrid modeling and simulation techniques, such as those developed in [8]. By appealing to diverse and interacting models of computation to formally define interaction semantics for generic integration and reconfiguration patterns, it is possible to validate the composition and use of these generic patterns for hybrid controls.

Acknowledgments

This work is supported under the DARPA Software Enabled Control Program under contracts #33615-98-C-1341 and #33615-99-C-1500. We gratefully acknowledge DARPA's Software Enabled Control Program, AFRL, and Boeing Phantom Works for their continued support. We have benefited greatly from the guidance of Helen Gill and John Bay (at DARPA), and Bill Koenig (at AFRL). We would also like to acknowledge the contributions of Suresh Kannan, Freeman Rufus, and Sam Sander of Georgia Tech and Brian Mendel at Boeing.

References

[1] Wills, L., Sander, S., Kannan, S., Kahn, A., Prasad, J.V.R., and Schrage, D., “An Open Control Platform for Reconfigurable, Distributed, Hierarchical Control Systems,” *Proceedings of the 19th Digital Avionics Systems Conference*, pp. 4.D.2-1 - 4.D.2-8, Philadelphia, PA, October 2000.

- [2] Guler, M., Sander, S., Clements, S., Rufus, F., Wills, L., Heck, B., and Vachtsevanos, G., "Generic Mechanisms for Creating Dynamically Reconfigurable Hybrid Control Systems in the Open Control Platform," in *Proceedings of 20th American Control Conference (ACC-2001)*, Arlington, VA, (on CD-ROM), June 2001.
- [3] Oishi, M. and C. Tomlin, "Switched Nonlinear Control of a VSTOL Aircraft," *Proc. 38th Conf. Decision and Control*, Phoenix, Arizona, December 1999, pp. 2685-2690.
- [4] Branicky, Michael, et al. "A Unified Framework for Hybrid Control: Model and Optimal Control Theory." *IEEE Trans. Automatic Control*, Vol. 32, No. 1, January 1998, pp. 31-45.
- [5] R. Johnson E. Gamma, R. Helm, and J. Vlissides, *Design Patterns, Elements of Reusable Object-Oriented Software*. Reading, MA: Addison-Wesley, October 1998.
- [6] Wills, L., Kannan, S., Sander, S., Guler, M., Heck, B., Prasad, J.V.R., Schrage, D., and Vachtsevanos, G., "An Open Software Infrastructure for Reconfigurable Control Systems," *IEEE Control Systems Magazine*, pp. 49-64, June 2001.
- [7] Schmidt D., Levine D., and Harrison T., "The Design and Performance of a Real-time CORBA Object Event Service," *Proceedings of OOPSLA '97*, Atlanta, Georgia, October, 1997.
- [8] J. Liu, L. Xiaojun, T. Koo, B. Sinopoli, S. Sastry, and E. Lee, "A hierarchical hybrid system model and its simulation," *Proceedings of the 38th IEEE Conference on Decision and Control*, vol. 4, pp. 3508-13, 1999.
- [9] Szyperski, C., *Component Software, Beyond Object-Oriented Programming*, New York, NY: Addison-Wesley, 1998.
- [10] Bakken, D. E., Zhan, Z., Jones, C. C., Karr, D. A., "Middleware Support for Voting and Data Fusion," *Proceedings of The International Conference on Dependable Systems and Networks*, Goteborg, Sweden, July, 2001.