Modeling the Simplex Architecture using Charon *

Rafael Fierro †, Yerang Hur ‡, Insup Lee ‡, Lui Sha**
General Robotics Automation, Sensing, and Perception (GRASP) Laboratory †
Department of Computer and Information Science ‡
University of Pennsylvania
Department of Computer Science **
University of Illinois at Urbana-Champaign
e-mail:rfierro@grasp.cis.upenn.edu {yehur,lee}@cis.upenn.edu lrs@cs.uiuc.edu

Abstract

Charon is a language for the hierarchical and modular specification of interacting hybrid systems and is based on the notions of agent and mode. It has well-defined formal semantics, and can be used to specify precisely hybrid systems that exhibit continuous behaviors within modes and discrete jumps between modes. The goal of the Simplex Architecture is to support the reliable upgrade of control software on the fly, and thus, it requires delicate and complex mode switching between different controllers. Due to complexity, it is a quite challenging task to show the correctness of control software built using the Simplex Architecture. As a first step toward addressing the challenge, this paper shows that how the Simplex Architecture can be specified in Charon using the well-known inverted pendulum system.

1 Introduction

An embedded system consists of a collection of components that interact with each other and with their environment through sensors and actuators. Embedded software is used to control these sensors and actuators and to provide application-dependent functionality.

A key aspect of embedded systems is that they can be viewed as hybrid systems that exhibit both discrete and continuous behaviors. These systems become harder to develop due to increasing complexity and functionality. Furthermore, these systems are frequently used in safety critical environment, and thus, require high assurance in reliability. The design and implementation of hybrid systems remains a challenging task. To facilitate the design of hybrid systems, Charon has been developed to allow hierarchical specification of interacting hybrid systems [1].

After a system has been implemented and deployed, the system is often updated to incorporate improvements and new capabilities. Modifications to a legacy control system are, however, costly because it is often hard to predict how the modifications affect the legacy code. Furthermore, it is time-consuming to shut down and restart the system to upgrade it [3]. Thus, it would be desirable to do such a job, while the system is running, without sacrificing reliability. Several architectures allowing on-line software upgrades have been proposed in the last few years and also become commonly available in web-based labs where a remote user is allowed to test his/her controller design on an actual experiment. These technologies have been hardly used in production-level systems because safety and reliability are difficult to guarantee. One of those efforts that begins to be accepted by industry is the Simplex Architecture [3].

This paper illustrates how the Simplex Architecture can be specified in Charon, and this is the first step in validating the correctness of control software built based on the Simplex Architecture.

2 Modular Specification of Hybrid System in Charon

Charon [1] is a language for modular specification of interacting hybrid systems based on the notions

*This research was supported in part by ARO DAAG55-98-1-0393, ARO DAAG55-98-1-0466, DARPA ITO MOBIES F33615-00-C-1707, DARPA ITO MARS 130-1303-4-534328-xxxx-2000-0000, NSF CCR-9988409, NSF CISE-9703220, and ONR N00014-97-1-0505 (MURI).

of agent and mode. For hierarchical description of the system architecture, CHARON provides the operations of instantiation, hiding, and parallel composition on agents, which can be used to build a complex agent from other agents. The discrete and continuous behaviors of an agent are described using modes. For hierarchical description of the behavior of an agent, CHARON supports the operations of instantiation and nesting of modes. Continuous behavior can be specified using differential as well as algebraic constraints, and invariants restricting the flow spaces, all of which can be declared at various levels of the hierarchy. The modular structure of the language is not merely syntactic, but also on the semantics so that it can be exploited during analysis.

Architectural and behavioral hierarchy. The building block for describing the system architecture is an *agent* that communicates with its environment via shared variables and also communication channels. The language supports the operations of *composition* of agents for concurrency, *hiding* of variables for information encapsulation, and *instantiation* of agents for reuse. The building block for describing a flow of control inside an atomic agent is a *mode*. A mode is basically a hierarchical state machine, that is, a mode can have submodes and transitions connecting them. Variables can be declared locally inside any mode with standard scoping rules for visibility. Modes can be connected to each other through well-defined entry and exit points. We allow the *instantiation* of a mode so that the same mode definition can be reused in multiple contexts. Finally, to support *exceptions*, the language allows group transitions from default exit points that are applicable to all enclosing modes, and to support *history retention*, the language allows default entry transitions that restore the local state within a mode from the most recent exit.

Discrete and continuous variable updates. Discrete updates are specified by *guarded actions* labeling transitions connecting the modes. Such updates correspond to mode-switching, and are allowed to modify variables through assignment statements. Variables in CHARON can be declared as type *analog*, and they flow continuously during the continuous updates that model the passage of time. The evolution of analog variables can be constrained in three ways: *differential* constraints (e.g., $x' = f(x, u)$), *algebraic* constraints (e.g., $y = g(x, u)$), and *invariants* (e.g., $x - y < c$) which limit the allowed durations of flows. Such constraints and invariants can be declared at different levels of the mode hierarchy.

Example. The problem of balancing an inverted pendulum has received considerable attention by the control [2] and computer science [4] communities. Figure 1 illustrates a typical inverted pendulum where θ represents the angular position of the pole with respect to the vertical axis, x represents the cart position, u_f is the control input, g is the gravity constant, and l is the pole length. The masses of cart and pole are M and m, respectively. The control objective is to move the cart from one position to a desired target position *target* maintaining the pendulum at the upright position. If one considers physical *hard* constrains (e.g., length of the track, available power supply), then this control task becomes nontrivial. Controllers are to stabilize the state of the system about $[x_s,\ 0,\ 0,\ 0]$, where x_s is the desired target position. Figure 2 shows the overall inverted pendulum system, which consists of IPControllers, IPDecisionModule, and IPPhysicalSystem. IPControllers provides control signals to IPDecisionModule that implements the control switching logic of the Simplex Architecture. Figure 3 describes IPPhysicalSystem. IPPhysicalSystem uses an experimentally identified nonlinear

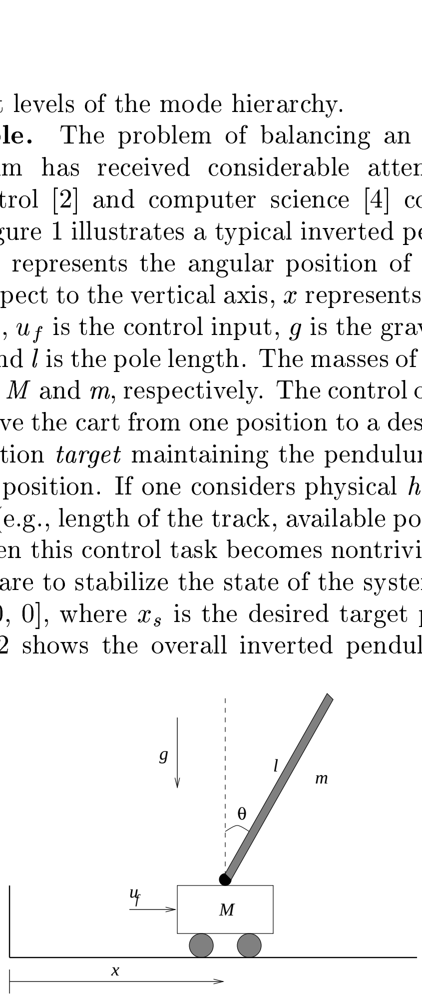


Figure 1: The Inverted Pendulum System

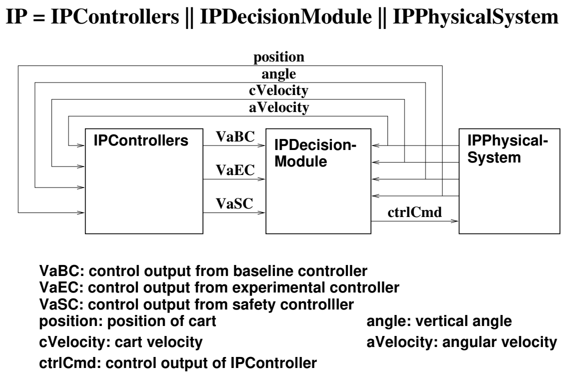


Figure 2: The Overall Structure of Inverted Pendulum System

model of actual physical inverted pendulum. It reads the output of IPDecisionModule and updates the system state. The system state comprises `position, angle, cVelocity`, and `aVelocity`. `position, angle, cVelocity`, and `tVelocity` are the position of the cart, the vertical angle of the inverted pendulum, the velocity of the cart, and the angular velocity of the inverted pendulum, respectively. `VaBC, VaEC`, and `VaSC` are control output of the controllers. `ctrlCmd` is the output selected by the decision module. We use nonlinear model of the actual inverted pendulum. Each component of IP is defined as an

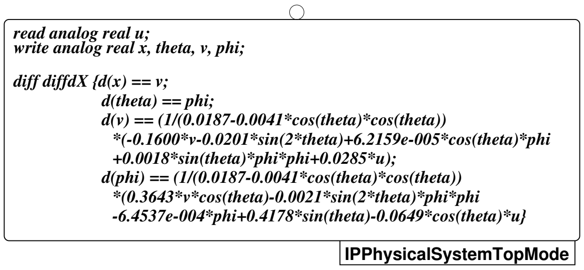


Figure 3: Top Mode of IPPhysicalSystem

agent running in parallel and communicating through shared variables as shown in Figure 2.

3 The Simplex Architecture of Inverted Pendulum

The goal of the Simplex Architecture is to make it possible for developers to upgrade control software, maintaining system reliability and availability, while the system is running [3]. The Simplex Architecture consists of three controllers: experimental controller (EC), safety controller (SC), and baseline controller (BC). EC is an experimental controller whose reliability is unknown or not trusted. SC is a stable controller that is known to be reliable and safe, but whose performance is not optimal. BC is another stable controller that has a smaller stability region and meets the required performance. Control system developers can use the Simplex Architecture to test or deploy a different experimental controller on the working system with safe guard against unstability of the new controller.

In the Simplex Architecture, an experimental controller controls the system as long as the system is in a desirable state. If the system enters an undesirable state, the safety controller takes over the control. Once the system state gets in the operating region of the baseline controller, the system will switch to the baseline controller to achieve a required performance. Switching to a different controller is done by the decision module that selects output from the active controller.

3.1 Three Inverted Pendulum Controllers

IPControllers comprises three different controllers: controllerBC, controllerEC, and controllerSC as shown in Figure 4. Each of these corresponds to baseline controller, experimental controller, and safety controller of the Simplex architecture. Figure 5 shows

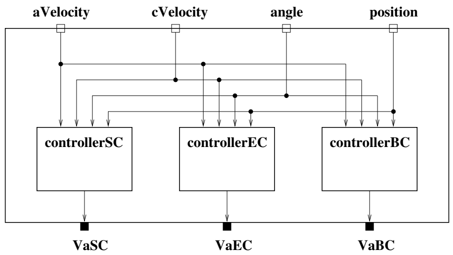


Figure 4: IPControllers

the generic controller in CHARON with global variables `x, theta, v, phi`, and `Va`. When it is instantiated, these variables are renamed. Global variables `x, theta, v`, and `phi` are *read* variables renamed to `position, angle, cVelocity`, and `aVelocity`. `Va` is a *write* variable that will be renamed to `VaBC, VaEC`, and `VaSC` for different controllers. The value of `Va` is computed by the *algebraic constraint algeControl*.

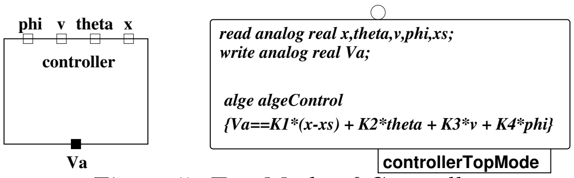
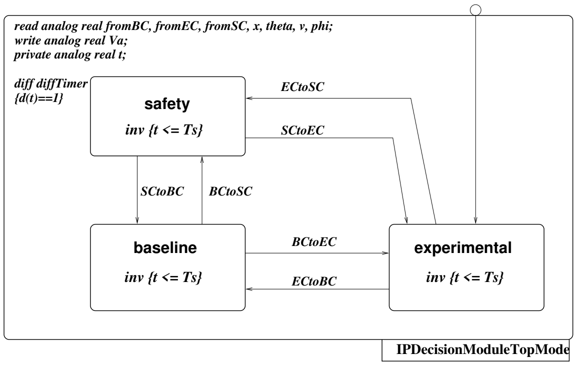


Figure 5: Top Mode of Controller

3.2 The Decision Module

Figure 6 describes the behavior of IPDecisionModule based on the state transition between the three controllers [3, 4]. The control switching logic depends on the boolean variables detecting timing faults and semantic faults. The timing faults are denoted by the variables *bc_ready* and *ec_ready*, which are set to *true* or *false* depending on whether or not the baseline and experimental controllers produce valid output by their deadline, respectively. When the IP state is near the boundary of the recovery region within a given threshold, we say a control fault has occurred. If a control fault has occurred and there is no information to identify what causes the control fault, a semantic fault from EC (BC) is assumed to have occurred, if EC (BC) has the control at that time. Transitions of CHARON are associated with




```
trans BCtoSC from baseline.toSC to safety.frBC
 when t == Ts and
  (!Function.safe(x,theta,v,phi,t,Ts) or
  (!Function.bcReady() and !Function.ecReady()))
 do {Va = fromSC; t=0}
trans BCtoEC from baseline.toEC
             to experimental.frBC ...
trans ECtoBC from experimental.toBC
             to baseline.frEC ...
trans ECtoSC from experimental.toSC
             to safety.frEC ...
trans SCtoBC from safety.toBC
             to baseline.frSC ...
trans SCtoEC from safety.toEC
             to experimental.frSC ...
```

Figure 6: Top Mode of IPDecisionModule

guards and actions as explained in Section 2. Guards are boolean expressions enabling transition. `BCtoSC`, `BCtoEC`, `ECtoBC`, `ECtoSC`, `SCtoBC`, and `SCtoEC` are transitions possible in the mode of IPDecisionModule.

In general, we are dealing with a nonlinear system

$$\dot{x} = f(x, u, t)$$

where $x \in \Re^n$ is the state vector, $u \in \Re^m$ is the generalized input vector. The system is subject to state constraints $g(x) < 0$ and control input limits $|u| < U_{max}$. These physical (also called *hard* in control theory) constraints are usually considered in designing a controller. In the IP example, the safety controller SC is designed using LMI (Linear Matrix Inequalities). The algorithm provides a control gain matrix K_s with the largest stability region (called here *safety region*). In contrast, the baseline controller is found using LQR (Linear Quadratic Regulator). In this case the performance is better but the stability region is smaller. It can be shown that the safety region is given by $S = \{x | x^T P_s x < 1\}$, where $P_s \in \Re^{n \times n}$ is a positive definite symmetric matrix. Since the control design is based on linear techniques, i.e., the system model is a linear approximation of the nonlinear system, the actual safety region may be smaller than S. To be specific, *safe* is set *true* as long as the system state is in the region $S_r = \{x | x^T P_s x < 0.8\}$, and *toBC* is set *true* as long as the state x satisfies $x^T P_s x < 0.5$. If the experimental controller EC drives the system outside the restricted safety region S_r then *safe* is set *false* and the SC becomes active.

4 Current and Future Work

We have shown that the hierarchical and modular specification features of CHARON can be effective in describing complex hybrid systems such as the Simplex Architecture. To make CHARON practically useful in developing reliable hybrid and embedded control systems, several researchers at the University of Pennsylvania are developing simulation supports, including hierarchical and distributed simulation, as well as model checking techniques. The emphasis is to exploit the hierarchical and modular aspects of CHARON to improve efficiency and performance. For example, new numerical techniques for event detection and localization are being developed. For more information about CHARON and related projects, please visit `http://www.cis.upenn.edu/mobies`.

Acknowledgments. We would like to thank Rajeev Alur, Radu Grosu, Vijay Kumar, Usa Sammapun, and Oleg Sokolsky for helpful discussions on the CHARON project.

References

[1] R. Alur, R. Grosu, Y. Hur, V. Kumar, and I. Lee. Modular specifications of hybrid systems in CHARON. In *Hybrid Systems: Computation and Control*, LNCS 1790, pages 6–19, 2000.

[2] C. C. Chung and J. Hauser. Nonlinear control of a swinging pendulum. *Automatica*, 31(6):851–862, 1995.

[3] D. Seto, B. H. Krogh, L. Sha, and A. Chutinan. Dynamic control system upgrade using the Simplex Architecture. *IEEE Control Systems*, 18(4):72–80, 1998.

[4] D. Seto and L. Sha. A case study on analytical redundancy of the inverted pendulum real-time control sysytem. Technical Report CMU/SEI-99-TR-023, Carnegie Mellon University, 1999.