Proc. American Control Conference, Albuquerque NM, June 1997 ©AACC

Rigorous Hybrid Systems Simulation of an Electro-mechanical Pointing System with Discrete-time Control

James H. Taylor & Dawit Kebede
Department of Electrical Engineering
University of New Brunswick
Fredericton, NB CANADA E3B 5A3
Internet: *jtaylor@unb.ca & kebede@zeus.ee.unb.ca*

Abstract

Earlier research in the modeling and simulation of hybrid systems led to the development of a general hybrid systems modeling language (HSML). Effort is underway to implement this concept in software. The standard MATLAB model framework and integration algorithms have been extended to support state-event handling in continuous-time components and to handle embedded discrete-time components.

In this paper we overview the algorithmic implementation of the HSML language constructs for dealing with state events and embedded discrete-time modules in MATLAB. An extensive example (an electro-mechanical pointing system with stiction under discrete-time control) will be presented to demonstrate the efficacy of these extensions.

1 Introduction

HSML was designed to support a broad definition of a hybrid system, which we may express informally as being an arbitrary interconnection of components that are arbitrary instances of continuous- and discrete-time systems [1, 2]. Requirements for HSML particularly focused on rigorous characterization and execution of "events", both discrete- and continuous-time, that cause discontinuous changes in system trajectories and/or the model structure itself.

One can rigorously model hybrid systems using certain other, extant languages. For example, ACSL [3] can be used to model and simulate state events in hybrid systems with considerable generality; however many other packages (especially commercially-supported ones) lack even the basic provisions for state-event handling. Also, the high-level features and strict semantics and syntax formulated for HSML facilitate and enforce a higher degree of rigor in hybrid systems modeling, thereby ensuring a greater probability of model correctness. For example, resetting the state after a state event can be done in ACSL, but not cleanly and reliably; here we use a strict protocol that permits state reset with safety. The ideas and algorithmic requirements underlying HSML can be translated into any modeling and simulation environments, assuming that a developer can gain access to the necessary internal "machinery", as demonstrated here.

This paper overviews our work to implement a subset of HSML in the MATLAB modeling and simulation environment [4]. It focuses on a detailed application involving significant state-event handling in the continuous-time part (for stiction effects) as well as dealing with "time events" associated with the execution of two discrete-time elements (a nonlinear inner-loop controller and a linear outer-loop one), thus providing a complete demonstration of the approach. This presentation complements the preliminary discussions of state-event handling in [5, 6], and provides a detailed description of the use of the later extensions for coordinating the execution of embedded discrete-time algorithms [7].

2 HSML Overview

HSML is designed to be a rigorous and modular hierarchical scheme for modeling hybrid systems. At the lowest level HSML components are "pure" continuous-time components (CTCs) and discrete-time components (DTCs) [1]. These elements are assembled into composite components, and then systems.

Here we consider CTCs that may be represented as[1]:

$$\begin{aligned} \dot{x_c} &= f_c(x_c, u_c, u_{d,k}, m, t) \\ y_c &= h_c(x_c, u_c, u_{d,k}, m, t) \end{aligned} \quad (1)$$

where x_c is the state vector, y_c is the output vector, u_c and $u_{d,k}$ are numeric input signals (continuous- and discrete-time, respectively), m is comprised of a finite alphabet of numeric or symbolic input variables that characterizes the "mode" of the model, and t is the time; in general $u_c, u_{d,k}$ and m are vectors. There are implicit "zero-order holds" operating on the elements of $u_{d,k}$ and m, i.e., these inputs remain constant between

[1]The specific class of CTC that can be modeled depends on the simulator's integration methods; MATLAB cannot handle differential algebraic equations (DAEs), so we restrict ourselves to ordinary differential equations and simplify the variable types in comparison with [1, 2].

those times when they change instantaneously. Of particular importance to the present exposition, the **mode** input m is included to provide means of controlling the model's structure and coordinating its behavior with the numerical integration process in state-event handling, as described below.

State events are characterized by *zero crossings,*

$$S(x_c, m, t) = 0 \tag{2}$$

A mode-change in the CTC can be classified as a `negative-going` event (i.e., one in which S becomes negative), an `on-constraint` event (where S remains equal to zero until another state event occurs), or a `positive-going` event. Finally, we include provision for instantaneous reset of the model state variables at a state event:

$$x_c^+ = x_c(t_e^+) = r(x_c(t_e^-), m, t_e^-) \tag{3}$$

where t_e is the event time. This feature is useful in resetting velocities after engagement to conserve momentum, for example. In accordance with this scheme for state-event definition, we permit elements of m to take on the values -1, 0, $+1$.

In the present effort, a DTC is a general algorithm which we can characterize in terms of internal variables called "discrete states" and outputs that also change discretely (instantaneously) at each execution:

$$\begin{aligned} x_{d,k}^+(t_k) &= f_d(x_{d,k}, u_{d,k}, m, t_k) \\ y_{d,k}^+(t_k) &= g_d(x_{d,k}^+, u_{d,k}, m, t_k) \end{aligned} \tag{4}$$

where $x_{d,k}$ is the discrete state vector, k is the index corresponding to the discrete time point t_k at which the state takes on the new value $x_{d,k}^+$, $u_{d,k}$ is the input vector, and $y_{d,k}^+$ is the output. Note that there are implicit "sampling" operators on $u_{d,k}$ if continuous-time variables are involved. The times t_k are usually – but not necessarily – uniformly spaced; in any case, we assume that the update times can be anticipated. Corresponding to this, we define the vector $t_{e,k}$ which at any time is comprised of the next execution times for every DTC.

3 Event Handling

3.1 State-Events

The HSML features for modeling state events are designed to permit the accurate and efficient integration of CTCs that may exhibit discontinuous behavior such as relays switching and mechanical components engaging/disengaging. The nature of the problem and an approach for proper handling of such events has been detailed previously[5, 6]; in this context, it suffices to observe that blindly integrating a CTC by stepping from a point t_k before switching to t_{k+1} after the discontinuity usually produces results that are both inaccurate and inefficient (in the sense of consuming an inordinate amount of computation).

The appropriate handling of state events requires coordination between the model and simulation package. This is achieved in HSML via `flag` variables in the model (S in Eqn. 2), and the model input variable m that can be used to control model switching. State-event handling then proceeds as follows:

1. Integrate as usual as long as the `flag` variables do not change sign. Each integration point is treated as a "trial" point until the sign condition is checked; if no sign change occurred, the point is "accepted".
2. When a sign change is detected, the trial point is discarded and an iterative procedure is initiated (within the simulator) to find the step h^* such that the `flag` variable is zero (within a small tolerance ϵ "on the other side"). The model does not switch during this part of the procedure.
3. The integrator produces an accepted point just past the switching curve (Eqn. 2) and then signals the model to switch (e.g., by changing m from 1 to -1 or *vice versa* if the boundary is to be crossed, or to 0 if the trajectory is to be confined to the boundary until the next state event).
4. The integrator then calls the model to determine if a state reset is desired, and if so executes it.
5. Normal integration proceeds from that point until the next state event is encountered.

This procedure is implemented via MATLAB extensions.

3.2 Time-Events

The approach and conventions needed to handle time events are much simpler than those required for state events, since we are merely emulating the execution of a computer algorithm in a digital setting (but without actual real time considerations). As we are implementing this feature, we assume that each DTC will "notify" the higher-level system integration block (SIB) about its next execution time, at the beginning of the simulation and at every subsequent DTC execution. The SIB determines the earliest of the anticipated time events (if there is more than one DTC), and signals the numerical integrator to stop at that time. At that point the SIB is invoked and it proceeds to execute the appropriate DTC(s), handling priority issues as required. At each DTC execution this process is updated and continued until the end of the simulation run.

3.3 Extended MATLAB Model Schema

The corresponding model input/output framework is defined as follows [5, 6]: The original MATLAB scheme was to create models in the form of functions with two inputs (t, x) and one output ($\dot{x}$). To this, we added the new input variable m (mode), allowing the numerical integration routine to request that the model switch according to state events as they are detected. In addition, two new output variables are S, the `flag` variable in Eqn. 2 that signals a state event, and r is included to permit state reset (Eqn. 3). Note that S and m may be vectors, to support multiple state event mechanisms.

The inclusion of embedded DTCs in hybrid system models necessitates a further increase in complexity in comparison with these earlier extensions. We adopted a modular model-building scheme [7], as portrayed in Fig. 1 (at end of paper). In this diagram, observe that:

- The Numerical Integrator (NI – see [5, 6]) must now serve as the "memory" for the *aggregate discrete-component states* (x_d) and for the DTC*'s times of next execution* t_e, a vector having dimension equal to the number of DTCs. The NI has the new requirement of stopping exactly at t_n, the earliest of the elements of t_e, and the "System Integrator Block" (SIB) has the responsibility of executing the correct DTC(s) when $t = t_n$.
- The continuous-time dynamics can reside in the SIB if they are simple; if it is helpful to create one or several separate CTCs, then one can do this as diagrammed, with inputs and outputs defined at the interface, as shown.

3.4 Extended MATLAB Integrators

Significant extensions must also be made in the MATLAB numerical integration algorithms. There are three features needed to permit the MATLAB integration routines to deal with state and time events:

1. the numerical integrator must coordinate with the extended model to establish the initial values of m and t_e;
2. the routine must continuously test for the occurrence of events by:
   (a) ensuring that t stops at t_n for a time event, and/or
   (b) watching for zero crossings in S, iterating exactly to the switching point and then changing m; and
3. it must execute a state reset operation after a state event, if it is called for by the model.

To support this functionality, the following conventions are imposed: The value of m for initialization is "empty" ($m = [\,]$). The model must return the appropriate value of S, based on the stipulated initial condition x_0. From this information, the integration routine will set $m = \text{sign}(S)$. During normal integration the value of m's elements will be $-1, 0, 1$. When a state event is detected and determined[2], the corresponding element of m is switched; then it is temporarily made complex and the model should respond by returning the reset value $\mathbf{r}$ (Eqn. 3) or $r = [\,]$ if no reset is to be done. Finally, that element of m is returned to $-1, 0, 1$ and numerical integration is resumed.

4 Example Application

The extensions to MATLAB outlined above were implemented and tested using a number of simple switching systems [5, 6]. Here we will focus on a more realistic (and difficult) application, control of a nonlinear model of an electro-mechanical testbed called the ATB1000[3].

[2]We determine zero crossings by embedding a modified version of MATLAB's `fzero` algorithm in the integrator [6].

The ATB1000 testbed consists of two subsystems: a drive subsystem (a DC motor with coulomb friction, a gear train with backlash, and an elastic shaft); and a wheel/barrel subsystem (including an inertial wheel, also with coulomb friction, and a flexible gun barrel). The barrel is a distributed-parameter system that can be approximated by a lumped-parameter model obtained using the finite element method. The control scheme for these dynamics is defined in two parts: a "nonlinear proportional-integral-derivative (PID)" controller for the drive system and a linear "outer loop" compensator for the flexible member; both of these are implemented in discrete-time form (as DTCs). Models for these subsystems are provided in [8].

Figure 2 depicts the results of running a 1.25-second simulation of the uncontrolled ATB1000 with a sinusoidal input, with stiction rigorously implemented. The trace of $\dot{\theta}_m$ (`dTh_m`) shows that stiction causes sticking for substantial time intervals near $t = 0.4$, 0.7 and 1.0 seconds; this is being handled without chatter, which would be observed using MATLAB's `ode45` algorithm. Similar simulation studies were performed with backlash present, and it was observed that stiction is the more dominant effect, and more troublesome to integrate because it can cause controller chattering in some circumstances. Unfortunately, page limitations do not

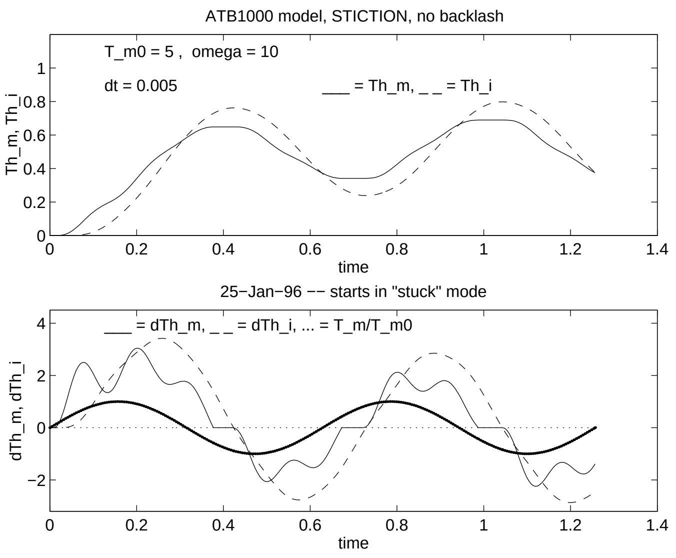

Figure 2: Illustration of State-event Handling with "Stiction"

permit further simulation results to be presented.

[3]The ATB1000 electro-mechanical pointing system was built by Integrated Systems Inc., Santa Clara, CA, for the Army Research, Development and Engineering Center (ARDEC), Picatinny Arsenal, Dover, NJ.

5 Conclusion

The MATLAB implementation presented above provides a demonstration of HSML in general and of rigorous time- and state-event handling in particular. Introducing the concept "mode" and the carefully prescribed "reset" protocol are both contributions toward making the modeling and simulation of switching in continuous-time systems more systematic. Recently added machinery for the execution of embedded discrete-time components further increases the generality available for modeling and simulation of hybrid systems. A key factor in these developments is the ability to guarantee the correct timing of state and time events, so that questions such as "does the DTC execute just before or after the parts engage" can be answered with high reliability.

Extending this modeling approach and associated numerical integration routines can be pursued in several obvious ways, e.g., they can be inserted into more sophisticated modeling environments (like the SIMULINK framework). A more important extension would involve the development of a "HSML compiler", that would take the more rigorous HSML formulations and autocode extended MATLAB models.

6 References

[1] Taylor, J. H. "Toward a Modeling Language Standard for Hybrid Dynamical Systems", *Proc. 32nd* IEEE *Conference on Decision and Control,* San Antonio, TX, December 1993.

[2] Taylor, J. H. "A Modeling Language for Hybrid Systems", *Proc.* IEEE/IFAC *Symposium on Computer-Aided Control System Design*, Tucson, AZ, March 1994.

[3] *Advanced Continuous Simulation Language* (ACSL), *Reference Manual.* Mitchell & Gauthier Associates, Concord MA 01742.

[4] MATLAB *User's Guide*, The MathWorks, Inc., Natick, MA 01760.

[5] Taylor, J. H., "Rigorous Handling of State Events in MATLAB', *Proc. IEEE Conference on Control Applications*, Albany, NY, 28–29 September 1995.

[6] Taylor, J. H. and Kebede, D., "Modeling and Simulation of Hybrid Systems", *Proc. IEEE Conference on Decision and Control*, New Orleans, LA, 13–15 December 1995.

[7] Taylor, J. H. and Kebede, D., "Modeling and Simulation of Hybrid Systems in MATLAB", *Proc. IFAC World Congress Vol. J*, San Francisco, CA, 275–280, July 1996.

[8] J. H. Taylor and J. Lu, "Robust Nonlinear Control System Synthesis Method for Electro-Mechanical Pointing Systems with Flexible Modes", *J. of Systems Engineering, Vol. 5* (special issue on motion control systems), pp. 192-204, January 1995.

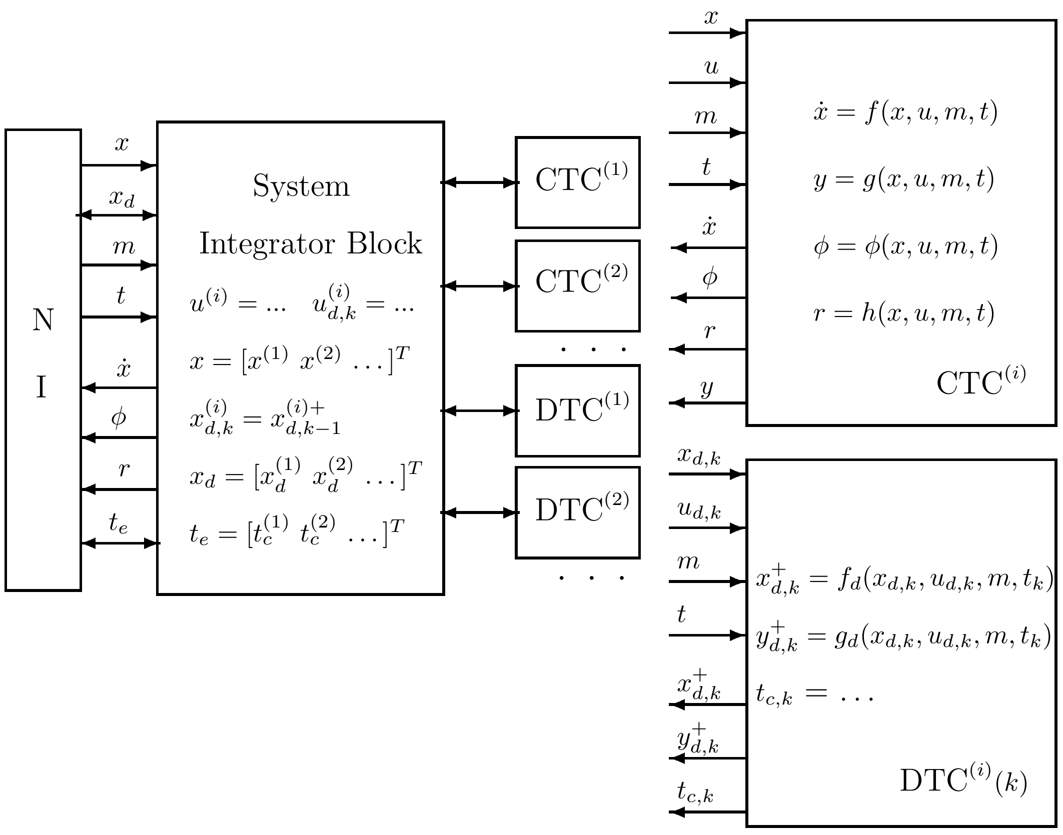
N
I
x
x_d
m
t
$\dot{x}$
ϕ
r
t_e
System
Integrator Block
$u^{(i)} = ...$ $u_{d,k}^{(i)} = ...$
$x = [x^{(1)}\ x^{(2)}\ \ldots]^T$
$x_{d,k}^{(i)} = x_{d,k-1}^{(i)+}$
$x_d = [x_d^{(1)}\ x_d^{(2)}\ \ldots]^T$
$t_e = [t_c^{(1)}\ t_c^{(2)}\ \ldots]^T$
CTC$^{(1)}$
CTC$^{(2)}$
DTC$^{(1)}$
DTC$^{(2)}$
x
u
m
t
$\dot{x}$
ϕ
r
y
$\dot{x} = f(x, u, m, t)$
$y = g(x, u, m, t)$
$\phi = \phi(x, u, m, t)$
$r = h(x, u, m, t)$
CTC$^{(i)}$
$x_{d,k}$
$u_{d,k}$
m
t
$x_{d,k}^+$
$y_{d,k}^+$
$t_{c,k}$
$x_{d,k}^+ = f_d(x_{d,k}, u_{d,k}, m, t_k)$
$y_{d,k}^+ = g_d(x_{d,k}, u_{d,k}, m, t_k)$
$t_{c,k} = \ldots$
DTC$^{(i)}(k)$

Figure 1: New MATLAB model component input/output structures