From *Proc. CCA'95* (IEEE Conference on Control Applications), Albany, NY, September 1995.

Rigorous Handling of State Events in MATLAB

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

Abstract

Previous research in the area of modeling and simulation of hybrid systems led to the development of a general hybrid systems modeling language (HSML) that has been described elsewhere. Features of HSML include: hierarchical, modular construction of models from components; consistent yet distinctive definition of continuous-time, discrete-time and logic-based components; prioritized scheduling of discrete-time components; mechanisms for state-event handling; approaches for dealing with vector-field conflicts and changing model order and structure; rigorous type and range checking; and a strict semantic basis that permits extensive checking and validation of the model.

This paper describes a first step towards algorithmic implementation of the HSML ideas and language constructs for dealing with state-event handling and vector-field conflicts in continuous-time components. Specifically, the standard MATLAB model framework and integration algorithms are extended to support these phenomena. An example is presented to show the efficacy of these extensions within the MATLAB environment.

Keywords: Modeling; simulation; numerical integration; hybrid system; dynamical system; discontinuity.

1 Introduction

The HSML language described previously [1, 2] 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-time, discrete-time and logic-based systems. 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. In this respect, there is much commonality between the HSML project and recent developments by Cellier *et al.* in the area of object-oriented modeling [3]; for a detailed view of state-event handling see especially [4, 5].

HSML is based partially on the modeling environment provided by Simnon [6]. Simnon was used as a starting point because it provides an excellent *language-based* environment for building hierarchies of interconnected components of various types with a considerable degree of encapsulation and rigor, which are features believed to be key ingredients of a solid modeling language. In addition, features of other standard packages were considered, e.g., the ACSL modeling approach [7], and the MEAD specification for component interconnection [8].

In conceiving and developing HSML, there was no claim that one cannot rigorously model hybrid systems using certain other, extant languages. For example, ACSL can be used to model and simulate hybrid systems of great generality; however many other packages lack the necessary 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. In contrast, modeling and simulating state events in other packages may require "work-arounds" and "hacks" to achieve the desired functionality (e.g., resetting states during an event). Finally, the ideas and algorithmic requirements underlying HSML can be translated into other modeling and simulation environments as well, assuming that a developer can gain access to the necessary internal "machinery" such as routines for numerical integration, as demonstrated in this presentation.

This paper describes the first steps in implementing a subset of the HSML concept in a working modeling and simulation environment, MATLAB [9]. It focuses narrowly on the issues surrounding state-event handling in continuous-time components (CTCs) and provides an illustrative example to demonstrate the efficacy of the approach. The remaining parts are as follows: Section 2 outlines the HSML structure of a CTC and its features for state-event handling, Section 3 overviews the state-event handling problem, Section 4 deals with the extensions needed in MATLAB for modeling such components, and Section 5 describes modifications required in MAT-

LAB's numerical integration routines. The final sections show the performance of the new algorithms on a simple CTC and summarize the present status and future directions of the HSML project.

2 HSML Overview

At the lowest level HSML components are "pure" CTCs, discrete-time components (DTCs) and logic-based components (LBCs) [1]. These elements are assembled into composite components, and then systems. Every component has an *interface* and a *body*; its interface defines the entities that are accessible from and to the outside. This is consistent with the MATLAB approach at the component level; above that (e.g., assembling systems from components) one would have to extend the SIMULINK formalism [10] instead.

The HSML interconnect schema is supported by the following general template for defining any type of component:

```
<Component_type> <Component_name> is
%
{ interface
  [ input(<name>,<type>,<range>); ]*
  [ output(<name>,<type>,<range>); ]*
  [ knob(<name>); ]*
  [ view(<name>); ]*
end interface; }
%
{ body
  declarations
   . . . ;
  end declarations;
  section_one
    . . . ;
  end section_one;
    . . . ;
  assignments
    [ <parameter_name> : <value>; ]*
  end assignments;
end body; }
%
end <Component_name>;
```

In this example and others in this presentation, the following BNF notation is used:

Symbolism	Meaning
< >	delimits an arbitrary syntactic entity
[ ]	delimits an optional element
{ }	delimits a compulsory element
*	repeat the marked element the appropriate number of times

In the `interface` section, note that the primary input/output variables must be typed ('signal', 'real', 'integer', 'boolean' or 'string') and may be constrained as to range, broadly interpreted to be a numerical range (`<range> = (v_min, v_max)`) or a set (e.g., `<range> =` {`"high", "medium", "low"`} for a string variable). In the `body`, the sections `declarations` and `assignments` exist for specifying internal variables and assigning parameter values, respectively. Beyond these general observations, this generic component template may be elaborated for the specific `<Component_type>`s CTC, DTC, LBC. To do so, the `body` part of the component must be comprised of distinct mandatory and optional sections; these are based on the model formulation underlying each category.

Within this framework, a CTC may be represented as[1]:

$$\begin{aligned} \dot{x_c} &= f_c(x_c, u_c, u_k, m_j, b_i, t) \\ 0 &= g_c(x_c, u_c, u_k, m_j, b_i, t) \\ y_c &= h_c(x_c, u_c, u_k, m_j, b_i, t) \end{aligned} \tag{1}$$

as outlined in [1], where x_c is the state vector, y_c is the output vector, u_c and u_k are numeric input signals (continuous- and discrete-time, respectively), m_j is comprised of symbolic input variables, b_i represents boolean inputs, and t is the time; in general u_c, u_k, m_j and b_i are vectors. There are implicit "zero-order holds" operating on the elements of u_k, m_j and b_i, i.e., these inputs remain constant between those times when they change instantaneously. Of particular importance to the present exposition, the symbolic and boolean inputs are 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.

The internal variables to be specified in the `declarations` section include `state` and `flag` variables; the first of these corresponds to x_c, and `flag` variables are used for state-event handling (see below). Accordingly, CTC components may include `initial ...end initial; dynamics ...end dynamics; constraints ...end constraints;` and `output ...end output;` sections. The first section is provided for initializing the model including its mode and state; the rest correspond directly to the sections of Eqn. (1). Of particular interest here, a CTC may include sections for rigorous handling of unpredictable state events (e.g., mechanical subsystems engaging and

[1] The specific class of CTC that can be modeled depends on the simulator's integration methods. Many cannot handle even index 1 DAEs (as in Eqn. 1), in other words, they cannot simultaneously integrate the ordinary differential equation $\dot{x_c} = f_c(\cdots)$ under the constraint $0 = g_c(\cdots)$. Integration routines have been developed for Index 1 systems; cf. [13, 14]. Higher index models present additional difficulties [15, 16].

disengaging, electronic elements switching [17]). These sections have the structure:

```
event(<signal_variable>)
  negative-going
    . . . ;
  end negative-going;
  on-event
    . . . ;
  end on-event;
  positive-going
    . . . ;
  end positive-going;
end event;
```

where `<signal_variable>` must be declared to be of type `flag` in the `declarations` section, and it characterizes the state event by a *zero-crossing condition,*

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

where S is a general expression involving the state, time and perhaps boolean variables and modes of the CTC model. The arbitrary result of the state event in the CTC model can be represented in the `negative-going`, `on-event` and `positive-going` subsections, or a simple switching variable (e.g., `sgn`) may be set therein and that variable may be used in arbitrarily complicated expressions of the form `if sgn > 0.0 then do . . . else do . . . endif;` in the `dynamics` section. Finally, support for models that undergo structural changes (e.g., changes in the definition or number of state variables) is provided by this framework [18]. In the case of mechanical subsystems engaging, the number of states decreases, producing a "higher-index" model that can be characterized by conditional constraint equations and may be reduced using the Pantelides algorithm [11] to automatically reduce it to state-space form, for example.

3 State-Event Handling

The HSML features for modeling state events are designed to prevent problems that interfere with accurately integrating CTCs that may exhibit discontinuous behavior such as relays switching and mechanical components engaging/disengaging. The difficulty is this: If the simulator blindly integrates a CTC with state events, then the switching point is typically "over-shot", i.e., the numerical integration routine steps from a point t_k before switching to t_{k+1} after the discontinuity, trusting its automatic step-size-control algorithm to make the transition by detecting a large error, decreasing the step size until error is acceptable, and continuing on to t_{k+2} etc. This creates two problems [4, 17, 18]:

1. Most high-order integration routines use a number of derivative evaluations to calculate $x(t_{k+1})$ given $x(t_k)$. In the case of the standard fourth-order Runge-Kutta method, these evaluations are at times t_k, $t_{k+\frac{1}{2}}$ and t_{k+1}; for predictor/corrector methods these are at times t_k, t_{k-1}, t_{k-2} ...and t_{k+1}. In passing the discontinuity, these derivative evaluations are made indiscriminately on both sides of the switching point, thus producing a "garbage" point (so called because its error is completely unpredictable, due to this erratic switching behavior). The fact is: the point $x(t_{k+1})$ obtained in this fashion is **not** on the same trajectory as $x(t_k)$, due to this anomolous handling of the discontinuity.
2. Using an integration routine without provision for catching and handling state events correctly produces inefficient simulations as well as inaccurate ones. The continual process of decreasing and increasing the step size leads to "creeping" simulation that may take large amounts of computation and cause excessively long simulation runs.

Both of these problems are evident in the example treated in Section 6.

The correct handling of state events is this:

1. The model should not be allowed to switch during a numerical integration step.
2. The integration routine should not integrate past the switching point.

This requires coordination between the model and simulation package, that is achieved in HSML via `flag` variables in the model (these signal the integrator that a state event has been overshot), and the model input variable `mode` that can be used to control model switching. The state-event-handling process is then as follows:

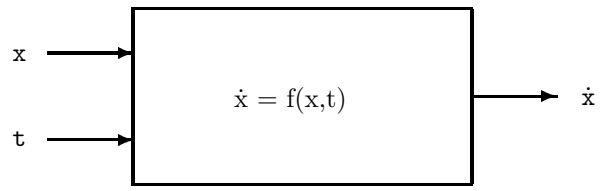
1. Integrate as usual as long as the `flag` variable does not change sign. Each integration point is treated as a "trial" point until the sign condition is checked; if no sign change has occurred, the point becomes "accepted".
2. When a sign change is detected, the trial point is discarded and an iterative procedure is initiated (within the simulator) to find the step h^* such that the `flag` variable is zero (within a tolerance ϵ). The model still does not switch during this procedure (so no "garbage" points are produced).
3. The integrator produces an accepted point on the switching curve (Eqn. 2) and then signals the model to switch (e.g., by changing `mode` from 1 to – 1 or *vice versa*).
4. Normal integration proceeds from that point.

This procedure is illustrated below using MATLAB extensions.

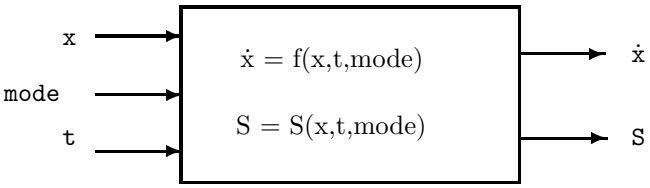
4 Extended Model Schema

The above outline of HSML representations of CTCs and their provision for characterizing state-events provides

a clear roadmap for implementation in other packages. One significant extension needed in MATLAB for modeling and simulating state events in CTCs is in the input/output structure of the model. The existing and extended schema are depicted in Fig. 1: The additional



(a) Standard MATLAB model schema



(b) Extended MATLAB model schema

Figure 1: MATLAB model input/output structure

output is the `flag` variable S (Eqn. 2) that signals a state event (`S` will change sign at such an occurrence); the new input `mode` allows the numerical integration routine to request that the model switch according to the state event just detected. Note that `S` and `mode` may be vectors, to support multiple state events.

5 Extended Integration Schema

A second significant extension must be made in the MATLAB numerical integration algorithms: neither those in MATLAB, i.e., `ode23` and `ode45`, nor those in SIMULINK, i.e., `gear`, `rk23` and `rk45`, can handle state events in the desired fashion. There are two features needed to permit the MATLAB integration routines to deal with state events:

1. the numerical integration must coordinate with the extended model to establish the initial value of `mode`, and
2. the routine must continuously test for the occurrence of the event by watching for zero crossings in the `flag` variable(s).

The following MATLAB function provides an example of this functionality:

```
function [tout,yout] = trap_seh(dyfun,...
   t0,tf,y0,step,tol,trace)
%TRAP_SEH       Solve differential equations,
%  "trapezoidal" method.  TRAP_SEH integrates
%  a system of ordinary differential equations
%  using the "trapezoidal" algorithm, with a
%  hybrid interpolation scheme to catch state
%  events (times where ydot is discontinuous).
%  (**remaining comment lines omitted.**)

%  J.H. Taylor - UNB, Fredericton, NB CANADA
%% Initialization
   if nargin < 5, step = 1.e-2; end
   if nargin < 6, tol = eps; end
   if nargin < 7, trace = 0; end
   t = t0; h = step; y = y0(:);
   chunk = 256;
   tout = zeros(chunk,1);
   yout = zeros(chunk,length(y));
   k = 1; tout(k) = t; yout(k,:) = y.';
   kkmax = 3;
   if trace, clc, t, h, y, end
%% get phi for initial mode evaluation:
   mode = 0;
   [junk,phi] = feval(dyfun,t,y,mode);
   mode = sign(phi);
%% Main integration loop
while (t < tf) & (t + h > t)
   if t + h > tf, h = tf - t; end
   % save last "accepted" point:
   yold = y(:); told = t; phiold = phi;
   % trapezoidal predictor/corr. trial step:
      [ydot,junk] = feval(dyfun,t,y,mode);
         ydot = ydot(:);
      yp = y + h*ydot; t = t + h;
      [ydotp,junk] = feval(dyfun,t,yp,mode);
         ydotp = ydotp(:);
      y = yold + h*(ydot + ydotp)/2;
   % Get the trial point phi; test:
   [junk,phi] = feval(dyfun,t,y,mode);
   if sign(phi) == -sign(phiold)
      % state event is detected:
      hstar = h; kk = 0;
      while abs(phi) > eps
         %% hybrid zero-solving routine
         %% captures crossing within eps;
         %% omitted for sake of space
         %% "hstar" is the corresponding step:
         t = told + hstar;
         yp = yold + hstar*ydot;
         [ydotp,phi] = feval(dyfun,t,yp,mode);
            ydotp = ydotp(:);
         y = yold + hstar*(ydot + ydotp)/2;
         [junk,phi] = feval(dyfun,t,y,mode);
      end
      mode = - mode;
      phi = - sign(phiold)*abs(phi);
   end
   k = k+1;
```

```
   if k > length(tout)
      tout = [tout; zeros(chunk,1)];
      yout = [yout; zeros(chunk,length(y))];
   end
   tout(k) = t; yout(k,:) = y.';
   if trace, home, t, h, y, end
end
if (t < tf), disp('Singularity likely.'), t
end
tout = tout(1:k); yout = yout(1:k,:);
```

6 Example Application

The extensions to MATLAB outlined above were implemented and tested using the following simple switching system:

$$\begin{aligned} \dot{x_1} &= x_2 \\ \dot{x_2} &= -\text{sign}(x_1) \end{aligned} \tag{3}$$

The extended MATLAB model for this system (Fig. 1) is:

```
function [xdot,phi] = relay(t,x,mode)
if mode == 0
   if x(1) == 0, phi = x(2) % x_2 governs mode
   else phi = x(1);         % ... if x_1 = 0
   end
end
xdot(1) = x(2);
xdot(2) = -mode;
phi = x(1);
```

The new features of this model compared with a standard MATLAB model are: a section to be called with `mode = 0` to aid in initializing `mode` correctly (based on `phi`), and the additional input variable `mode` and output variable `phi` to provide the coordination for state-event handling described previously. In addition to the extended model defined in Section 4, a standard MATLAB model and a standard SIMULINK model (based on the `sfunc` formalism) were prepared, to permit comparisons with respect to accuracy and efficiency of simulation.

Figure 2 depicts the results of running a 10-second simulation with initial condition $x0 =$ `[ 0.25 ; 0 ]` using the methods ode45 (MATLAB), rk45 (SIMULINK) and trap_seh, the trapezoidal integration routine with state-event handling reproduced Section 5. In the first two cases (built-in variable step-size routines) the step size was unspecified; for trap_seh (a fixed-step routine) the step was set to 0.05 sec. The trap_seh routine produced the most precise solution, a smooth, closed trajectory represented by the solid curve in the plot. The most erratic results were produced by the rk45 algorithm; as shown, the switching is missed quite significantly, leading to an initial increase in the amplitude of the response followed by a decay that (for longer simulation times) continues indefinitely. MATLAB's ode45 method (dashed curve) appears to maintain a steady amplitude that exhibits a "cutting the corners" effect due to using a much larger step size than trap_seh.

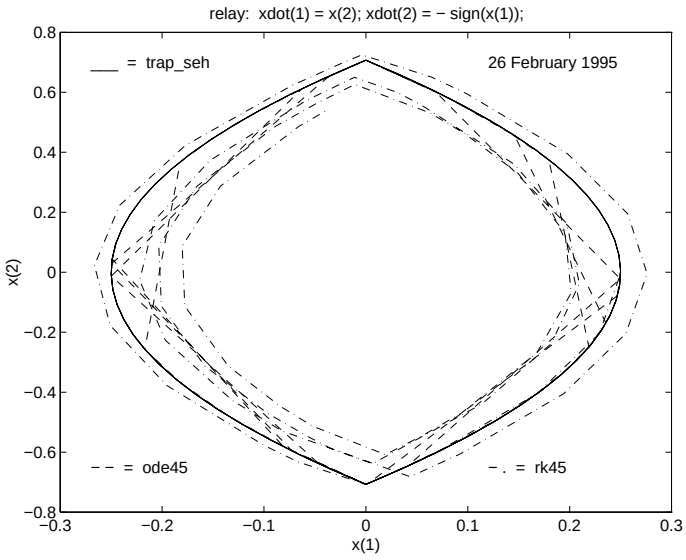


Figure 2: Numerical Integration Method Comparison

A secondary benefit in using a fixed-step routine with state-event handling is run-time efficiency. Longer simulations were conducted with the three methods above (simulation final time = 100 sec); for both ode45 and rk45 these simulations took about 60 seconds of wrist-watch time, while trap_seh took less than half as much time. More precise computations are difficult, since built-in MATLAB routines do not register the number of `flops` consumed by a simulation. However, the m-file for ode45 was copied into our workspace and renamed so it could be compared with trap_seh; since both are thus interpreted rather than compiled code, this is the most meaningful comparison. The results were: 32 sec of wrist-watch time and 36,949 flops for trap_seh, 140 sec of wrist-watch time and 1,220,204 flops for (interpreted) ode45. While some of the added overhead of ode45 may be attributed to its being of higher order and including an automatic step-size control algorithm (but this is supposed to be an advantage!), some of the big difference is due to the "creeping" effect [4] caused by state-event discontinuities making the step-size reduce greatly until the discontinuity is past.

7 Conclusion

The implementation presented above provides a simple but compelling demonstration of HSML in general and of the the concept of state-event handling and its value

for modeling and simulating switching systems in particular. The machinery for extending the model and numerical integration routines is being generalized in several ways, i.e., it has been inserted into more sophisticated models and integration routines (like ode45), and it now supports vector modes and switching functions for defining multiple state events in one model. In addition, provision for state resetting at state events has been implemented. These extensions will be presented in [19].

Acknowledgement: The author would like to thank his EE 6383 Nonlinear Controls Systems students, especially Mssrs. Roger Cormier and Dawit Kebede, for discussions related to this publication.

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] Elmqvist, H., Cellier, F. E. and Otter, M., "Object-Oriented Modeling of Power-Electronic Circuits Using Dymola", *Proc. CISS'94* (First Joint Conference of International Simulation Societies), Zurich, Switzerland, August 1994.

[4] Cellier, F. E., Elmqvist, H., Otter, M. and Taylor, J. H., "Guidelines for Modeling and Simulation of Hybrid Systems", *Proc.* IFAC *World Congress*, Sydney Australia, 18–23 July 1993.

[5] Cellier, F. E., Otter, M. and Elmqvist, H., "Bond Graph Modeling of Variable Structure Systems", *Proc. ICBGM'95* (Second International Conference on Bond Graph Modeling and Simulation), Las Vegas, Nevada, January 1995.

[6] Elmqvist, H., "SIMNON - An Interactive Simulation Program for Non-Linear Systems", in *Proc. of Simulation '77*, Montreux, France, 1977.

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

[8] Taylor, J. H., Frederick, D. K. Rimvall, C. M. and Sutherland, H. A., "The GE MEAD Computer-Aided Control Engineering Environment", *Proc.* IEEE *Symposium on CACSD,* Tampa, FL, December 16, 1989.

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

[10] SIMULINK *User's Guide*, The MathWorks, Inc., Natick, MA 01760.

[11] Pantelides, C.C., "The Consistent Initialization of Differential-Algebraic Systems," *SIAM Journal of Scientific and Statistical Computation*, **9**, No. 2, pp. 213–231, 1988.

[12] Gear, C. W., *Numerical Initial Value Problems in Ordinary Differential Equations*, Prentice-Hall, 1971.

[13] Brenan, K. E., Campbell, S. L. and Petzold, L. R., *Numerical Solution of Initial Value Problems in Differential-Algebraic Equations*, North Holland, 1989.

[14] Andersson, M., Brück, D., Mattsson, S. E. and Schönthal, T., "OmSim – An Integrated Interactive Environment for Object-Oriented Modeling and Simulation", *Proc.* IEEE/IFAC *Symposium on Computer-Aided Control System Design*, Tucson, AZ, March 1994.

[15] Mattsson, S. E. and Söderlind, G., "A New Technique for Solving High-Index Differential-Algebraic Equations Using Dummy Derivatives", *Proc. CACSD'92, IEEE Computer-Aided Control Systems Design Conference,* Napa, CA, pp. 218–224, March 17–19, 1992.

[16] Campbell, S. L., "High Index Differential Algebraic Equations", preprint/report, Dept. of Math, North Carolina State University, Raleigh NC 27695-8205, June 1993; *slc@math.ncsu.edu.*

[17] Cellier, Francois, "Combined Continuous/Discrete System Simulation by Use of Digital Computers: Techniques and Tools", PhD Thesis, Swiss Federal Institute of Technology, Zurich, Switzerland, 1979, Number ETH 6438.

[18] Taylor, J. H. *A Rigorous Modeling and Simulation Package for Hybrid Systems*, US National Science Foundation SBIR Report, Award No. III-9361232, Odyssey Research Associates, Inc., June 1994 (available only from the author).

[19] J. H. Taylor, "Modeling and Simulation of Hybrid Systems", accepted for IEEE Conference on Decision and Control, New Orleans, LA, December 1995.