13th Triennial World Congress, San Francisco, USA

MODELING AND SIMULATION OF HYBRID SYSTEMS IN MATLAB

James H. Taylor & Dawit Kebede

Department of Electrical Engineering
University of New Brunswick
PO Box 4400, Fredericton, NB CANADA E3B 5A3
email: jtaylor@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) that has been described elsewhere. Effort is underway to implement this concept in software. First, the standard MATLAB model framework and integration algorithms were extended to support state-event handling in continuous-time components, including approaches for dealing with vector-field conflicts and changing model order and structure. More recently, further extensions have been made to handle embedded discrete-time components.

In this paper we describe the algorithmic implementation of the HSML ideas and language constructs for dealing with state events and embedded discrete-time modules in MATLAB. An example will be presented to show the efficacy of these extensions.

Keywords: Hybrid systems; modeling; simulation; numerical solutions; discontinuities; embedded discrete-time systems.

1 INTRODUCTION

HSML, as described previously [1, 2, 12], 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. 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 events see especially [4, 5].

In conceiving and developing HSML, there was no claim that one cannot rigorously model hybrid systems using certain other, extant languages. For example, ACSL [6] 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 safely. 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 in this presentation.

This paper describes our work to implement a subset of HSML in a working modeling and simulation environment, MATLAB [7]. It focuses on state-event handling in continuous-time components (CTCs), dealing with "time events" associated with the execution of discrete-time components (DTCs), and provides an illustrative example to demonstrate the approach. The primary novel feature over the first step [8, 9] is machinery for executing embedded discrete-time algorithms.

The remaining parts are as follows: Section 2 outlines the HSML framework for hybrid systems modeling, Sections 3 and 4 overview state- and time-event handling, respectively, Section 5 deals with the extensions needed in MATLAB for modeling hybrid systems, and Section 6 describes modifications required in MATLAB's numerical integration routines. The final sections show the use of the new approach on a simple switching hybrid system with discrete-time "filters" and summarize the present status and future directions of the HSML project.

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. Every component has an *interface* and a *body*; its interface defines the entities that are accessible from and to the outside, and the body describes the dynamic behavior of the component. As described in [1, 2], the `interface` section requires 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, a component is elaborated for the specific component type (CTC, DTC) by adding distinctive mandatory and optional sections; e.g., `initial`, `dynamics` and `output` for a CTC.

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

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

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 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*,

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

where S is a general expression involving the state, time and perhaps the mode of the CTC. An arbitrary state 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. Note that this framework provides support for models that undergo structural changes (e.g., changes in the definition or number of state variables) [12]; e.g., in the case of mechanical subsystems engaging, the number of states decreases. Finally, we include provision for instantaneous reset of the model state variables at an event, according to

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

where r is also an arbitrary expression and t_e is the event time. This feature is useful in resetting velocities after engagement to conserve momentum, for example.

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 ($t_k = t_0 + k * T_s$ where T_s is the "sampling period"); in any case, we assume that the update times can be anticipated. (In future extensions provision will be made for computational delay δ_k between the sample time and the output change; this may be modeled with varying degrees of realism, from a fixed delay time to an actual emulation of the computational burden required in handling the computations. Also, note that while the above appears in the form of a discrete-time controller or Kalman filter algorithm, we ultimately have other more general "logic-based" component formulations in mind [1, 2, 12].)

[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].

3 STATE-EVENT HANDLING

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 difficulty is this: If the simulation environment 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 produces an invalid point $x(t_{k+1})$ that is not on the same trajectory as $x(t_k)$. How accurate this point is depends on the quality of the variable step-size algorithm; an earlier example [8] shows that MATLAB's ode45 is quite good but SIMULINK's rk45 is not.

Also, 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 excessively long simulation runs. Finally, a third problem also arises in state-event handling: the possibility that the states might have to be reset at the event. Most simulation software does not permit this, or permits it only as a "hack". Correct treatment of these issues is illustrated in the examples in [8, 9] and in Section 7.

The appropriate 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.
3. State reset should only be permitted as a part of state-event handling.

This requires coordination between the model and simulation package, that is achieved in HSML via `flag` variables S in the model (these signal the integrator that a state event has been overshot), 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` 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 small tolerance ϵ "on the other side"). The model still does not switch during this procedure (so no invalid points are produced).
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 for some time interval).
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 illustrated below using MATLAB extensions.

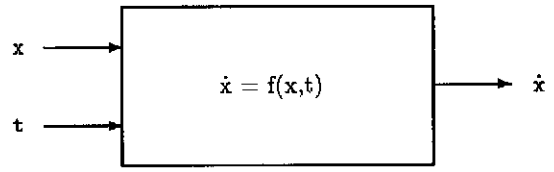
4 TIME-EVENT HANDLING

The approach and conventions needed to handle time events are much simpler than those required for state events. After all, we are merely emulating the execution of a computer algorithm in a digital setting (but without real time considerations). As we are implementing this feature, we assume that each DTC will "notify" the higher-level system integration block (SIB) about the next execution time, `t_exec` ($t_{e,k}$). This is done 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 with $t = t_{e,k}$ and it thus executes 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.

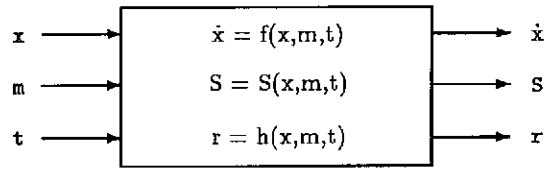
5 EXTENDED MODEL SCHEMA

The above outline of HSML's approach for characterizing state and time events provides a clear roadmap for implementation in a software package. So far, we have focused on MATLAB for this purpose, since it is so readily extensible.

Significant extensions are needed in MATLAB for modeling and simulating state and time events. For state-event handling in CTCs, we modified the input/output structure of the model substantially, as depicted in Fig. 1: The new input variable m (mode) allows the numerical integration routine to request that the model switch according to the state event just detected. One additional output variable S is the `flag` variable (S in Eqn. 2) that signals a state event (S will change sign or be set to 0 at such an occurrence), and another output variable `r` is included to permit state reset (Eqn. 3). Note that S and m may be vectors, to support multiple state event mechanisms (switching conditions).



(a) Standard MATLAB model schema



(b) Extended MATLAB model schema

Figure 1: MATLAB model input/output structures

The inclusion of embedded DTCs in hybrid system models necessitates a further increase in complexity in comparison with Fig. 1 (b). We have recently adopted a modular model-building scheme much closer to the conceptual design of HSML (and reminiscent of SIMNON [10]). This scheme is portrayed in Fig. 2 (at end of paper). In this diagram, observe that:

- The Numerical Integrator (NI – see [8, 9]) 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 $t = t_n$.
- If multiple DTCs are to be executed at t_n, then the SIB has the responsibility of calling them in the correct order.
- The continuous-time dynamics can reside in the SIB if they are simple (see example); if it is helpful to make several CTCs, then one can do this as diagrammed; note that this necessitates CTCs having inputs and outputs that are defined at the interface, as shown.

6 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 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 with the indicated mode change.

7 EXAMPLE APPLICATION

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

```
function [xdot,S,r,xd,t_exec] = ...
                    twin_ball(t,x,m,xd,t_exec)
% mode = [] at init. -> S(0), t_exec(0)
if mode == [],
  for i=1:2,
    if x(2*i-1) == 0, S(i) = x(2*i);
    else S(i) = x(2*i-1);
    end
  end
  [junk,t_c1] = dfilt1(t,0,0,mode);
  [junk,t_c2] = dfilt2(t,0,0,mode);
  t_exec = [ t_c1 ; t_c2 ];
  return
end % of initialization
icall = max(abs(imag(mode)));
if icall == 0, % real => DTC, x_dot & S eval.
  if t = t_exec(1),
    x_d1 = xd(1); u_d1 = x(1);
    [x_d1,t_c1] = dfilt1(t,x_d1,u_d1,mode);
    xd(1) = x_d1; t_exec(1) = t_c1;
  end % of DTC # 1 processing
  if t = t_exec(2),
    x_d2 = xd(2); u_d2 = x(3);
    [x_d2,t_c2] = dfilt2(t,x_d2,u_d2,mode);
    xd(2) = x_d2; t_exec(2) = t_c2;
  end % of DTC # 2 processing
  xdot(1) = x(2); xdot(2) = -mode(1);
  xdot(3) = x(4); xdot(4) = -mode(2);
```

[2] We determine zero crossings by embedding a modified version of MATLAB's fzero algorithm in the integrator [9].

```
  S(1) = x(1); S(2) = x(3);
else % complex => x or S reset
  reset = x;
  for i=1:2,  % reset x at mode change:
    if abs(imag(mode(i))) == 1,
      reset(2*i-1) = 0;
      reset(2*i) = 0.8*x(2*i);
    end
  end
end
```

This model represents two independent switching systems, with dynamics $\ddot{x} = \pm 1$ in both cases and with an 80 % loss of "velocity" at each switching [9] with two digital filters added to track states 1 and 3. While this model is simplistic, it is a challenging test because the state events occur more and more frequently and can be made simultaneous. The new features of this model compared with a standard MATLAB model are: a section to be called with m= [ ] to aid in initializing m correctly (based on S), the additional input variable m and output variable S to provide the coordination for state-event handling described previously, the second new output `reset`, to contain the state reset (if desired), plus the inputs and outputs to accommodate the two DTCs.

In previous work [8] a standard MATLAB model and a standard SIMULINK model (based on the `sfunc` formalism [11]) were prepared for the illustrative example, to permit comparisons against the extended first-phase integration approach with respect to accuracy and efficiency of simulation. The generated results showed that ode45 could catch a simple relay-switching state event quite accurately (due to its superior error/step-size-adjustment algorithm), while the SIMULINK model integrated using rk45 provided a poor result (erratically missing the state event by a substantial amount). Using the uncompiled (m-file) version of ode45, it took about 10 to 15 times longer to run that example compared to our approach and routine. For the extensions presented here, we cannot make such comparisons, since neither approach can handle state events with reset.

Figure 3 depicts the results of running a 6.4-second simulation with initial condition `x0 = [ 0.25 ; 0 ; -0.25 ; 0 ]` using our latest integration routine with state-event handling, reset and DTC execution protocols. There are no comparative results, since the model cannot be handled by other MATLAB or SIMULINK routines.

8 CONCLUSION

The MATLAB implementation presented above provides a demonstration of HSML in general and of the importance of careful 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

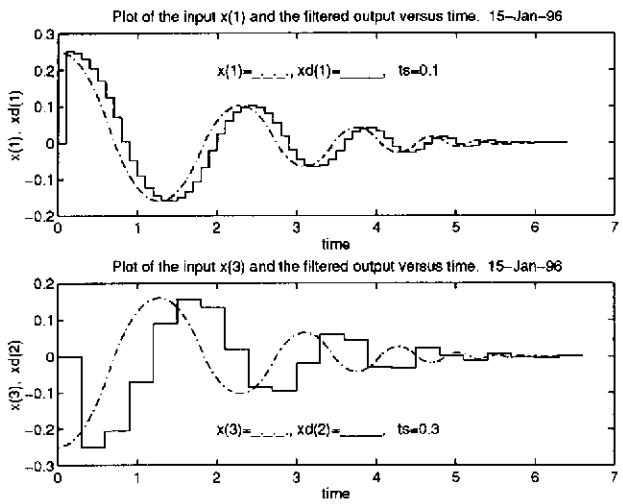


Figure 3: Illustration of State-event Handling, Reset and DTC Execution

in continuous-time systems more systematic and rigorous. Recently added machinery for the execution of embedded discrete-time components further increases the level of generality in modeling and simulation of hybrid systems.

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. (See Section 2, e.g. typing and constraining input/output variables.) In a more technical vein, handling vector-field conflicts (situations where the solution must remain on the surface $S = 0$ because the ODE vector is "into" the surface on both sides) is not completely treated in the above approach. In short, much remains to be done!

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

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

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

[9] 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.

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

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

[12] 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).

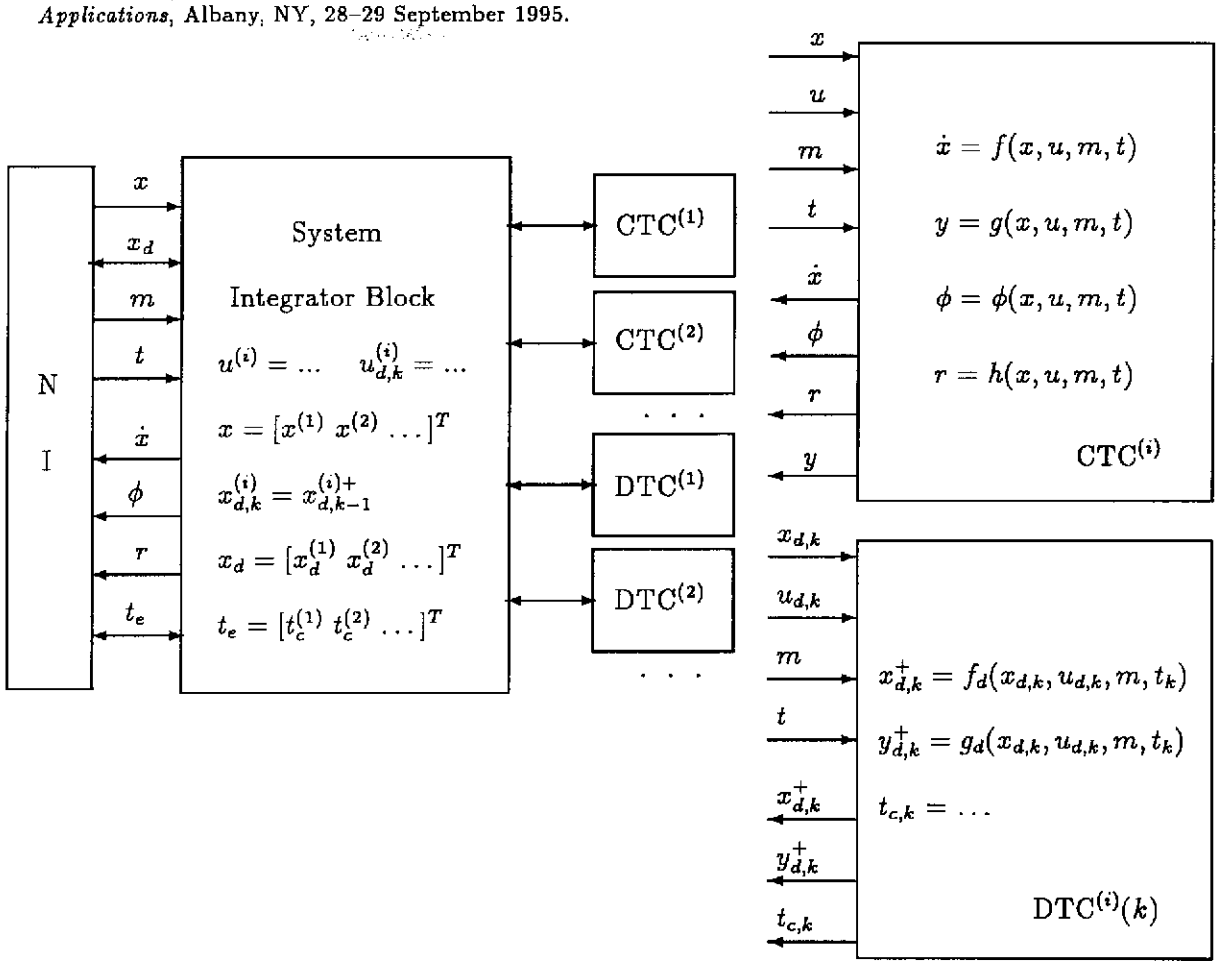


Figure 2: New MATLAB model component input/output structures