From *Proc. CACSD'94* (IEEE/IFAC Symp. on Computer-Aided Control System Design), Tucson, AZ, March 1994. ©IEEE

# A Modeling Language for Hybrid Systems

James H. Taylor
Odyssey Research Associates (ORA)
301 Dates Drive, Ithaca, NY 14850
*jim@oracorp.com*

## Abstract[1]

The general hybrid systems modeling language (HSML) described here will serve two purposes: to define formally what is meant by the term "hybrid system", and to provide the basis for language-based "front ends" for hybrid system simulation environments. Features of HSML include: hierarchical, modular construction of models; 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 order and structure; rigorous type and range checking; and a strict semantic basis that permits extensive checking and validation of the model.

**Keywords:** Modeling languages; hybrid systems; dynamical systems; integration methods; discontinuity handling; modeling and simulation.

## 1 Introduction

The language described below is being designed to support a broad definition of a hybrid system (within the context of intelligent distributed control), 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. HSML is based partially on the modeling environment provided by Simnon [1]. The reason Simnon is used as a starting point is that 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 as envisioned here.

In addition to these underpinnings, we have considered features of the earlier standard CSSL [2], the ACSL modeling approach [3], and the MEAD paradigm for component interconnection [4]. Note that a completely *graphical* model-building environment isn't considered – in part, this is because we believe that the definition of a language standard should precede the definition of a graphical standard; in part this is due to a prejudice that at the lowest component level a language-based approach is more rigorous, flexible and natural. Thus a graphical notation for connecting components is presented, and a supporting graphical editor for defining hierarchical models of hybrid systems would be viewed as an important asset (refer to [5] for a concept close to this philosophy).

In developing and evaluating HSML, it is important to recognize that there is no claim that one cannot rigorously model hybrid systems using other, extant languages. For example, ACSL [3] can be used to model hybrid systems of great generality. However, the high-level features and strict semantics and syntax being formulated for HSML will facilitate and enforce a higher degree of rigor in hybrid systems modeling, thereby ensuring a greater probability of model correctness. In this context, note that HSML is being created to be as simple as possible within the constraints imposed by the perceived need for rigor and functionality.

This paper is a companion to [6], which described the three types of components (CTC = continuous-time component, DTC = discrete-time component, and LBC = logic-based component) and provided two illustrative examples of HSML - a CTC for a simple electromechanical component with a state event handler, and a composite component (CC) comprised of the electromechanical subsystem and digital sensor and controller modules. Here we will concentrate on the connection of components and on modeling language features that support a broad spectrum of hybrid systems and features. It should be stressed that this language is still under development, and details will be subject to change as they are finalized. Syntax, in particular, is still an open question.

The remaining sections of this paper are as follows: Parts 2 and 3 deal with the connection and modeling of components, and Part 4 describes how to build a system

[1]Support for this work has been provided by the Advanced Research Project Agency through the U.S. Army AMCCOM at Picatinny Arsenal, NJ by Contract Number DAAA21-92-C-0013.

model. The final section outlines additional considerations needed to complete the definition of HSML.

## 2 Component Interfaces and Connections

As mentioned in [6], at the lowest level HSML components are "pure" CTCs, DTCs and LBCs. These are assembled into composite components (CCs), and then systems. Every component has an *interface* and a *body*; its interface defines the entities that are accessible from and to the outside, as follows:

1. *Primary input/output variables:* `input` and `output` signals may be connected to other components' outputs and inputs, respectively.

2. *Secondary input/output variables:* `scope` and `knob` entities can *not* be connected to other components' inputs and outputs; rather `scope` variables may be stored and displayed, and `knob` parameters (constants) may be set/changed arbitrarily using a supporting simulation environment during the definition of simulation experiments.

At the interface, every component may have an arbitrary number of inputs and outputs (primary and secondary), of the following types: continuous-time signal, real number, integer, boolean variable, and character-string (message). Correspondingly, there are five types of "pure" connections in this formalism, as illustrated in Fig. 1 and denoted by:

1. Continuous-time signal: $\triangle\!\longrightarrow\!\triangle$ Example: input `disturbance` connected to turret input `load_dist`.

2. Real number, transmitted at discrete time(s): $\bullet\!\!-\!\!\bullet$ Example: track_mgr output `theta_com` connected to DPID input `ref`.

3. Integer, transmitted at discrete time(s): $\square\!\longrightarrow\!\square$ Example: engagement_mgr output `which_one` connected to track_mgr input `threat_num`.

4. Boolean variable, transmitted at discrete time(s): $\oplus\!\longrightarrow\!\oplus$ Example: input `engage` connected to engagement_mgr input `hit_it`.

5. Character-string (message), transmitted at discrete time(s): $\diamond\!\!-\!\!\diamond$ Example: engagement_mgr output `stat` connected to system output `track_status`.

In addition, there are two allowed types of "mixed" connections, denoted by:

1. Continuous-time signal to real number, transmitted at discrete sampling time(s): $\triangle\!\!-\!\!\bullet$ Example: turret output `theta` connected to tracker input `theta`.

2. Real number (transmitted at discrete time(s) and held until the next sample) to continuous-time signal: $\bullet\!\longrightarrow\!\triangle$ Example: DPID output `command` connected to turret input `volts`.

These mixed connections are permitted to eliminate the need for explicit modeling of analog-to-digital and digital-to-analog converters; if there are reasons to model these transformations more rigorously, then one may define a component for each such conversion. Most simulation environments take care of these conversions automatically; we assume this to be so in defining HSML.

## 3 HSML Component Model Structure

The interconnect formalism outlined above 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 will be used:

```
Symbol Meaning

 < >   delimits an arbitrary syntactic entity
```

| | |
|---|---|
| `[ ]` | `delimits an optional element` |
| `{ }` | `delimits a compulsory element` |
| `*` | `repeat the marked element the appropriate number of times` |
| `%` | `the text that follows is a comment` |

The sections `declarations` and `assignments` exist for specifying internal variables and assigning parameter values, respectively. 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).

The component template defined above may be elaborated for the specific `<Component_type>`s CTC, DTC, LBC and CC. 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:

*Continuous-time Components* (CTCs):

- A CTC may be described by an arbitrary ordinary differential / algebraic equation set:[2]

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

  As outlined in [6], $x_c$ 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.

- The internal variables to be specified in the `declarations` section include `state, local` and `flag` variables; the first corresponds to $x_c$, the second to internal variables to be (for example) constrained in range, and `flag` variables are used for state-event modeling (below).

- Correspondingly, 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 state; the rest correspond directly to the sections of Eqn. (1).

- a CTC may include sections for rigorous handling of unpredictable state events (e.g., mechanical subsystems engaging and disengaging [10]). These sections have the structure:

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

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

$$S(x_c, b_i, m_j) = 0 \qquad (2)$$

  where $S$ is a general expression involving the state 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-to-positive` and `positive-to-negative` 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.

- Support for models that undergo structural changes (e.g., changes in the definition or number of state variables) will be provided. 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.

Note that none of the above itemized sections are *mandatory;* e.g., a CTC need not include constraint equations, thereby eliminating the need for a `constraints ...end constraints;` section. A significant example CTC model in included in [6].

*Discrete-time Components* (DTCs):

- A DTC may be described by an arbitrary difference equation set:

[2]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. [7]. Higher index models present additional difficulties [8, 9].

$$\begin{aligned} x_{k+1}(t_k) &= f_k(x_k, u_c, u_k, m_j, b_i, k) \\ 0 &= g_k(x_{k+1}, u_c, u_k, m_j, b_i, k) \quad (3) \\ y_{k+1}(t_k + \delta_k) &= h_k(x_{k+1}, u_c, u_k, m_j, b_i, k) \end{aligned}$$

where $x_k$ is the discrete state vector, $k$ is the index corresponding to the discrete time point $t_k$, $y_{k+1}$ is the output vector, and $u_c, u_k, m_j, b_i$ are as above. There are implicit "sampling" operators acting on $u_c$, i.e., the input value $u_c(t_k)$ is used in updating $x_k$. The times $t_k$ are usually – but not necessarily – uniformly spaced ($t_k = k * T_s$ where $T_s$ is the "sampling time"); in any case we assume that the update times can be anticipated and included in the component model. Note that there may be computational delays $\delta_k$ in the output equation, modeled with an arbitrary degree of realism.

- The internal variables to be specified in the `declarations` section include several that are common to CTCs, i.e., `state` and `local` variables; as before, the first corresponds to $x_k$, and the second to internal variables to be (for example) constrained in range. Additional variables to be declared are: `tsample` (the internal variable defining the time of next update), `tdelay` corresponding to $\delta_k$ in Eqn. (3), and `priority` which establishes the precedence order of execution of DTC and/or LBC modules that may always or sometimes have coincident execution times.
- Corresponding to the parts in Eqn. (3), DTC components may include `initial ...end initial; update ...end update;` and `constraints ...end constraints;` sections.
- Note that DTCs are easier to emulate in a digital simulation environment; therefore, we do not anticipate that special features for state-event handling will be required.
- Support for models that undergo structural changes (e.g., changes in the definition or number of state variables) will be provided.

This component type represents a particular digital module class that is reserved for pure numerical computations. The advantages of this particular taxonomy are that (i) the detailed structure of Eqn. (3) can be fully supported, and (ii) such components can meaningfully be linearized and analyzed while, in general, logic-based components (below) cannot.

*Logic-based Components* (LBCs):

- Each LBC may have numeric and/or symbolic inputs, symbolic outputs, and symbolic internal variables called "modes". At this point, it is not clear that these components have a "generic form" in mathematical terms as above except in terms of the categorization of input and output variables. Thus we *formally* write

$$m_{j+1} = \Phi_j(m_j, u_c, u_k, j) \qquad (4)$$

where $m_j$ is the mode vector, $j$ is the index corresponding to the discrete event triggering the LBC action, $\Phi_j$ is a completely undefined relationship, and $u_c, u_k$ are as above. The output of each LBC *is* the mode $m_j$ which changes instantaneously at a discrete event (e.g., triggered by an event in a CTC such as a zero-crossing); in contrast to the situation in DTCs, we assume that the update times (mode changes) often can *not* be anticipated. There may be a computational delay between the trigger event and the mode 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 event.

- LBCs may also exhibit unpredictable state-event or discrete-event behavior – provision will provided for this as in the continuous-time case.
- The lack of a unifying paradigm for logic-based components will, at this time, preclude providing more than a "shell" definition for this class of component; attempting to specify the structure of LBC models beyond the general template above is thus inadvisable.

*Composite Components* (CCs):

- Each CC may have inputs and outputs specified as in the case of the pure CTC, DTC and LBC components. The example in Fig. 1, and the corresponding textual description in [6], illustrates this type of component.
- the CC-specific element of the `body` section is the `connections ...end connections;` section, whose function is obvious.

## 4 Building a System

A *system* is simply completed using the component models of the above types, as follows: First, pure and composite components are built to model every part of the system, as outlined in the preceding sections. Then, signal generators are created for every component input that needs to be defined in this way, of whatever type is appropriate (e.g., a CTC signal generator would be used to generate a sine-wave continuous-time input). Finally, a `System` is defined using the same paradigm as in the

CC definition outlined above, with the additional condition that a `System` may not have input variables in its interface. Such a model is then ready to run under a suitable simulation environment.

## 5 Conclusion

The following comments have governed the developments outlined in the body of this paper:

- HSML should naturally lack many of the features of modern general-purpose high-level languages:
  - to avoid excessive complexity and unnecessary detail, and
  - to allow model-specific support (see below).
- Ancillary modeling support should include:
  - syntactic checks (e.g., is there a state differential equation corresponding to each declared state variable, do all input/output connections involve consistent variable types), and
  - semantic checks (e.g., detection of algebraic loops, checks for the correct use of structures for state-event handling, model structural changes, etc.)
- Note that some of the features of the modeling language outlined above impose significant requirements on the corresponding algorithms for numerical integration.
- In addition, there are many support features that would add to the facility and effectiveness of HSML; for example, these include a graphical connections editor, automatic template generation for CTC, DTC, LBC and CC components, and a language-sensitive editor to provide preliminary elimination of syntactic and elementary semantic errors.

For further detail and for discussions of advanced features of HSML, refer to [14]. Also, see [13, 12] for related modeling issues including a more object-oriented approach to modeling than that presented here.

## References

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

[2] Augustin, D. C., Strauss, J. C., Fineberg, M. S., Johnson, B. B., Linebarger, R. N., and Sansom, F. J., "The SCi Continuous System Simulation Language (CSSL)", *Simulation*, Vol. 9, No. 6, December 1967.

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

[4] 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.

[5] Marttinen, A., *ISEE Interface to Simnon*, Control CAD, Espoo Finland.

[6] Taylor, J. H. "Toward a Modeling Language Standard for Hybrid Dynamical Systems", *Proc. 32nd IEEE Conference on Decision and Control,* San Antonio, TX, 15-17 December 1993.

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

[8] 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, Calif., pp. 218–224, March 17–19, 1992.

[9] 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.*

[10] 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.

[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] Mattsson, S. E. and Andersson, M. "The Ideas Behind Omola", *Proc. CACSD'92, IEEE Computer-Aided Control Systems Design Conference,* Napa, Calif., pp. 218–224, March 17–19, 1992.

[13] 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.

[14] Taylor, J. H. *A Rigorous Modeling Language for Hybrid Systems*, Odyssey Research Associates, Inc., December 1993.

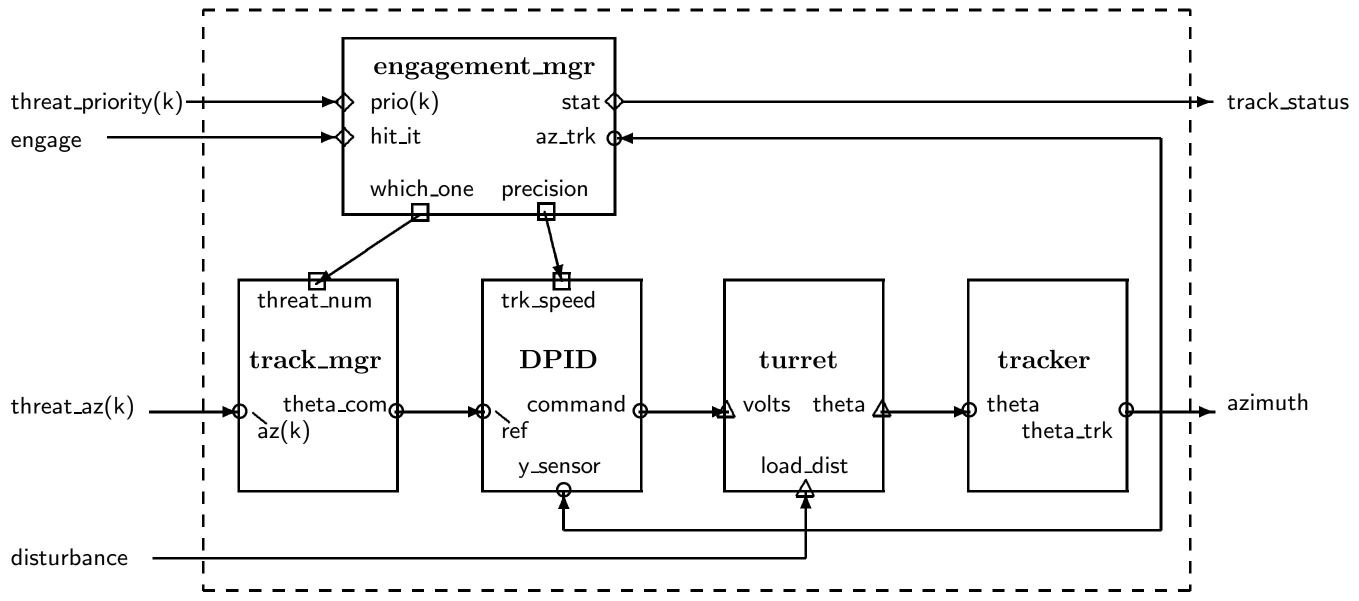

Figure 1: Illustrative Composite Component Model