

SIMULATION OF HYBRID SYSTEMS IN OMOLA/OMSIM

Michael S. Branicky* and Sven Erik Mattsson**

**Department of Electrical Engineering and Applied Physics, Case Western Reserve University, 10900 Euclid Avenue, Glennan 515B, Cleveland, OH 44106-7221 USA. branicky@eeap.cwru.edu*

***Department of Automatic Control, Lund Institute of Technology, PO Box 118, S-221 00, Lund, SWEDEN. svenerik@control.lth.se*

Abstract. Hybrid systems—those composed of the interaction of discrete and continuous inputs, outputs, states, and dynamic equations—are an important class of models of complex, real-world phenomena. However, the simulation tools currently available seem to be (1) *ad hoc* retro-fitting of existing packages, (2) hastily-built new languages, or (3) specialized software for particular subclasses (e.g., piecewise-constant dynamics). Our goal is to produce fast, high fidelity simulations of (networks of) a very broad class of hybrid systems in a user-friendly environment. In this paper, we first review expertise in the mathematical modeling of hybrid systems, viz. the hybrid dynamical systems of Branicky (HDS). Also, we discuss the object-oriented modeling and simulation of combined discrete/continuous systems using the Omola modeling language and Omsim simulation environment developed over the last eight years at Lund. Leveraging these, we are led to our main contribution: a general set of hybrid systems model classes which encompass HDS and hence several other models popularized in the literature that combine finite automata and discrete event systems with ordinary differential (ODEs) and differential algebraic equations (DAEs). These Omola model classes may be viewed as “templates” or “macros” for quick and easy entering of hybrid systems for subsequent analysis and numerically-sophisticated simulation using Omsim.

Key Words. Hybrid systems, simulation, modeling, Omola, Omsim

1. INTRODUCTION

Hybrid systems are those in which a melding of two worlds—the analog and the digital—exists, and they are intertwined to the extent that a “one-world” description is not desired, not tractable, or not possible. Such systems seem to naturally arise in a variety of applications due to autonomous or controlled phenomena.

In the autonomous case, the system evolution itself may fall naturally into a finite number of different phases (a.k.a. modes or epochs), between which abrupt changes in continuous dynamics (switching) or continuous states (jumps or resets) occur. In the controlled case, a simple finite state machine may be used to regulate a physical process, such as may arise even in a simple thermostat. In more complicated situations, a mixture of autonomous and controlled phenomena may be present. Alternatively, a hybrid dichotomy may be invoked as a means of dealing with complexity through abstraction, e.g., using logic to switch among various continuous controllers (each with predictable behavior when used alone) in order to accomplish more global objectives, as in mode-switched aircraft. Of course, such abstractions may occur multiple times, leading to layered or hierarchical control schemes seen in robotics and communication networks.

Other application areas include power electronics (state-dependent circuit switching), motion control (disk drives, transmissions, stepper motors,

position encoders), robotics (constrained robots, flexible manufacturing, interacting agents), intelligent transportation systems (automated highway systems and personal rapid transit), and aerospace (mode-switched flight, vehicle management systems, air traffic control). We invite the reader to imagine more or consult the literature (Branicky, 1995; Alur *et al.*, 1996).

Researchers in hybrid systems have been trying to build a theory for such systems in recent years. However, it is still a challenge to even accurately simulate them because of the sophistication needed to do mixed continuous/discrete simulation in a timely manner. Simulation is not a means unto itself, though, but a means to understanding. Backtrack a moment and consider a slightly more refined definition of a hybrid system which consists of a finite automaton or discrete-event system “supervising” the action of a collection of ODEs or DAEs by giving commands for when to switch between them, and how to update variables upon switching. Clearly, this an instantiation of the digitally-regulated plant and two-layer control schemes mentioned above. It has been demonstrated that even low-dimensional systems of this type possess rich possibilities of behavior, including the power of universal computation in as little as three dimensions (Branicky, 1995). Further, most engineering insight and tools have been developed for “one-world” scenarios.

Thus, with hybrid systems, we are also faced with low intuition and high complexity. In such a situ-

ation, we believe that simulation is an important tool to gaining intuition about, and a stepping stone to the automatic analysis of, hybrid systems. Further, we believe that many of the simulation tools currently available are lacking in some respect. Without classifying tools, they seem fall into one of the following three categories: (1) *ad hoc* retro-fitting of existing “one-world” packages, (2) hastily-built new languages, or (3) specialized software for particular subclasses (e.g., piecewise-constant dynamics). Therefore, it is the goal of this research to *produce fast, high fidelity simulations of (networks of) a very broad class of hybrid systems in a user-friendly environment.*

Our approach builds on a modeling language and simulation package, Omola/Omsim resp., designed from inception to handle mixed discrete/continuous simulation. Omola/Omsim use a sophisticated mix of symbolic equation manipulation, solution approximation, zero-finding, and ODE/DAE solvers to accurately perform such mixed simulations. Omola/Omsim have been under research and development—and test through numerous projects, theses, and users—since 1989 in the CACE group headed by the second author. In this work, we have added a library of hybrid systems *model classes*, which may be thought of as a set of templates or “macros,” designed for the quick specification and simulation of a broad class of hybrid systems: the hybrid dynamical systems (HDS) of Branicky (1995). HDS have been shown to encompass various hybrid phenomena as well as many popular hybrid systems models proposed in the literature. Given this, our macros may be used to easily simulate these systems as well. Indeed, we give explicit *model subclasses* suited to the purpose of simulating several interesting subclasses of hybrid systems: autonomous switching and the models of Brockett (1993) and Artstein (1996).

The paper is organized as follows. Next, we review HDS and present some examples used throughout. In Section 3, we introduce some of Omola and Omsim’s philosophy, syntax, and capabilities. Section 4 presents our main contributions: two different instantiations of HDS model classes, as well as the model subclasses and examples.

2. HYBRID DYNAMICAL SYSTEMS

2.1. Background and Motivation

Hybrid systems involve both continuous-valued and discrete variables. Their evolution is given by equations of motion that generally depend on both. In turn these equations contain mixtures of logic, discrete-valued or *digital* dynamics, and continuous-variable or *analog* dynamics. The continuous dynamics of such systems may be continuous-time, discrete-time, or mixed (sampled-data), but is generally given by differential equations. The discrete-variable dynamics of hybrid systems is generally governed by a *digital automaton*, or input-output transition system with a countable number of states. The continuous and discrete dynamics interact at “event” or “trigger” times when the continuous state hits certain prescribed sets in the continuous state space. *Hybrid control systems* are control systems that

involve both continuous and discrete dynamics and continuous and discrete controls.

In this paper, our focus is on the case of hybrid systems where the continuous dynamics is modeled by a differential equation¹

$$\dot{x}(t) = \xi(t), \quad t \geq 0. \quad (1)$$

Here, $x(t)$ is the *continuous component* of the state, taking values in some subset of a Euclidean space. $\xi(t)$ is a *controlled vector field* that generally depends on $x(t)$, the *continuous component* $u(t)$ of the control policy, and “discrete phenomena” corresponding to discrete states, dynamics, and controls. We have classified the discrete phenomena generally considered into four types (Branicky, 1995):

1. autonomous switching, where the vector field $\xi(\cdot)$ changes discontinuously when the state $x(\cdot)$ hits certain “boundaries”;
2. autonomous impulses, where $x(\cdot)$ jumps discontinuously on hitting prescribed regions of the state space;
3. controlled switching, where $\xi(\cdot)$ changes abruptly in response to a control command;
4. controlled impulses, where $x(\cdot)$ changes discontinuously in response to a command.

Two examples that we will use throughout are

Example 1 (Hysteresis) Consider a system with hysteresis: $\dot{x} = H(x)$, where H is the two-valued function

$$H(x) = \begin{cases} 1, & x < -\Delta, \\ -1, & x > \Delta, \\ \pm 1, & -\Delta \leq x \leq \Delta, \end{cases}$$

with switching between ± 1 whenever $H(x)x \geq \Delta$.

Note that this system is not just a differential equation whose right-hand side is piecewise continuous. There is “memory” in the system, which affects the vector field’s value. Indeed, the system naturally has a finite automaton associated with the function H .

Example 2 (Planar Autonomous Switching)

$$\dot{x}(t) = M_{q(t)}x(t), \quad x(t) \notin A_{q(t)}, \quad (2)$$

$$q(t^+) = \nu(x(t), q(t)), \quad x(t) \in A_{q(t)} \quad (3)$$

where $q \in \{1, 2, \dots, N\}$, $M_q \in \mathbb{R}^{2 \times 2}$, and $A_q \subset \mathbb{R}^2$ for each q .

Two nice classes of hybrid systems from the literature are:

Example 3 (Brockett’s Model)

$$\dot{x} = f(x, p, z), \quad \dot{p} = r(x, p, z), \quad z[p] = \nu(x, p, z[p]).$$

where $x(t) \in X \subset \mathbb{R}^n$, $p(t) \in \mathbb{R}$, and the *rate equation* r is nonnegative for all arguments. The last equation means that z is updated whenever p passes through integer values.

¹ Please refer to (Branicky, 1995) for background, references, and details.

Example 4 (Artstein’s Model) This model consists of a finite number of modules, N , each of which executes its own dynamics, $\dot{x} = f_q(x)$, for a given time T_q . After timeout, execution switches to a different module depending on whether a test function, $\psi_q(x)$, is positive or not.

2.2. A Mathematical Model

We now introduce a mathematical model of hybrid systems that is general enough to cover significant phenomena found in real-world examples and encompass other reasonable models, but is specific enough to build on previous insight and results in order to derive new theory.

We build on the much-studied notion of dynamical system. Briefly, a *dynamical system* is a system $\Sigma = [X, \Gamma, \phi]$, where X is an arbitrary topological space, the *state space* of Σ . The *transition semigroup* Γ is a topological semigroup with identity. The *extended transition map* $\phi : X \times \Gamma \rightarrow X$ is a continuous function satisfying the identity and semigroup properties. We will also denote by dynamical system the system $\Sigma = [X, \Gamma, f]$, where X and Γ are as above, but the *transition function* f is the *generator* of the extended transition function ϕ . In this paper, we restrict our attention to the case where X is a subset of \mathbf{R}^n for some $n \in \mathbf{N}$, $\Gamma = \mathbf{R}_+$, and the dynamics are given by vector fields: $\dot{x} = f(x)$.

A hybrid dynamical system is an indexed collection of dynamical systems along with some map for “jumping” among them (switching dynamical system and/or resetting the state). This jumping occurs whenever the state satisfies certain conditions, given by its membership in a specified subset of the state space. Hence, the entire system can be thought of as a sequential patching together of dynamical systems with initial and final states, the jumps performing a reset to a (generally different) initial state of a (generally different) dynamical system whenever a final state is reached. Formally, a *controlled hybrid dynamical system (CHDS)* is a system $H_c = [Q, \Sigma, \mathbf{A}, \mathbf{G}, \mathbf{C}, \mathbf{F}]$, where

- $Q \subset \mathbf{N}$ is the set of *index* or *discrete states*.
- $\Sigma = \{\Sigma_q\}_{q \in Q}$ is the collection of controlled dynamical systems, where each

$$\Sigma_q = [X_q, \mathbf{R}_+, f_q, U_q, Y_q].$$

Here, the $X_q \subset \mathbf{R}^{d_q}$, $d_q \in \mathbf{N}$, are the *continuous state spaces* and the vector fields $f_q : X_q \times U_q \rightarrow \mathbf{R}^{d_q}$ represent the *continuous dynamics*. $U_q \subset \mathbf{R}^{m_q}$ and $Y_q \subset \mathbf{R}^{p_q}$ are the *input* and *output spaces*, resp., with $m_q, p_q \in \mathbf{N}$.

- $\mathbf{A} = \{A_q\}_{q \in Q}$, $A_q \subset X_q$ for each $q \in Q$, is the collection of *autonomous jump sets*.
- $\mathbf{G} = \{G_q\}_{q \in Q}$, where $G_q : A_q \rightarrow S$ is the *autonomous jump transition map*, said to represent the *discrete dynamics*.
- $\mathbf{C} = \{C_q\}_{q \in Q}$, $C_q \subset X_q$, is the collection of *controlled jump sets*.
- $\mathbf{F} = \{F_q\}_{q \in Q}$, where $F_q : C_q \rightarrow 2^S$, is the collection of *controlled jump destination maps*.

Thus, $S = \bigcup_{q \in Q} X_q \times \{q\}$ is the *hybrid state space* of H_c . The dynamics of H_c are as follows. The

system is assumed to start in some hybrid state in $S \setminus A$, say $s_0 = (x_0, q_0)$. It evolves according to $f_{q_0}(\cdot, u)$ until the state enters—if ever—either A_{q_0} or C_{q_0} at the point $s_1^- = (x_1^-, q_0)$. If it enters A_{q_0} , then it *must* be transferred according to transition map $G_{q_0}(x_1^-)$. If it enters C_{q_0} , then we *may* choose to jump and, if so, we may choose any destination point in $F_{q_0}(x_1^-)$. Either way, we arrive at a point $s_1 = (x_1, q_1)$ from which the process continues.

The case where the sets U_q , \mathbf{C} , and \mathbf{F} above are empty is simply a *hybrid dynamical system (HDS)*.² In equations, it might look as follows:

$$\left. \begin{aligned} \dot{x}(t) &= f(x(t), q(t)), & x(t) &\notin A_{q(t)}, \\ q(t^+) &= G_q(x(t), q(t)) \\ x(t^+) &= G_x(x(t), q(t)) \end{aligned} \right\}, \quad x(t) \in A_{q(t)}.$$

3. OMOLA AND OMSIM

3.1. Omola

Omola stands for “Object-oriented Modeling Language.” A recent, comprehensive description may be found in (Andersson, 1994), to which we refer the reader for more details.

Omola is a language for describing dynamic models that is

- user-oriented (engineers specify models, computer scientists don’t program them).
- high-level and textual.
- supports reuse, through an object-oriented philosophy.
- directly usable in simulation and analysis tools, e.g., Omsim.

Omola supports mixed continuous evolution and discrete events. It has an object-oriented modeling philosophy. Models are to be perceived as *declarative*, encoding facts and relations. They are not thought of as procedures with inputs, outputs, and flow of information. Thus, Omola supports acausal physical models. E.g.,

$$R \times I = V, \quad R = V/I, \quad V/R = I. \quad (4)$$

are all equally valid descriptions of a resistor.

It is the view of Omola that *models are classes*. That is, *models are descriptions of a system type, not a representation of a particular system*. Thus, any one of the equations in (4) is a description of a system type, viz. **Resistor**. A particular resistor is instantiated by adding the fact that, e.g., $R = 2\text{k}\Omega$. Note that before a model can be simulated, all variables must be fully instantiated (this is checked automatically by Omsim).

Omola supports model *inheritance* as a mechanism for information sharing and reuse. We note that inherited attributes may be selectively overridden. E.g., Here is a fragment of Omola code for a resistor viewed as a two-port device (not shown).

² The case of GHDS with $|Q|$ finite is a coupling of finite automata and differential equations and includes previously posed hybrid systems models, systems with impulse effect, and hybrid automata (Branicky, 1995).

Omola 1 (Resistor Example)

```
Resistor ISA TwoPort WITH
  R ISA Parameter WITH default := 1.0; END;
  V1 - V2 = R * (I1 - I2);   I1 = I2;   END;
```

Reserved words are in all capitals. Inheritance is accomplished through the **ISA** (and **ISAN**) command, with refinements given after **WITH**. The reader may guess from this that **Parameter** is an Omola base class consisting of a real value plus a default value. The symbol **:=** denotes equality with causality specified: the left-hand side is dependent on the right-hand side. The equality sign denotes an acausal relation. Below, **Model** is a basic Omola class, which the user may think of as an “empty slot.”

3.2. Omsim

Omsim (“**Omola Simulation**” package)

- provides a graphical model editor, used for input, visualization, and editing of models.
- compiles Omola to simulation code, performing consistency checks, determining causality, solving initialization problems, etc.
- provides simulation tools, including plotters, access windows for changing parameters and initial conditions; output routines for exporting to files, printers, or Matlab; a variety of DAE and ODE solvers.

3.2.1. *Continuous Dynamics.* Omsim can handle continuous-time dynamics given by DAEs:

$$\dot{x} = f(x, y, t), \quad 0 = g(x, y, t).$$

It also allows $g = g(\dot{x}, x, t)$. Omsim does sophisticated pre-processing of such models in compiling them into simulation code, including symbolic manipulation and index reduction.

3.2.2. *Discrete Events.* The Omola fragment **x TYPE DISCRETE Real;** means that the value of **x** changes only at discrete times, i.e., $\dot{x} = 0$. Integers and booleans are automatically **DISCRETE**'s. Events, their conditions, and their side effects are specified using a template as follows:

```
<event name> ISAN Event WITH
  CONDITION := <logical expression>;
  <body of actions> END;
```

Here, the logical expression can depend on a continuous variable, e.g., $(y > 0)$ for **y TYPE Real**; a discrete variable, e.g., $(i == 1)$ for **i TYPE Integer**; or be edge-triggered, e.g., $\hat{(y > 0)}$. In resetting variables, the **NEW** operator is used to distinguish previous from new values. Events may also be scheduled to occur after some delay; scheduled events may be descheduled:

```
schedule(EventName, Delay); deschedule(EventName);
```

Event propagation can be specified as follows:

```
WHEN <cond.> [CAUSE <events>] [DO <actions>] END;
```

3.2.3. *Detecting State-Dependent Events.* Omsim uses sophisticated techniques for dealing with mixed continuous/discrete events. The full discussion is lengthy (Andersson, 1994; pp. 200–208).

4. SIMULATING HDS

We first discuss simulating HDS using Omsim, saving the more general case of CHDS for later. There are two main approaches for representing HDS: distributed state and global state.

In the *distributed state* approach, one views the HDS as consisting of N different dynamic systems, each of which always exists and is either “active” (controlling the dynamics) or “inactive.” Here, the state of the simulation is given by the concatenation of the states of each of the N systems, $[x_1, \dots, x_N]^T$ plus an integer, q , representing the currently active one. If a system is inactive, its state does not change, which is represented by setting its vector field to zero in that case.

In the *global state* approach, $q(t)$ denotes the active system. It dynamically redefines the vector field $f_{q(t)}$, the test set $A_{q(t)}$, and the transition map $G_{q(t)}$. Here, we use one global continuous state, $x(t)$, which must be given a size equal to the largest dimension of the q 's.

4.1. Distributed State HDS

We now build up a distributed state HDS model class, piece by piece. We start with a dynamic system (DS) given by an ODE:

Omola 2 (DS class)

```
DS ISA Model WITH
  d TYPE Integer;           % model order
  fx, x TYPE Column [d];   % vector field, state
  x' = fx;                  % behavior
  END;
```

The prime denotes derivation. Hence, this model simply says that the state has dimension **d** and the dynamics are given by a **d**-dimensional vector field **fx**. Hence, $\dot{x} = f(x)$, $x \in \mathbf{R}^d$.³

Now, we add the capability of switching the ODE between **Active** and **Inactive** modes. Recall that when in the inactive mode, the vector field is set to zero.

Omola 3 (Switched DS class)

```
SwitchedDS ISA DS WITH
variables:
  A TYPE Real;
  mode TYPE DISCRETE (Active, Inactive);
  Init, Restart ISAN EventInput;
  AutoJump ISAN EventOutput;
  f TYPE Column [d];
equations:
  fx = IF mode=='Inactive' THEN 0.0 ELSE f;
events:
  WHEN mode=='Active' and A >= 0 CAUSE AutoJump DO
    NEW(mode):='Inactive'; END;
  WHEN Restart DO NEW(mode):='Active'; END;
  END;
```

Note we have encoded $x \in A$ by a real value **A**'s (whose functional form will be specified in instantiated models) being nonnegative.

³ **Column**'s, **Row**'s, and **Matrix**'s are only real-valued; indexing conventions are those widely used, e.g., as in Matlab.

What we would want now is to have a hybrid model that is a *vector* of `SwitchedDS`'s, e.g.,

Omola 4 (Distributed-State HS)

```
HybridSystem ISA Model WITH
  N, Q TYPE Integer;    % # disc./active state(s)
  Sigma TYPE SwitchedDS Column [N];  END;
```

Unfortunately, Omola does not currently support vectors of models—though this capability is being actively pursued. Of course, one can define templates—just once, then stored in a library—for each size of hybrid system. One can save some typing by doing this in an inductive manner using inheritance. However, below is an explicit example of a hybrid system with two discrete states, which can be generalized to higher N .

Omola 5 (2 Distributed-State HS)

```
HybridSystem2 ISA Model WITH
  Q, d TYPE Integer;
  x, Gx1, Gx2 TYPE Column[d];
components:
  Q1, Q2 ISA SwitchedDS;
equations:
  x = IF Q==1 THEN Q1.x ELSE Q2.x;
  WHEN Q1.AutoJump CAUSE Q2.Restart DO
    new(Q2.x) := Gx1; END;
  WHEN Q2.AutoJump CAUSE Q1.Restart DO
    new(Q1.x) := Gx2; END;  END;
```

We now give an Omola model of our hysteresis example using the above model classes.

Omola 6 (Hysteresis Example)

```
Mode1 ISA SwitchedDS WITH
  Delta ISA Parameter WITH default := 0.1; END;
  d = 1;  f = 1;  A = x-Delta;  END;

Mode2 ISA SwitchedDS WITH
  Delta ISA Parameter WITH default := 0.1; END;
  d = 1;  f = -1;  A = -x-Delta;  END;

Hysteresis ISA HybridSystem2 WITH
  d = 1;  Q1 ISA Mode1;  Q2 ISA Mode2;
  Gx1 := Q1.x;  Gx2 := Q2.x;  END;
```

Each mode has dimension one, with dynamics given by either ± 1 . Switching occurs from `Mode1` to `Mode2` whenever $x \geq \Delta$. Here, we have specified Δ to be a parameter, `Delta`, with default value 0.1. In Omsim, we might look at simulation results with different values of the parameter, which could be changed using the aforementioned access windows. Finally, note the reset maps are the identity since only the dynamics, and not the continuous state, is reset upon changing modes.

4.2. Global State HDS

Here is a global state model class of HDS. It parallels the equations of Section 2 closely enough that the reader should be able to follow the code, perhaps using the comments.

Omola 7 (Global State HS)

```
HDS ISA Model WITH
states:
  d TYPE Integer;          % max state dim.
  x TYPE Column [d];      % continuous state
  N TYPE Integer;         % # discrete states
  q TYPE DISCRETE Integer; % discrete state
```

```
continuous_dynamics:
  f TYPE Matrix [d,N];    % vector fields
  x' = f[1..d,q];
discrete_transitions:
  % autonomous switching functions and map
  Ajump TYPE Row [N];
  Gq TYPE Row [N];  Gx TYPE Matrix [d, N];
  ModeChange ISA Event;
  WHEN Ajump[q] >= 0 CAUSE ModeChange DO
    new(q):=Gq[q];  new(x):=Gx[1..d,q];  END;
END;
```

Note that we have added a `ModeChange` event that could be used by Omsim to count events, change plotter colors, etc. upon mode changes. Finally, here is an Omola model of our hysteresis example using the HDS model class.

Omola 8 (Hysteresis Example)

```
Hysteresis ISA HDS WITH
  Delta ISA Parameter WITH default := 0.1; END;
  N = 2;  d = 1;  f[1,1] = 1;  f[1,2] = -1;
  Ajump[1]=x-Delta;  Ajump[2]=-x-Delta;
  Gq[1]=2; Gq[2]=1;  Gx[1,1]=x; Gx[1,2]=x;
END;
```

For the remainder of this paper, we will talk *only of the global state model class* HDS above and subclasses thereof.

4.3. Subclasses and Examples

In this section we show how several classes of hybrid systems models may be easily recovered as subclasses of the global state HDS above. An autonomous switched hybrid system is one where the continuous state does not change at autonomous jump times, that is, where the projection of the reset map G onto the continuous state is the identity: $G_x(x, q) = x$ for each discrete state q .

Omola 9 (Autonomous-switched HS)

```
AutoSwitch ISA HDS WITH Gx=x*ones(1,N); END;
```

Here is a special case of an HDS, Artstein's model (cf. Example 4), which can naturally be represented as a subclass of HDS:

Omola 10 (Artstein's HS)

```
Artstein ISA HDS WITH
  dim TYPE Integer;  T TYPE Row [N];
  d = dim + 1;
  f[d,1..N] = ones(1,N);
  Ajump = x[d]*ones(1,N) - T;
  Gx[1..d-1,1..N] = x[1..d-1]*ones(1,N);
  Gx[d,1..N] = zeros(1,N);
END;
```

Note that if the state has dimension `dim`, then the model has one more dimension, with `x[d]` representing the timer value τ in Example 4. The vector `T` denotes the vector of timeout values, $[T_1, \dots, T_N]^T$. As expected, jumps occur whenever the timer, `x[d]` is greater than or equal to the timeout value. Note also that this variable is reset to zero—and the continuous state otherwise does not jump—upon timeout, no matter what the discrete state.

Brockett's model (cf. Example 3) may also be recovered as a subclass of the HDS model class:

Omola 11 (Brockett's HS)

```

Brockett ISA HDS WITH
  dim TYPE Integer;
  p, pfrac TYPE Real; % p, its fractional part
  r TYPE Row [N]; z TYPE DISCRETE Integer;
  d=dim+2;
  p=x[d-1]; pfrac=x[d]; z=q;
  f[d-1,1..N]=0; f[d,1..N]=r;
  Ajump = x[d]*ones(1,N)-ones(1,N);
  Gx[1..d-2,1..N] = x[1..d-2]*ones(1,N);
  Gx[d-1,1..N] = (x[d-1]+1)*ones(1,N);
  Gx[d,1..N] = zeros(1,N); END;

```

Note that we added two more dimensions, representing the counter, p , and its fractional part. Note also that jumps occur whenever the counter's fractional part, $x[d]$, is equal to one (not at integer points); we compensated by resetting it to zero at jump times, yielding an equivalent description.

Below is a model class for switching among two planar linear systems. Since it is autonomous switching, we can make it a subclass of `AutoSwitch`. Note that the switching surfaces, given by the `Ajump` vector (inherited from `HDS` through `AutoSwitch`), are still unspecified.

Omola 12 (Switched Planar HDS)

```

Switcher ISA AutoSwitch WITH
  N=2; d=2;
  A TYPE MATRIX [2, 2] := [1, -100; 10, 1];
  B TYPE MATRIX [2, 2] := [1, 10; -100, 1];
  f[1..2,1] = A*x; f[1..2,2] = B*x;
  Gq[1] = 2; Gq[2] = 1; END;

```

Using this class, we can investigate different switching rules for the same underlying system, e.g.,

Omola 13 (Two Switching Systems)

```

SwitchQuadrant ISA Switcher WITH
  Ajump[1]=IF (x[1]>=0 AND x[2]<=0) THEN 1 ELSE -1;
  Ajump[2]=IF (x[1]<0 AND x[2]>=0) THEN 1 ELSE -1;
  END;

```

4.4. Adding Control

The power of Omola's object-oriented philosophy, with reuse via subclasses, allows us to easily add controlled versions of `DS`'s, `HDS`'s, and `Artstein`'s. A `CHDS` is simply a subclass of `HDS` with inputs, outputs, and controlled switching added.

Omola 14 (CHDS)

```

CHDS ISA HDS WITH
  inputs_outputs:
    m,p TYPE Integer; % max control/output dim.
    u TYPE Column [m]; % continuous control
    y TYPE Column [p]; % continuous output
  controlled_transitions:
    Cjump TYPE Row [N];
    Fq TYPE Row [N]; Fx TYPE Matrix [d, N];
    WHEN Cjump[q] >= 0 CAUSE ModeChange DO
      new(q):=Fq[q]; new(x):=Fx[1..d,q]; END;
  END;

```

A `CDS` ISA `DS` WITH the `inputs_outputs` codeblock of `CHDS` above as refinement. A `CARTstein` ISA `CHDS` WITH the same codeblock as for `Artstein`. Hence, the code below instantiates the closed-loop system consisting of a simple harmonic oscillator and Pait's two-state hybrid controller (Artstein, 1996).

Omola 15 (SHO)

```

SHO ISA CDS WITH
  d=2; m=1; p=1;
  f[1]=x[2]; f[2]=-x[1]+u[1]; y[1]=x[1]; END;

Cont2 ISAAN CARTstein WITH
  delta ISA Parameter WITH default := 0.1; END;
  N = 2; dim = 0; m = 1; p = 1;
  y[1]=IF q==1 THEN 0 ELSE -3*u[1];
  Gq[1] = IF u[1] >=0 THEN 2 ELSE 1; Gq[2] = 1;
  Fq[1] = 1; Fq[2] = 1;
  Fx[1,1] = x; Fx[1,2] = x;
  Cjump[1] = -1; Cjump[2] = -1;
  T[1] = delta; T[2] = 2.3562; END;

```

```

Full12 ISA Model WITH
  Plant ISA SHO WITH u[1]=Controller.y[1]; END;
  Controller ISA Cont2 WITH u[1]=Plant.y[1]; END;
  END;

```

Note how easy it is to connect the subcomponents in making `Full12`. Figure 1 shows a screendump of an Omsim simulation of the system. The two plot windows, from left to right, are the SHO phase plane plot and controller state versus time.

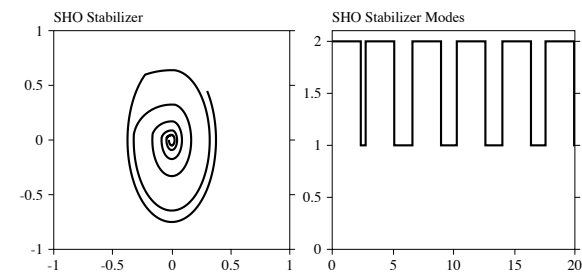


Fig. 1. Omsim simulation of Pait's SHO stabilizer.

5. CONCLUSIONS

We presented Omola model classes for hybrid systems that encompass HDS and hence several other hybrid systems models (as subclasses). Our classes may be viewed as “macros” for quick and easy entering of networks of interacting hybrid systems for subsequent analysis and numerically-sophisticated simulation using Omsim.

6. ACKNOWLEDGMENTS

This work was done while Branicky was visiting Lund in May/June 1996; he thanks Profs. K.J. Åström and S.K. Mitter for making the visit possible. Omola/Omsim may be freely downloaded at <http://www.control.lth.se/~cace/omsim.html>.

7. REFERENCES

- Alur, R., T.A. Henzinger and E.D. Sontag (Eds.) (1996). *Hybrid Systems III*. Springer, NY.
- Andersson, M. (1994). Object-Oriented Modeling and Simulation of Hybrid Systems. PhD thesis. Lund Inst. of Tech. Dept. Automatic Control.
- Artstein, Z. (1996). Examples of stabilization with hybrid feedback. In (Alur *et al.*, 1996), 173–185.
- Branicky, M.S. (1995). Studies in Hybrid Systems: Modeling, Analysis, and Control. ScD thesis. Massachusetts Inst. of Tech. Dept. EECS.
- Brockett, R.W. (1993). Hybrid models for motion control systems. In: *Essays in Control* (H.L. Trentelman and J.C. Willems, Eds.), 29–53. Birkhäuser, Boston.