HyAuLib: modelling Hybrid Automata in Modelica

Tiziano Pulecchi Francesco Casella
Politecnico di Milano, Dipartimento di Elettronica e Informazione
Piazza Leonardo da Vinci 32, 20133 Milano, Italy

Abstract

A library of components for modelling hybrid automata in a natural fashion has been implemented in Modelica. This library exploits and extends the free Modelica library StateGraph to the modelling and simulation of deterministic hybrid systems described by the hybrid automaton formalism. In this contribution the library's main features are described and its flexibility highlighted by developing models for two classic hybrid systems literature examples.
Keywords: hybrid automata; simulation

1 Introduction

Hybrid systems (see [7]) are dynamical systems involving the interaction of both continuous state and discrete state dynamics. Recall that a state variable is called discrete if it can take a finite (or countable) number of values and continuous if it takes values in the Euclidean space R^n for some $n \geq 1$. By their nature, discrete states can change value only through a discrete *jump*; on the other hand, continuous states can change values either through a *jump*, or by *flowing* in time according to a given differential equations set.

Physical systems are by their own nature inherently continuous. Nevertheless, because of the couplings with very high frequency dynamics, or in presence of mechanisms too complicated to be dealt with in simulation by a sound physical description, many physical systems can be conveniently represented under the hybrid systems paradigm. This provides a convenient framework for modelling systems in a wide range of engineering applications, including for instance electrical circuitry, where continuous dynamic is affected by switches opening and closing; chemical processes control, where the continuous evolution of chemical reactions is controlled by valves and pumps; or digital control, where digital computers interact with a continuous time physical system. Of course, highly nonlinear systems such as for instance diodes, switches, valves, mechanical backlashes and dead strokes, can be conveniently described via abstracted hybrid models.

The analysis and design of hybrid systems is in general more demanding than that of purely discrete or purely continuous systems, because of the necessity to accurately deal with the interplay between the discrete and continuous dynamics. The same consideration holds true for their simulation, that presents specific challenges requiring special care. Specifically, it is of paramount importance to be able to determine with great accuracy the time instant when discrete jumps take place, and consistently deal with simultaneous events occurrences, which represents one of the main sources of modelling inconsistencies.

Nowadays, general purpose simulation packages such as Matlab and Simulink can deal adequately with most complications. Specialized packages have also been developed that allow accurate simulation of hybrid systems (see *e.g.* [2], [3], [1], [5]). The interested reader is addressed to [4] for a thorough overview on the subject.

In this paper a library of components for modelling (autonomous) hybrid automata (HyAuLib), implemented in Modelica (see [9, 6]), has been designed. Modelica is already capable of efficiently handle hybrid systems modelling and simulation via suitable scripts (see [8]). Nevertheless, sometimes this operation can turn out to be very cumbersome and error prone for the unexperienced user. This library, extending the free Modelica library StateGraph (see [10]), overcomes these difficulties by providing an easy way to the consistent modelling and simulation of hybrid systems described by the hybrid automaton formalism.

The paper is organized as follows: in Sections 2 and 3 the hybrid automaton formalism and the implemented Modelica HyAuLib will be respectively described. In Section 4, the library capabilities shall be illustrated on two classic hybrid systems textbook examples. Fi-

nally, in Section 5 concluding remarks and future developments will be presented.

2 Hybrid Automata

A hybrid automaton is a dynamical system describing the evolution in time of a set of discrete and continuous variables. In this paper we will focus on autonomous hybrid automata, *i.e.* hybrid automata which have no inputs nor outputs. More specifically, the transitions between two modes of our automata shall occur in accordance with a user-specified determinism. This topic will be thoroughly discussed in Section 3. The hybrid automaton will answer to the following definition (see *e.g.*[7]):

Definition 2.1 (Hybrid Automaton) A hybrid automaton H is a collection $H = (Q, X, f, Init, D, E, G, R)$ where

- $Q = \{q_1, q_2, \ldots\}$ is the set of all admissible discrete states, or *modes* of H;
- $X \subseteq R^n$ is the set of continuous states;
- $f(\cdot,\cdot) : Q \times X \to R^n$ is a *vector field*, defining the evolution in time of the continuous part of the state of H;
- $Init \subseteq Q \times X$ is the set of all admissible initial states for H;
- $Inv(\cdot) : Q \to P(X)$ is a the *invariant set* or domain;
- $E \subseteq Q \times Q$ is a set of *edges*, defining all transitions from one mode of H to the next;
- $G : E \to P(X)$ is the set of *guard* conditions;
- $R(\cdot,\cdot) : E \times X \to P(X)$ is a *reset map*,

where $P(X)$ denotes the power set (the set of all possible subsets of X), and the pair $(q,x) \in Q \times X$ is the state of H, made up by its discrete and continuous contributors.
Hybrid automata define possible evolutions for their state. Roughly speaking, starting from an initial value $(q_0, x_0) \in Init$, the continuous state x flows according to the differential equation

$$\begin{cases} \dot{x} = f(q_0, x) \\ x(0) = x_0 \end{cases}$$

defined by the hybrid automaton vector field, while the discrete state q remains constant, *i.e.*,

$$q(t) = q_0$$

as long as $x \in Inv(q_0)$. If at some point x reaches the guard $G(q_0, q_1) \subseteq R^n$ of some edge $(q_0, q_1) \in E$, then the discrete part of the state q *may* change to q_1. If this happens, x is reset according to the reset map $R(q_0, q_1, x)$. After a discrete transition has taken place, continuous evolution resumes, until a new transition is triggered, and so on.
To define the time horizon over which the states of the hybrid system evolve, we need to introduce the concept of Hybrid Time Set by the following definition (see *e.g.* [7]):

Definition 2.2 (Hybrid Time Set) A hybrid time set is a sequence of contiguous intervals $\tau = \{I_0, I_1, \ldots, I_n\}$ finite or infinite such that

- $I_i = [\tau_i, \tau_i']$ for all $i < n$;
- if $N < \infty$ then either $I_N = [\tau_N, \tau_N']$ or $I_N = [\tau_N, \tau_N')$;
- $\tau_i \leq \tau_i' = \tau_{i+1}$ for all i,

where τ_i' represents the time instant immediately preceding a discrete transition occurrence, whereas τ_{i+1} corresponds to the time instant just following the discrete transition. The adoption of this representation of time in hybrid automata allows the handling of situations where multiple transitions occur simultaneously. When the range of a generic interval I_j shrinks to a single value τ_j, it means that the associated mode Q_k has been entered and exited in the very same instant. If multiple transitions are enabled and do occur simultaneously, the automaton evolves to its new mode Q_r passing through several intermediate modes, whose associated residing times are zero.
The triple (τ, q, x) consisting of a hybrid time set and two sequences of functions $q = \{q_i\}$ and $x = \{x_i\}$ is named a *hybrid trajectory*, whereas an *execution* of H is a hybrid trajectory (τ, q, x) admissible by the hybrid automaton H.
Note in passing that the exploitation of the invariant set definition, allowable within the hybrid automaton formalism, could be used to efficiently simulate a broad variety of *unconventional* engineering applications, such as, for instance, the suitability of the designed safety procedures for continuous dynamical systems. This could be achieved simply by modelling the safety critical conditions for the system via undesirable regions for the continuous state (by defining the automaton domain accordingly). As a consequence, if the recovery procedures fail, the automaton will violate the invariant set and the simulation will be terminated with a warning message specifying the safety

critical condition violated, leading to a procedure redesign. Many other useful controls of this kind, such as for example *Zeno behavior* detection, can be easily incorporated into the Modelica HyAuLib.
Another interesting field of application for the HyAuLib could be in the framework of simulation of systems undergoing failures. Let's focus on a simple example, where a relief valve is used to control the pipeline pressure in an hydraulic plant. During nominal operational regime, the valve remains closed while pressure at the valve inlet is lower than the valve preset pressure. When the preset pressure is reached, the valve is opened and the pressure at the inlet reduced. Now, if the valve experiences a failure and got stuck in the open position, and for some operational reason the pressure in the pipeline crosses the valve preset pressure and keeps rising, either the valve's backup (if any) will be activated, or the system will suffer damage and loose functionality. If no backup is activated, or it results ineffective, the pressure will utterly increase to the point of exceeding a new threshold value (defining when the system is no more operative), specified via the invariant set. The simulation will be either terminated if the experienced failure is classified as safety significant or safety critical (the system architecture needs a redesign), or kept running with the system in failure. Note that our simple example requires that the transition between the operational modes of the pressure relief valve (open and close) is triggered either by an opening (*resp.* closing) command or as a consequence of a failure experienced by the equipment and associated to a probability of occurrence. Relevant data both in terms of failure modes and failure rates can be obtained from the equipment's Failure Mode Effect and Criticality Analysis document, and the necessary hybrid models easily implemented exploiting the HyAuLib models, described in the following Section 3.

3 The Modelica HyAuLib

The Modelica HyAuLib addresses the problem of supporting the designers working with hybrid systems, by providing them with an efficient and intuitive modelling and simulation tool for hybrid automata. The library has been derived by extending the free Modelica StateGraph library by Otter and Dressler (see [10]), which is based on the JGraphChart method and provides components to model finite state machines. The HyAuLib allows for the modelling of complex hybrid systems that can be represented throughout the hybrid automata paradigm in a natural fashion. Although such models could of course be obtained by writing explicitly the relevant Modelica code, this task is likely to turn out to be burdensome and error prone even for very simple models. The HyAuLib, by automatically managing all state transitions and the discrete/continuous domains interaction, seeks to minimize all possible sources of wrong modelling behavior. An expanded view of the HyAuLib tree is given in Figure 1.

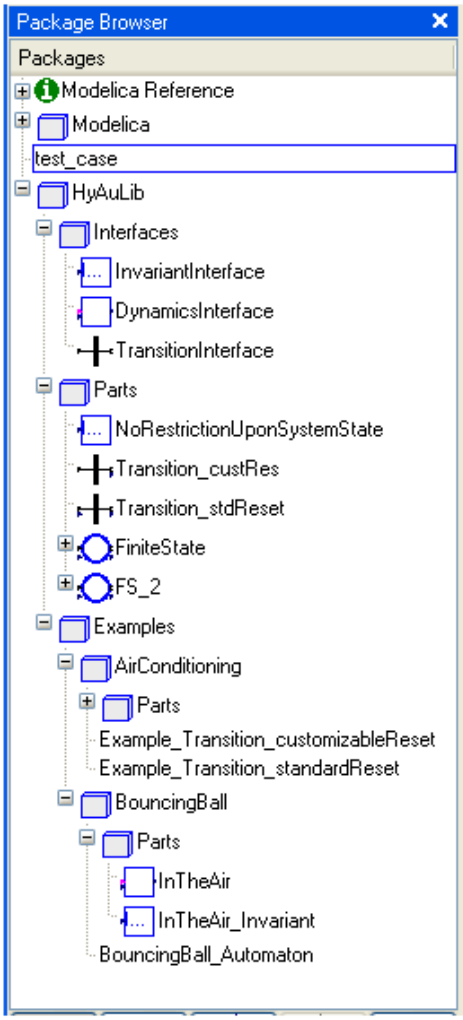

Figure 1: The Modelica HyAuLib library's tree.

The Modelica HyAuLib encompasses two basic components, the *FiniteState* and *Transition* models, which are briefly summarized in the following:

- Component *FiniteState*: defines each finite state (or mode) for the generic automaton. Embedded in this model are the definitions of the vector field and of the invariant set associated to the present state of the hybrid automaton. Component *FiniteState* extends StateGraph's *Step* com-

ponent, which is used to define which state is presently active. (See Figure 2, where the Modelica code used to generate the *FiniteState* component is shown.)
The *FiniteState* options selection mask is shown in Figure 3. For every *FiniteState* component instantiated in the hybrid automaton model, the number of input and output connections needs to be specified, jointly with the selection between StateGraph's InitialStep and Step components (for further information on StateGraph, see [10]). Finally, notice that a given time continuous dynamics and invariant set must necessarily be specified. These models can be elementarily defined by extending suitable interfaces provided within the HyAuLib library. Several of such examples are provided in the library's *Example* folder.

- Component *Transition*: defines the generic transition between modes. The model, extending StateGraph's *Transition* component, comprises the definition of the guard and the reset conditions. Notice that all transitions are triggered according to a deterministic mechanism, *i.e.*, at the moment no provision is given within the library in order to assign a probability to the transition. Two possible reset conditions are available: the standard option guarantees the continuity of the continuous state variable throughout the transition, whereas the second option allows for the definition of a specified reset. The Modelica code for the *Transition* model is provided in Figure 4. Also, Figure 5 shows the options selection mask for the HyAuLib *Transition* component.

Both models take full advantage of the Modelica *redeclare* construct feature, which makes it possible to create general classes which are defined only when the model is instantiated. It is then possible, once suitable models for the hybrid automaton's dynamics, invariant sets, guards and reset conditions have been defined, to simply drag and drop in the automaton model the base *FiniteState* and *Transition* models, select the relevant features from the graphical user interface, and connect them to reproduce the automaton scheme.
The HyAuLib supports multiple edges connection between different modes (the number of input and output connections being a parameter of the *mode* component). The transition mechanism adopted is deterministic. The transition is triggered as soon as the guard condition is satisfied, or with a delay that can be specified as a function of the hybrid automaton state and current time. Future development will comprise the implementation of a probabilistic approach in the definition of the transition occurrence.
Notice that no care needs to be taken in the HyAuLib with respect to the definition of an hybrid time set for the automaton execution. Modelica is indeed capable of dealing with this issue, requiring no further modelling endeavor.

4 Case studies

In the following, two classic applications, which have been extensively discussed in the hybrid systems control literature, are presented to illustrate the HyAuLib capability when modelling hybrid automata.

4.1 Bouncing ball

A bouncing ball is a very effective example of an highly nonlinear dynamical system, which can be conveniently represented as a simple hybrid automaton with a single discrete state, describing the ball being above the ground. Here, all the system nonlinearities are easily modelled by introducing a hybrid component in the model.
The system state is a two dimensional vector comprising the ball's center of gravity height from ground and its derivative, the vertical velocity. The state continuous time evolution is then described by

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = -9.81 \end{cases}$$

where x_1, x_2 are the vertical position and velocity respectively, and the associated invariant is given by the condition $x_1 \geq 0$ (ball in the air).
The only transition possible occurs from the state to itself when the ball hits the ground: the associated guard condition is then

$$x_1 = 0 \;\; \textit{and} \;\; x_2 \leq 0 \, .$$

A nonconservative description of the phenomenon may be easily accounted for by acting upon the reset condition. Proceeding like that, we could easily force an energy loss due to the deformation of the system simply by setting

$$x_2 := -c\, x_2$$

with c non negative and less than unity.

```
model FiniteState
  "Interface for the generic finite state of the Hybrid Automaton"

  parameter Integer nIn(min=0)=1 "Number of input connections";
  parameter Integer nOut(min=0)=1 "Number of output connections";

  replaceable model StepType = Modelica.StateGraph.StepWithSignal
    extends StepType
    "choice between StaeGraph's InitialStep and Step (default) blocks" a;

  replaceable model ContinuousDynamics =
    HyAuLib.Examples.AirConditioning.Parts.Dynamics.PowerON extends
    HyAuLib.Interfaces.DynamicsInterface
    "Options for Automaton's time continuous dynamics model" a;

  replaceable model Invariant =
    HyAuLib.Examples.AirConditioning.Parts.Invariants.PwrOn_Invariant
    extends HyAuLib.Interfaces.InvariantInterface
    "Options for Automaton's Invariant set (time continuous dynamics Domain)" a;

equation

end FiniteState;
```

Figure 2: Modelica code of the *FiniteState* component.

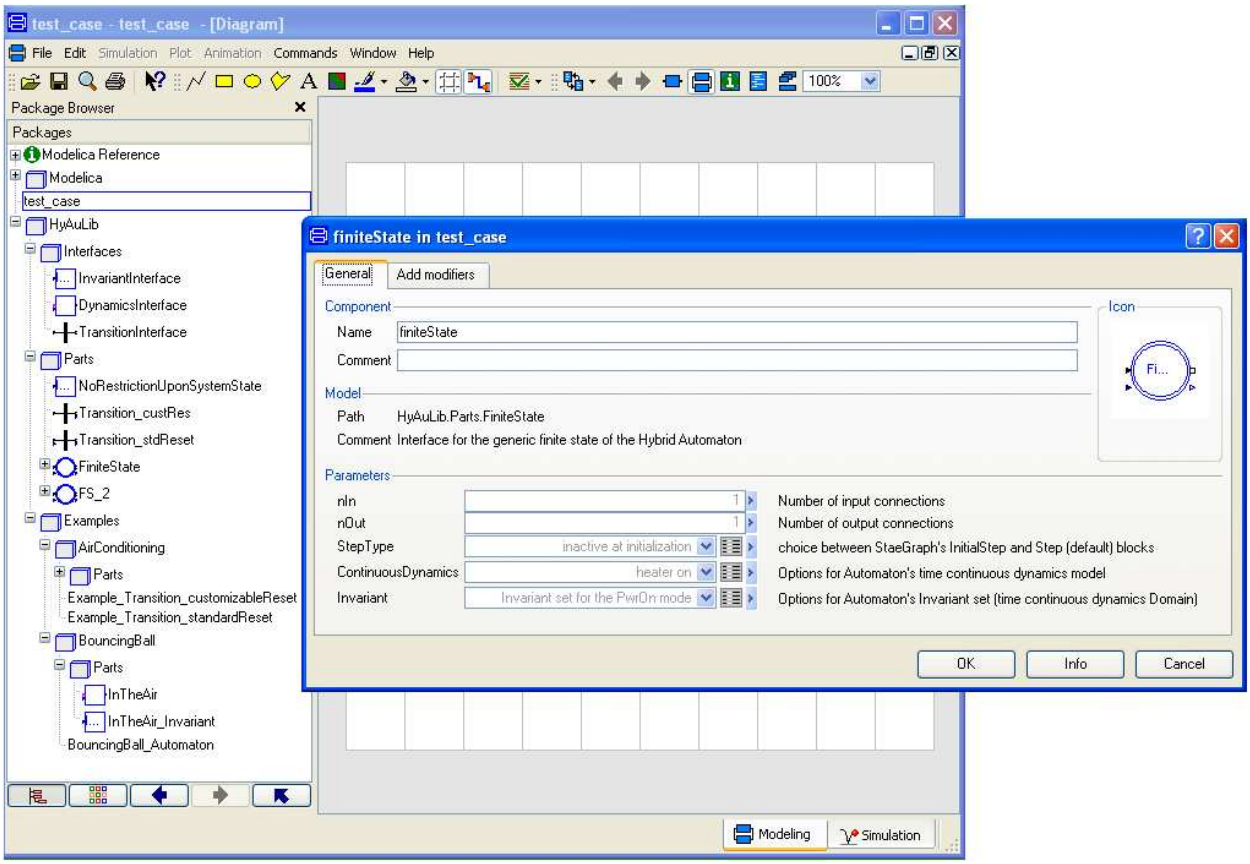

Figure 3: User's selection mask for the HyAuLib library's *FiniteState* component.

```
model Transition_custRes
  "Model for transition from finiteState_In to finiteState_Out with customizable reset"

  extends Interfaces.TransitionInterface;

  // reset settings
  parameter Integer nOut(min=1)=1 "cardinality of finiteState_Out state";
  parameter Real[nOut] resetValue=zeros(nOut)
    "chosen reset value for finiteState_Out state";

equation
  reset = resetValue;

end Transition_custRes;
```

Figure 4: Modelica code of the *Transition* component.

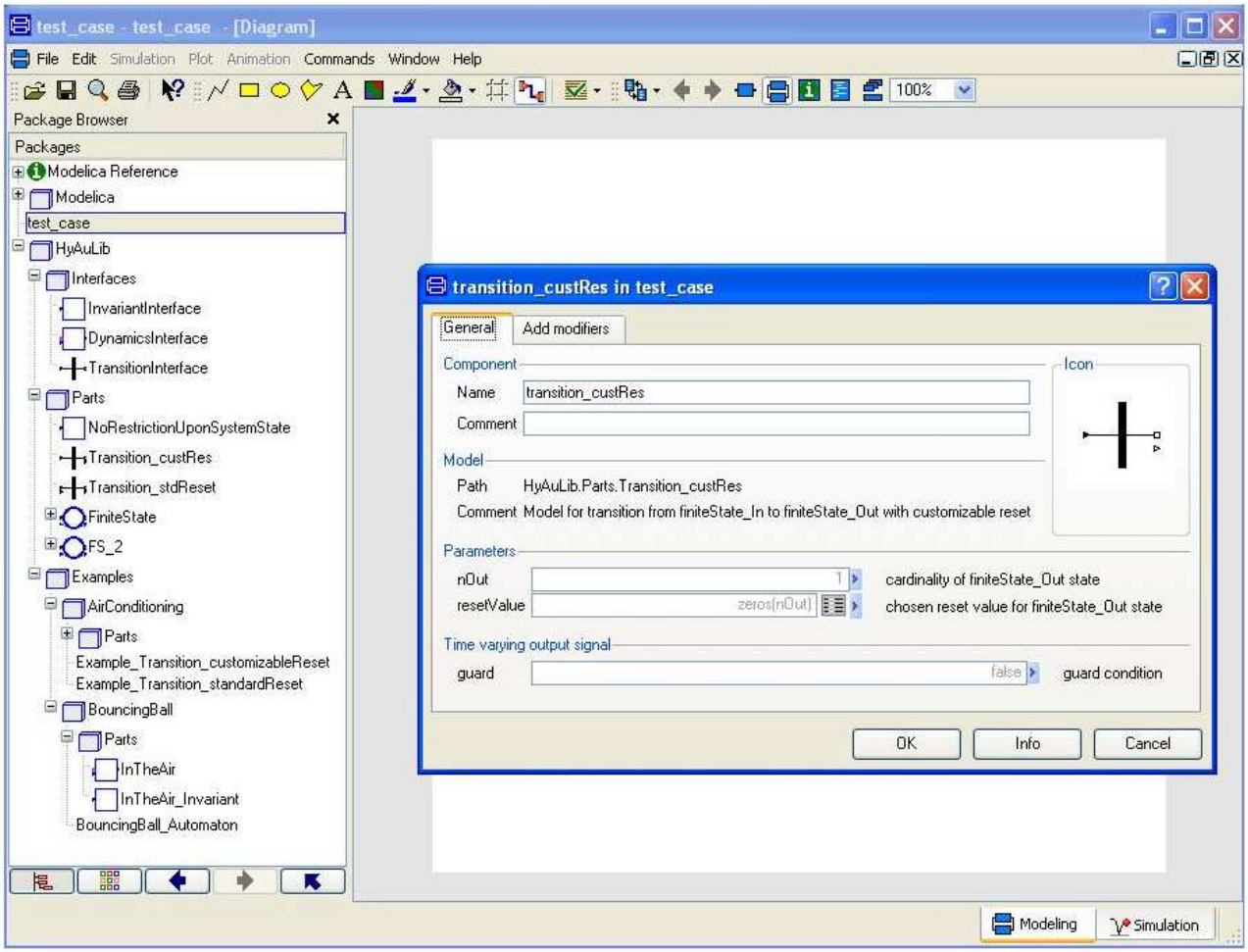

Figure 5: User's selection mask for the HyAuLib library's *Transition* component.

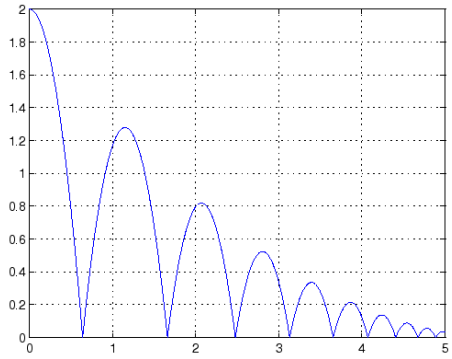

Figure 6: Vertical position of the bouncing ball vs. time. HyAuLib model.

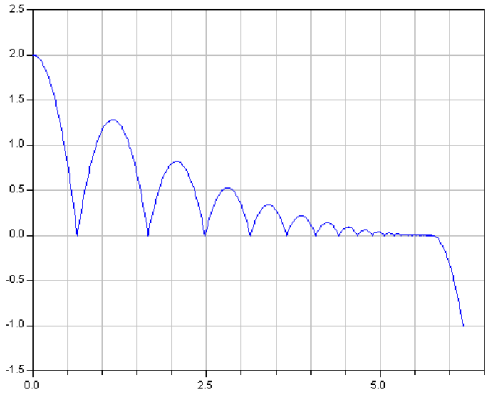

Figure 7: Vertical position of the bouncing ball vs. time. Modelica flat code.

Figure 6 shows the evolution in time of the ball position, given an initial height of 2 meters, a null initial velocity and a damping coefficient $c = 0.8$.

Both a Simulink/Stateflow and a Modelica flat model for the bouncing ball were realized, to serve as a reference for a discussion about the HyAuLib modelling performance. Both models (provided that the zero-crossing block is used in Simulink) provide good accuracy as long as the ball's energy is sufficiently large. Anyway, the *Zeno behavior* typical of this example, cause a severe impair of performance when the ball's vertical position x_1 gets very small. Due to numerical errors, x_1 will eventually become negative and, since the equations used to describe the model are still satisfied, the ball position will keep decreasing. This behavior, depicted in Figure 7, corresponds to the ball passing through the floor and keep falling, and is of course not admissible. This problem is naturally avoided if the HyAuLib's components are used, since the specification of the hybrid automaton invariant set clearly marks negative values for x_1 as unfeasible.

4.2 Air conditioning system

Let's now consider the problem of designing a room air conditioning system. We want to keep the room temperature within a specified range by acting upon a heating device. Assume that the desired temperature is 19 degrees centigrade, and the thermostat policy is to turn the heater on whenever the room temperature drops below 17°C, and turn it off when is passes 21°C. For simplicity's sake let the room temperature evolution in time be subjected to the simplified law

$$\dot{T} = -0.05\,T + 1.5\,\delta,$$

where $\delta = 0$ if the heater is turned off and $\delta = 1$ if the heater is on. The hybrid automaton will then comprise two modes and two transitions, which can be defined as follows:

1. Hybrid Automata modes:

   a) heating on ($q = ON$)
   The continuous state evolves according to

   $$\dot{T} = -0.05\,T + 1.5 \qquad (1)$$

   whereas the invariant set is $T \leq 21$.

   b) heating off ($q = OFF$)
   The continuous state evolution is given by

   $$\dot{T} = -0.05\,T \qquad (2)$$

   whereas the invariant set is $T \geq 17$.

2. Transitions:

   a) from ON to OFF
   The guard condition is $T \geq 21$,
   whereas the reset condition is $T := 21$.

   b) from OFF to ON
   The guard condition is $T \leq 17$,
   whereas the reset condition is $T := 17$.

Notice how the evolution of the continuous and discrete states of the automaton are tightly coupled. Whenever $q = ON$, the temperature rises according to (1), whereas it decays according to (2) when $q = OFF$. Likewise, the evolution in time of the discrete state is constrained by the continuous state value: it cannot jump from ON to OFF or viceversa unless the guard condition is triggered.

Figure 8 shows the evolution in time of the room air temperature and the periodic switching of the heater from power on to power off and viceversa. The air conditioning system was initialized in power off, with a room temperature of 14°C. Whenever the temperature upper bound (21°C) is reached, the heater is powered off, and the room starts cooling until the lower bound for the admissible temperature (21°C) is hit. Then, the heater is powered off and a new cycle begins.

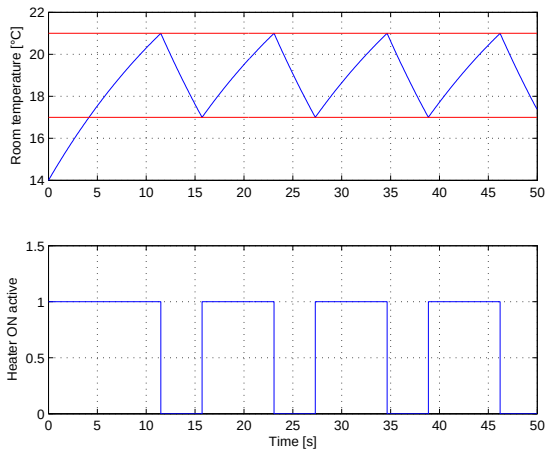

Figure 8: Room temperature and Heater ON status vs. time.

5 Concluding Remarks

In this paper the HyAuLib, a Modelica library for modelling and simulation of autonomous hybrid automata, extending the free Modelica StateGraph library for Finite State Machines, has been presented, and its main features illustrated throughout the simulation of two classic hybrid system textbook case studies. The HyAuLib allows, even to the most unexperienced user, to derive in a natural way models for simulating complex hybrid systems. Future developments of the HyAuLib will comprise the inclusion of a probabilistic approach with respect to the transition occurrence and the exploration of the library's capabilities and efficiency in modelling complex applications in the system safety design area.

References

[1] R. Alur, R. Grosu, Y. Hur, V. Kumar, I. Lee. Modular Specification of Hybrid Systems in Charon. Technical Memorandum, http://www.cis.upenn.edu, University of Pennsylvania, Philadelphia, PA.

[2] M. Anderson. Object-Oriented Modeling and Simulation of Hybrid Systems. Ph.D thesis, Lund Institute of Technology, Lund, Sweden, December 1994.

[3] C. Brooks, A. Cataldo, E. A. Lee, J. Liu, X. Liu, S. Neuendorffer, H. Zheng. HyVisual: A Hybrid System Visual Modeler, Technical Memorandum, UCB/ERL M05/ 24, http://ptolemy.eecs.berkeley.edu/ publications /papers/05, University of California, Berkeley, CA 94720, 2005.

[4] L. Carloni, M. D. Di Benedetto, R. Passerone, A. Pinto, A. Sangiovanni-Vincentelli. Modeling Techniques, Programming Languages and Design Toolsets for Hybrid Systems. Technical report, http://www.columbus.gr/documents/ public/WPHS, 2002.

[5] A. Deshpande, A. Gollu, and L. Semenzato. The SHIFT Programming Language for Dynamic Networks of Hybrid Automata. IEEE Transactions on Automatic Control, 43 (4): 584-587, April 1998.

[6] P. Fritzson, and P. Bunus, Modelica - a general object-oriented language for continuous and discrete-event system modelling and simulation. In Proceedings of the 35th IEEE Annual Simulation Symposium, San Diego, CA, 2002.

[7] J. Lygeros. Lecture Notes on Hybrid Systems. Rio, Patras, Greece: Internal report, Department of Electrical and Computer Engineering University of Patras, 2004.

[8] S. E. Mattsson, M. Otter, and H. Elmqvist. Modelica Hybrid Modeling and efficient simulation. In IEEE Conference on Decision and Control, Phoenix, AZ, 1999.

[9] Modelica Association, Modelica - a unified object-oriented language for physical systems modelling. Language specification. Technical report, http://www.modelica.org, 2002.

[10] M. Otter, K.E. Arzen, I. Dressler. StateGraph - A Modelica Library for Hierarchical State Machines. In Proceedings of the 4th International Modelica Conference, Hamburg, Germany, pp.569-578, 2005.