Proceedings 5th MATHMOD Vienna, February 2006 (I.Troch, F.Breitenecker, eds.)

INTERCHANGE SEMANTICS FOR HYBRID SYSTEM MODELS

Alessandro Pinto[1], Luca P. Carloni[2], Roberto Passerone[3], and Alberto Sangiovanni-Vincentelli[1]

[1] Department of Electrical Engineering and Computer Sciences
University of California at Berkeley, Berkeley, CA 94720
apinto,alberto@eecs.berkeley.edu
[2] Department of Computer Science
Columbia University in the City of New York, NY 10027-7003
luca@cs.columbia.edu
[3] University of Trento, Trento, Italy
roby@dit.unitn.it

Abstract. We propose an interchange format for hybrid systems to allow tool interoperability and seamless exchange of information among the hybrid system research community. The need of an interchange format is clear as the tools that have been developed so far use different modeling assumptions and semantics. In our approach, we focus on the semantics of the interchange format as we believe that only using a well-defined semantics will allow developing translation mechanisms from one tool to another that have guaranteed correctness properties. A unique, precise model of computation would make it impossible to support a variety of models that can be radically different. Hence, we use an "abstract semantics" in the sense that it can be refined to yield any model of computation, or "concrete semantics", which, in turn, is associated to the existing languages that are used to specify hybrid systems. We show how leveraging its abstract semantics, the interchange format can be used to capture the essential information across different modeling approaches and how such information can be effectively used in the translation process.

1 Introduction

Hybrid systems [1–3] have proven to be powerful design representations for system-level design in particular for embedded controllers. The term *hybrid* refers to the use of multiple models of computation in a unified framework. Often, hybrid refers to a mix of continuous dynamical systems and finite-state machines even though compositions of heterogeneous systems may be defined in larger semantic domains. The needs for a way of mixing and matching different tools is very much felt because of the relative novelty of this design representation and of the immaturity of the tools available today. There are two camps in the community who deals with hybrid systems: one would prefer to define a common model of computation for hybrid systems that should be used uniformly across different tools, the other pushes for an *interchange format*, i.e., a file, or a set of files, which contains data in a given syntax that is understood by different interacting tools. An interchange format is not a database nor a data structure, but a simpler object whose goal is to foster the exchange of data among different tools and research groups. Of course, the approach fostered by the first group has innumerable advantages but it faces an uphill battle with respect to the existing tool vendors or providers because embracing a common model of computation would require a substantial re-write of their tools. The second approach could be strengthened by providing rigorous semantics to the interchange format, thereby allowing a formal analysis of the properties of the translation between different hybrid models. And this is the path we decided to follow.

To motivate our views, we offer some considerations about interchange formats that are the result of our experience in the field of Electronic Design Automation (EDA) and of a long history in participating to the formation of standard languages and models for hardware design as well as of Columbus [4], a research project supported by the European Community that spearheaded collaboration across the ocean between European and US research groups.

Interchange Formats in EDA: History of EDIF. In the early 80s, after the advent of automated chip design, the EDA community observed a proliferation of tools from different companies and for different purposes. Given the immaturity of the field, and driven by the necessity of maintaining the market share, each company based its set of tools on a proprietary representation whose meaning was not known to other companies. The inability of the customers to exchange design data between tools became a severe limitation to the possibility of integrating design flows. On the other hand, tools integration became a need since EDA companies specialized their products to target specific problems like simulation, verification, synthesis, place and route etc.

In 1983, representatives of the major EDA companies and of the University of California at Berkeley formed the Electronic Design Interchange Format (EDIF) Steering Committee with the intent of defining a standard format for interchanging design information across EDA tools. Once the interchange format had been defined, each company started developing translators to write and read designs. Besides limitations in the expressiveness of the chosen syntax, the main problem with the early versions of EDIF was the ambiguity of the language whose free interpretation lead to the definition of many flavors of the same standard. The meaning of an EDIF description was indeed **encoded in the translators**. In order to solve this problem, the EDIF Committee realized that the such ambiguities had to be ruled out by giving a more precise semantics to EDIF. This is why, in the latest version of the interchange format, an information model is attached to a description. The information model is described in the formal language EXPRESS and has a formally defined semantics.

Other Interchange Formats: LEF, DEF and BLIF. The Library Exchange Format/Design Exchange Format (LEF/DEF) were defined by Cadence Design Systems to exchange data across synthesis and layout tools. These formats have been recently made publicly available as part of the Open Access initiative. The company is providing also a C++ application programming interface (API) that can be used to interface tools based on these formats and, that ultimately offer a unique semantic interpretation of these formats. The Berkeley Logic Interchange Format (BLIF) is a hardware description language for the hierarchical description of sequential circuits which serves as an interchange format for synthesis and verification tools. The BLIF language has a very precise semantics which can be used to define the implementation of finite state machine in terms of latches and combinational logic.

HSIF: Hybrid Systems Interchange Format. The *Hybrid Systems Interchange Format* (HSIF) has been developed by G. Karsai, R. Alur and colleagues at Vanderbilt University and the University of Pennsylvania. HSIF models represent a system as a network of hybrid automata. In contrast to interchange formats that only define their syntax, HSIF imposes a model with a formal concrete semantics. An HSIF model is a network of hybrid automata defined as a tuple (HA, V, P, C), where HA is a set of hybrid automata, V is a set of variables, P is a set of parameters, and C is an input constraint. A hybrid automaton is defined in terms of discrete states and flows. The semantics of an HSIF model is defined in terms of valuations of the system variables. Each automaton evolves through a series of continuous and discrete steps. In the same way, the execution of a network is a series of continuous and discrete steps that are defined in terms of the interconnected automata. The philosophy behind HSIF is to foster a precise concrete semantics for hybrid systems. As a result, a model described in a source language can be translated into an equivalent HSIF description only if its semantics adheres to the one defined by the interchange format. This approach has two limitations: (1) existing tools that do not match the HSIF semantics cannot be used in synergy with other tools that match it; (2), the interchange format forces new tools to base the semantics of their design capture language to a particular choice. As a concrete example, HSIF does not support hierarchy while many hybrid system description languages do. Also, algebraic loops are forbidden in HSIF, but many languages (e.g. languages that allow the implicit description of differential algebraic systems of equations) support such specifications.

A new Interchange Format for Hybrid Systems. We combine our experience in past interchange formats for EDA and the lecture of HSIF on the importance of giving a semantics to an interchange format. However, we define a novel interchange format for hybrid systems that uses an abstract semantics to be able to express models with different semantics.

In particular, in designing our interchange format, we believe it should:

- support all existing tools, modeling approaches and languages in a coherent global view of the applications and of the theory;
- be open, i.e., be available to the entire community at no cost and with full documentation;
- support a variety of export and import mechanisms;
- support hierarchy and object orientation (compact representation, entry error prevention).

By having these fundamental properties, an interchange format can become the formal backbone for the development of sound design methodologies through the assembly of various tools. The process of moving from the design representation used by tool A to the one used by tool B is structured in two steps: first, a representation in the standard interchange format is derived from the design entry that is used by A, then a preprocessing step is applied to produce the design entry on which B can operate. Notice that tool B may not need all the information on the design that were used by A and, as it operates on the design, it may very well produce new data that will be written into the interchange format but that will not ever be used by A. Naturally, the semantics of the interchange format must be rich enough to capture and

Name	Automata Definition	State-to-Dynamics Mapping	Supported Dynamics	Guards	Invariants	Reset Maps	Hierarchy	CT/DT interface
Simulink/Stateflow	Stateflow and Simulink switches	Stateflow output selecting state evolution	No limitations	Conditions on Stateflow inputs and threshold crossing detector	Not supported	Integrator's reset from Stateflow output	Yes	Stateflow outputs acting on Simulink blocks
Modelica	Not explicitly defined	Events enabling equations	No limitations	Triggering relations on variables (`when` statement)	Not an explicit language feature	Through `reinit` statement	Yes	Events enabling equations
HyVisual	Explicit finite state machine representation	Discrete-state refinement	No restrictions	Triggering conditions on state variables	Not supported	Assignment on the FSM edges	Yes	States refined into dynamical systems and special conversion blocks
Scicos	Not explicitly defined	Events switching dynamics	No restrictions	Threshold detectors	Threshold detectors	Reinitialization of integrators' state	Yes	Discrete states affecting continuous states
Shift	Textual definition of locations and transitions	Flows as locations' arguments	No Restrictions	Conditions on system variables	Conditions on system variables	Assignment statements	Yes	Location associated with flows and reset maps
Charon	Mode compositions and refinement	Differential and algebraic constraints inside modes	No restrictions	Enabling conditions on system variables	Constraints on system variables	Assignment statements	Yes	Modes defining differential and algebraic constraints and reset maps
HyTech	Explicit declaration of locations and transitions	Flows defined in each location	Convex predicate over the derivative of state variables	Conjunction of linear constraints	Convex predicate over state variables	Assignment statements	No	Locations associated with flows and reset maps
CheckMate	Stateflow	Mode selector from Stateflow to a set of dynamics	Linear or nonlinear (simulation only or approximation to linear dynamics)	Affine inequalities	Not supported	Affine maps	No	Mode selectors switching dynamics and affine reset maps
d/dt	Explicit declaration of locations and transitions	Flows defined in each location	Linear	Convex polyhedra	Convex polyhedra	Not supported in the version shipped to us	No	Location associated with flows
Hysdel	Logic formulas on Boolean variables	Mode selectors	Discrete Time and Linear	Threshold conditions on system variables	Not supported	Modeled as one step dynamics	No	Mode selectors switching dynamics

Table 1. Comparing the modeling approaches: modeling the basic hybrid system structure.

"protect" the different properties of the design at the various stages of the design process. This guarantees that there will be no loss going from one design environment to another due to the interchange format itself. The format is indeed a *neutral go-between*.

We believe that no approach is mature enough today to recommend its general adoption (see [5] for a detailed discussion on how the present models do not satisfy important criteria). With this paper we attempt to give the foundations of a standard interchange format as well as a standard design capture language where semantics is favored over syntax.

2 Review of Hybrid System Tools

In this section we give a comparative summary of a representative set of design approaches, languages, and tools for hybrid systems. These considerations are based on a comprehensive study that we completed as part of the of the Columbus project [4]. An important conclusion of our analysis is that no single tool covers all the needs of designers that use hybrid systems as models to solve their problems. While being able to capture the behavior of the system under study in an intuitive and compact way and to simulate it is an important feature for any design framework, formal analysis and synthesis tools have a much higher potential in delivering a substantial productivity gain and error-free designs. These tools rely upon abstraction and hierarchy to solve industrial-strength problems. The choice of abstraction levels and of decompositions into parts is not unique and it is rare that a designer can find the right solution at the first try. Hence, interactive environments where simulation is used to guide the selection of the appropriate abstractions and decompositions are indispensable to advance the state of the art. To build this kind of environments it is essential to provide a common ground for the different tools to integrate. When models are as complex as hybrid systems, defining this common ground is by no means trivial.

Table 1 shows the approaches adopted by each language for modeling the basic hybrid system structure. For example, while most languages provide support to describe finite state machines, discrete states cannot be clearly distinguished in Simulink/Stateflow [6, 7], Modelica [8–10] and Scicos [11, 12]. In Simulink/Stateflow the discrete automata can be described using a Stateflow chart but it is also possible to use Simulink blocks to encode state. Modelica does not define locations and transitions. It is up to the user to define discrete states and derive a finite state machine using the statements that the language provides.

Another basic feature is the association of a dynamical system to a specific state of the hybrid automaton. HyVisual [13] and Charon [14–16] have perhaps the most intuitive syntax and semantics for this

purpose. In HyVisual a state of the hybrid automaton can be refined into a continuous time system. Charon allows a mode to be described by a set of algebraic and differential equations. In CheckMate [17], Simulink, and Hysdel [18, 19] a hybrid system is modeled as two main blocks: a state machine and a set of dynamical systems. The automaton is described by a finite state machine where a transition can be triggered by an event coming from a particular event-generation block that monitors the values of the variables of the dynamical system. On the other hand, the finite state machine can generate events that are sent to a mode-change block whose purpose is to select a particular dynamics depending on the events.

Transitions semantics is not the same across tools. Invariants are only explicitly supported by Charon, HyTech [20–22] and d/dt [23–25], while the other tools have triggering guards semantics.

Two very important features for modeling complex systems are hierarchy and composition. Not all languages allow the composition of hybrid systems: CheckMate, d/dt and Hysdel only allow the description of a monolithic model. Even if composition is essential, these tools are used for model checking whose complexity is prohibitive already for few variables and discrete locations.

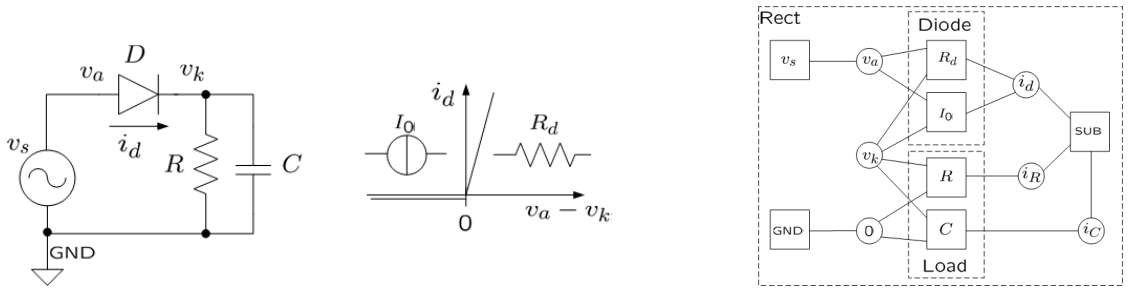
Finally, another very important feature is the possibility of modeling non-causal systems. Modelica is the only language that allows non-causal modeling.

None of the languages that we analyzed [4] has a clear definition of the semantics of programs that contain algebraic loops. All of them rely on the simulation engine while we believe that a language has to give a meaning to programs containing algebraic loops and the meaning should be independent from the simulator's engine.

All these structural and semantic differences ask for an interchange format that allows formal reasoning on the properties of a model in order to understand when a model coming from one tool can be exactly represented in another tool. The interchange format, then, must have a formal semantics. On the other hand, such semantics has to leave some flexibility to be able to represent many different models. This is why we define an *abstract semantics* that can be refined in the *concrete semantics* of the different tools by determining a set of key features. The definition of the abstract semantics is the subject of the next section.

3 Interchange Format Syntax

Notation Basics. For a tuple $W = (w_1, ..., w_n)$, we denote the component w_i of W with $W.w_i$. Given a variable with name v, its value is denoted by $val(v)$ where val is a valuation function. If V is the tuple $(v_1, ..., v_n)$ then $val(V) = (val(v_1), ..., val(v_n))$. If, instead, V is the set $\{v_1, ..., v_n\}$ then its valuation is the multi-set $val(V) = \{val(v_1), ..., val(v_n)\}$. For a set of variables V, the set of all possible valuations of V is denoted by $\mathcal{R}(V)$. Given a subset $D \subseteq \mathcal{R}(V)$ of the possible values of the set of variables V, and given another set $V' \supseteq V$, the lifting of D to V' is given by the operator $\mathcal{L}(V')(D) = \{p \in \mathcal{R}(V') : \exists p' \in \mathcal{R}(V)$ s.t. $p \supseteq p'\}$.



(a) Circuit diagram (b) Block Diagram

Fig. 1. Half-wave rectifier used as running example in this paper

Running Example. Figure 1(a) illustrates a *half-wave rectifier circuit*, a simple electronic circuit that can be modeled as a hybrid system and that will be used as a running example throughout the paper to illustrate the proposed interchange format. In particular, we model the diode by dividing the voltage across its endpoints in two regions of operation: if $v_a - v_k < 0$ the diode behaves as a constant current source of value $-I_0$; if $v_a - v_k \geq 0$ the diode behaves like a resistor of value R_d. The half-wave rectifier can be "structurally" represented by the block diagram in Figure 1(b). The three currents i_d, i_R and i_C must satisfy the Kirchoff's current law that states that the sum of all currents of components attached to the same node is equal to zero. This constraint is implemented by the block SUB in Figure 1(a).

Syntax. With the term syntax we refer to the language constructs that are provided by the interchange format to express hybrid systems. Our definitions are based on sets and functions that have a direct connection to the syntax defined in [5]. To simplify our notation, and without loss of generality, all components in our model are already instantiated and unique. The introduction of renaming functions and instantiation is straightforward in this context. We describe the syntax of a hybrid system as a tuple $H = (V, E, \mathcal{D}, I, \sigma, \omega, \rho)$ where:

- $V = \{v_1, ..., v_n\}$ is a set of variables;
- $E = \{e_1, ..., e_m\}$ is a set of equations in the variables V. An equation e_i is of the form $l(V) = r(V)$ (or equivalently $l(V) = 0$) where $l(V)$ and $r(V)$ are expressions;
- $\mathcal{D} \subseteq 2^{\mathcal{R}(V)}$ is a set of domains, or regions, of the possible valuations of the variables V;
- $I \subseteq \mathbb{N}$ is a set of indexes. The index set is used to capture the distinct dynamics of a hybrid system. Its precise role is explained in detail later when we discuss the composition of hybrid systems;
- $\sigma : 2^{\mathcal{R}(V)} \rightarrow 2^I$ is a function that associates a set of indexes to each domain and such that $\sigma(D) = \emptyset$ if $D \notin \mathcal{D}$;
- $\omega : I \rightarrow 2^E$ is a function that associates a set of equations to each index;
- $\rho : 2^{\mathcal{R}(V)} \times 2^{\mathcal{R}(V)} \times \mathcal{R}(V) \rightarrow 2^{\mathcal{R}(V)}$ is a function to reset the values of the variables (after a transition between two domains has happened) and such that $\rho(D_1, D_2, val(v)) = \emptyset$ if $D_1 \notin \mathcal{D} \vee D_2 \notin \mathcal{D}$.

A hybrid system is characterized by a set of variables that are related by equations. The dynamics of a hybrid system, i.e. the system of differential and algebraic equations that determine its continuous-time evolution, depends on the values of the variables, and can change over time. This behavior is captured by the two functions σ and ω. For each domain, σ provides a set of indexes J. The union $\cup_{i \in J} \omega(i)$ is the set of equations that are active in that domain.

Example 1. The `Load` component instantiated in the `Rect` component of Figure 1(a) is a hybrid system such that $V = \{v_R, v_C, i_R, i_C, v_k, i_d\}$, $E = \{v_R = v_C, v_R = v_k, i_C + i_R = i_d, i_R = v_R/R, i_C = C\dot{v}_C\}$, $\mathcal{D} = \{\mathbb{R}^6\}$, $I = \{1\}$, $\sigma(\mathbb{R}^6) = \{1\}$, $\omega(1) = E$. The reset function ρ acts as the identity on the values of the variables V: $\rho(\mathbb{R}^6, \mathbb{R}^6, val(V)) = val(V)$. □

In the previous example, a continuous time system is described as a hybrid system with one domain, where all equations are active, and a trivial reset map. The following example shows a system with two domains and a more elaborated reset map.

Example 2. A bouncing ball is a hybrid system whose dynamics is described by two variables: the vertical position y and the vertical velocity v. Every time the ball touches the ground, the sign of the velocity is reversed and the value is scaled by a factor called the restitution factor, and denoted by ϵ, that accounts for the energy loss due to the impact. A bouncing ball can be modeled as a hybrid system with $V = \{y, v\}$, $E = \{\dot{v} = -g, \dot{y} = v\}$. The set of possible valuations of the variables V is partitioned in two subsets: $D_1 = \{\{val(y), val(v)\} : val(y) \leq 0 \wedge val(v) < 0\}$ and $D_2 = \overline{D_1} = \{\{val(y), val(v)\} : val(y) > 0 \vee val(v) \geq 0\}$, hence $\mathcal{D} = \{D_1, D_2\}$; $I = \{1\}$, $\sigma(D_1) = \sigma(D_2) = \{1\}$, $\omega(1) = E$. The reset function is defined as follows: $\rho(D_2, D_1, val(V)) = \{val(y), -\epsilon val(v)\}$ and $\rho(D_1, D_2, val(V)) = \{val(y), val(v)\}$. □

Both these examples show hybrid systems where the index set is a singleton. The reason is that the dynamics of the hybrid system is the same in each domain. Hybrid systems for which the dynamics changes depending on the domain, or hybrid systems resulting from the composition of other hybrid systems, will have non-singleton index set.

Before defining the composition of hybrid systems, we extend the hybrid system tuple by adding two more elements: a set of temporary variables V_t, which store the intermediate results of a computation, and a function $\pi : E \rightarrow \{1, 2, \ldots, |E|\}$ that fixes an order on the set of equations [4]. Hence, the tuple denoting a hybrid system that was defined in the previous section is extended as follows: $H = (V, V_t, E, \mathcal{D}, I, \sigma, \omega, \rho, \pi)$.

Composition of hybrid systems. Given two hybrid systems $H_1 = (V_1, V_{t1}, E_1, \mathcal{D}_1, I_1, \sigma_1, \omega_1, \rho_1, \pi_1)$ and $H_2 = (V_2, V_{t2}, E_2, \mathcal{D}_2, I_2, \sigma_2, \omega_2, \rho_2, \pi_2)$, we define their composition as a new hybrid system $H = H_1 || H_2$ such that:

- the variable, equation and domain sets are the union of the corresponding sets of the two hybrid systems H_1 and H_2:

$$V = V_1 \cup V_2,\ V_t = V_{t1} \cup V_{t2},\ E = E_1 \cup E_2,\ \mathcal{D} = \mathcal{L}(V)(\mathcal{D}_1) \cup \mathcal{L}(V)(\mathcal{D}_2)$$

  where domains are lifted as the new set of variables contains V_1 and V_2;

[4] Note that π is not necessarily an injective function. For instance, for languages like MODELICA that do not define any specific equation ordering all the equations are mapped to the same integer.

- the index set is the juxtaposition of the two index sets $I = \{1, ..., |I_1| + |I_2|\}$;
- for a given domain, the set of enabled dynamics (which is a subset of the index set) is the union of the sets of enabled dynamics of H_1 and H_2: $\forall D \in 2^{\mathcal{R}(V)},\ \sigma(D) = \sigma_1(D|_{V_1}) \cup (\sigma_2 + |I_1| + 1)(D|_{V_2})$ where $(\sigma + k)(D) = \{n + k : n \in \sigma(D)\}$ is a shifting of the indexes;
- for each given index, the set of equations associated with it (and, therefore, the set of equations associated with the dynamics denoted by that index) is the same as in H_1 and H_2 (after a suitable shifting of the indexes):

$$\omega(i) = \omega_1(i),\ 1 \leq i \leq |I_1|,$$
$$\omega(i) = \omega_2(i - |I_1|),\ |I_1| + 1 \leq i \leq |I_1| + |I_2|$$

- the equations order is directly derived form the orders in H_1 and H_2. The new order must preserve the original order within the two sets E_1 and E_2 such that equations in E_1 precede equations in E_2:

$$\pi(e) = \begin{cases} \pi_1(e) & if\ e \in E_1 \\ \pi_2(e) + |I_2| + 1 & if\ e \in E_2 \end{cases}$$

- given the two reset functions ρ_1 and ρ_2 and given $D_i,\ D_j \in 2^{\mathcal{R}(V)}$:

$$\rho(D_i, D_j, val(V)) = \mathcal{L}(V)(\rho_1(D_i|_{V_1}, D_j|_{V_1}, val(V_1)) \cup \mathcal{L}(V)(\rho_2(D_i|_{V_2}, D_j|_{V_1}, val(V_2))$$

The composition of hybrid systems is associative but it is not commutative because the equation ordering depends on the position of the hybrid systems in the composition. The n-ary composition of n hybrid systems $H_1, ..., H_n$ is another hybrid system $H = H_1||...||H_n = (((H_1||H_2)||H_3)||...||H_n)$.

Example 3. We model here the diode of Figure 1(a). Resistor R_d is a hybrid system such that $R_d.V = \{v_a, v_k, i_d\}$, $R_d.E = \{e_1\} = \{i_d = (v_a - v_k)/r_d\}$, $D_1 = \{p \in \mathcal{R}(R_d.V) : val(v_a) - val(v_k) \geq 0\}$ and $R_d.\mathcal{D} = \{D_1\}$, $R_d.I = \{1\}$, $R_d.\sigma(D_1) = \{1\}$, $\omega(1) = R_d.E$, $\pi(e_1) = 1$ and $R_d.\rho$ acts as the identity the values of the variables.

The current source I_d is a hybrid system such that $I_d.V = \{v_a, v_k, i_d\}$, $I_d.E = \{e_2\} = \{i_d = -I_0\}$, $D_2 = \{p \in \mathcal{R}(I_d.V) : val(v_a) - val(v_k) < 0\}$ and $I_d.\mathcal{D} = \{D_2\}$, $I_d.I = \{1\}$, $I_d.\sigma(D_1) = \{1\}$, $\omega(1) = I_d.E$, $\pi(e_2) = 1$ and $I_d.\rho$ acts as the identity on the values of the variables.

A `diode` is the parallel composition $R_d||I_d = diode$ that results in the hybrid system with the following properties: $diode.V = \{v_a, v_k, i_d\}$, $diode.E = \{e_1, e_2\}$, $diode.\mathcal{D} = \{D_1, D_2\}$, $I = \{1, 2\}$ $diode.\sigma(D_1) = \{1\}$, $diode.\sigma(D_2) = \{2\}$, $\omega(1) = e_1$, $\omega(2) = e_2$, $\pi(e_1) = 1$, $\pi(e_2) = 2$ and $diode.\rho$ acts as the identity on the values of the variables. □

In the previous example D_1 and D_2 are disjoint, therefore the ordering among the various equations is irrelevant because they will never belong to the same system of equations. If we consider the entire rectifier that is the parallel composition $rect = V_s||diode||load$, the reader can verify that such composition has three domains: the entire set of possible valuation coming from the voltage source and the load, and the two domains D_1 and D_2 defined by the diode. Moreover, equations are ordered with $V_s.E$ coming before $diode.E$ which, in turn, come before $load.E$.

4 Interchange Format Structure and Semantics

In order to keep structure and semantics well separated and also to clearly represent the hierarchical structure of a design, we partition a hybrid system into components and schedulers and we organize them into a tree that has both a structural as well as an algebraic interpretation.

A hybrid system is a pair $H = (c, s)$ where c is a component and s is a scheduler. The component is a tuple $c = (V, E, \mathcal{D})$ of variables and equations while the scheduler is a tuple $s = (I, \sigma, \omega, \rho, \pi)$. Let $\mathcal{C}$ be the set of all component instances and $\mathcal{S}$ be the set of all scheduler instances. Then, $\mathcal{I} : \mathcal{C} \rightarrow \mathcal{S}$ is a bijection that for a given a component c returns the scheduler associated with it. Note that we use instances of components and schedulers instead of objects.

The n-way composition for components and schedulers can be easily derived from the composition of hybrid systems defined in Section 3. Let $||^c$ and $||^s$ be such operations, respectively. Give two hybrid systems $H_1 = (c_1, s_1)$ and $H_2 = (c_2, s_2)$, their composition is $H = H_1||H_2 = (c_1||^c c_2, s_1||^s s_2)$.

We now consider the hierarchical structure of hybrid systems. A hybrid system structure $\mathcal{H} = (C, S)$ is a pairs where C is a rooted tree of components and S is a rooted tree of schedulers. $C = (C_N, C_E)$ where C_N is a set of components and $C_E \subset C_N \times C_N$ is a set of relations (the edges of the tree). If $r = (c_i, c_j) \in C_E$ we say that c_j is instantiated in c_i.

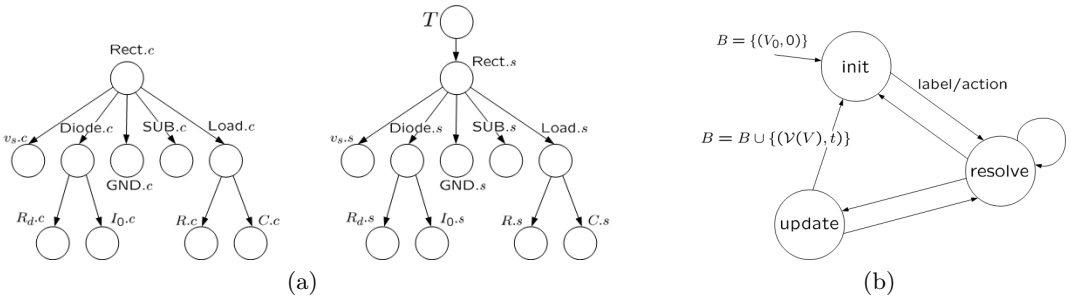


Fig. 2. a) Structural representation of the half-wave rectifier, b) time stamper finite state machine.

The tree of schedulers has the following structure: $S = (S_N, S_E)$ where S_N is a set of schedulers and $S_E \subset S_N \times S_N$ is a set of connections among schedulers. $S_N = T \cup S'_N$ where T is a special object called *time-stamper*. The subtree induced[5] by S'_N is isomorphic to C, and the isomorphism is $\mathcal{I}$. Also, if $s \in S'_N$ is the root of such induced subtree, then $(T, s) \in S_E$ and it is the only outgoing edge of T whose input degree is equal to zero. We illustrate this concept using the example in Figure 1(a). Figure 2, which shows the structure of the rectifier, has two interpretations: (1) it captures the organization of a design. For instance, component `Diode` contains component R_d and component I_0; (2) it represents the parse tree of the algebraic composition $Rect = v_s||Diode||GND||SUB||Load = v_s||(R_d||I_0)||GND||SUB||(R||C)$. Being able to capture hierarchies in a formal way is extremely important for an interchange format to retain the structure of the original specification and to allow "back translation" without loss of information.

The semantics of a hybrid system is defined by a set B of pairs (γ, t) where $\gamma \in \mathcal{R}(H.V)$ is a multi-set of possible values of the hybrid system variables and $t \in \mathbb{R}_+$ is a time stamp. The computation of the time stamps is controlled by the abstract finite state machine T (the *time stamper*), whose transition diagram is reported in Figure 2(b) [6].

Let $\mathcal{G} : S_N \rightarrow 2^{S_N}$ be a function that associates to each scheduler the set of its children, and let $\Pi : S_N \rightarrow \{1, ..., |S_N|\}$ be a global ordering of the nodes. Such ordering depends on the order in which hybrid systems are composed. Each scheduler implements three algorithms: `init`, `resolve`, and `update`.

In particular, the `resolve` algorithm is shown in Algorithm 1 and proceeds as follows: first, the set of all children of the scheduler s is computed. If s is a leaf then the active equations are selected and solved, while if s is not a leaf the recursion along the trees calls the `resolve` method on all children of s in the order specified by Π. Notice that Π together with ordering π defined in the leaves implement the ordering $H.\pi$.

Algorithm 1 `resolve` algorithm of $s \in S_N$

```
resolve(t)
children ⇐ G(s)
if children = ∅ then
  //s is a leaf, proceed to solve the equations and end recursion
  D' ⇐ {D ∈ I^-1(s).D | val(I^-1(s).V_t) ∈ D}
  J ⇐ ∪_{D∈D'} s.σ(D)
  E_t ⇐ ∪_{i∈J} s.ω(i)
  E_t ⇐ sort(E_t, s.π)
  for all e_i ∈ E_t do
    solve(e_i, t)
  end for
  markchange ( D', val(I^-1(s).V_t) )
else
  //s is not a leaf, continue the recursion
  children ⇐ sort(children, Π)
  for all s_i ∈ children do
    s_i.resolve(t)
  end for
end if
```

[5] A subgraph induced by a set of vertices of a graph G is the set of vertices together with any edge whose endpoints are both in the subset.

[6] For further details please refer to [26].

The `init` and `update` algorithms recursively call the `init` and `update` along the tree using the ordering in Π. They simply initialize variables to a given value and copy the auxiliary variable V_t into V, respectively. The set B is initialized with a pair $(V_0, 0)$ representing the initial condition of the hybrid system H. In the initial state `init` the time stamper T invokes the initialization of H. This is carried out by executing the `init` algorithm. In the `resolve` state, T invokes the execution of the `resolve` algorithm that produces a valuation of all the variables of H. Finally, in the `update` state, T invokes the execution of the `update` algorithm and adds a new pair (γ, t) to the set B.

The interchange format has a very general way of specifying states and dynamics (through indexes and sets of equations associated with them) that can capture the way in which the same objects are specified in all the tools reviewed in Section 2. There are no limitations on the nature of the equations that can be expressed and it is possible to specifiy causality among equations.

There are some key features on which it is possible to operate in order to obtain a concrete semantics. The time stamper automata determines whether a new pair (γ, t) belongs to the set of behaviors of the hybrid systems, hence it decides if the time-stamp should be reduced or can be increased. The `markchange` function is used to implement different transition semantics. The `solve` function is used to specify a method for the solution of a system of differential and algebraic equations. Since it is not in general easy to specify these functions in an analyzable way, we foresee the development of libraries of standard algorithms.

Back-tracking and Algebraic Loops. A time stamper can invoke the `resolve` algorithm of a hybrid system multiple times. It is also possible to re-initialize the system before updating the values of the variables. Such iterations can be used for back-traking or to reach a fixed-point in case of algebraic loops. Many iterations are also required for event detection. This is the main reason for having auxiliary variables and separating the resolution step from the update step as it is also defined by the *stateful* firing abstract semantics of Ptolemy [27].

5 Applications

The structure of the interchange format introduced in Section 4 and its abstract semantics are very effective in 1) representing models coming from different languages, 2) developing algorithms for the translation of models to and from different tools and 3) understanding the concrete semantics of different languages for hybrid systems.

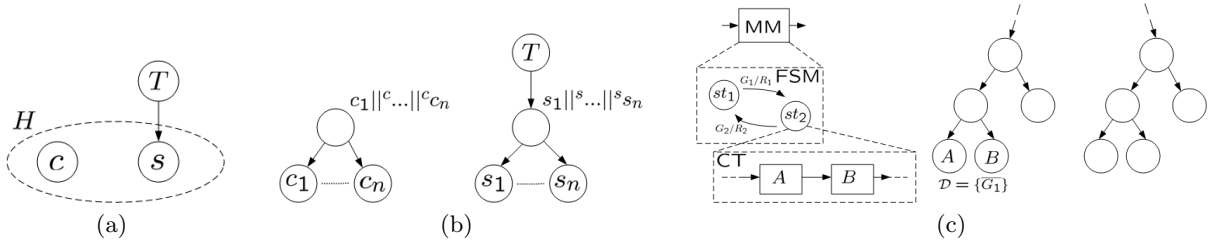


Fig. 3. Structure of the programs that do not support a) hierarchy and composition, b) hierarchy. c) Structure of a HyVisual modal model.

Figure 3(a) shows the structure of a language that does not support neither hierarchy nor composition. Examples of languages belonging to this class are CheckMate [17], `d/dt` [28], and Hysdel [19]. The tree of components has only one node which is the entire hybrid system described as a single monolithic component. In CheckMate, c is a switched dynamical system and a set of linear inequality that defines the domains implemented in Simulink. The scheduler is implemented by a Stateflow chart and the time stamper is provided by the Simulink solvers.

Figure 3(b) shows the structure of programs that support composition but not hierarchy. Examples of languages belonging to this class are HyTech [22] and HSIF [29]. Each child of the root node is a hybrid system. For HSIF programs, hybrid automata are ordered with respect to a dependency graph. Such graph has nodes that are hybrid automata and there is an edge $H_i \rightarrow H_j$ if an output of H_j is used in some equation, invariant, guard or assignment of H_j. The dependency graph, which is required to be acyclic, can be used to order the automata. Moreover, differential equations precede algebraic equations in the order.

Figure 3(c) shows the structure of a HyVisual *modal model* [30]. A modal model is described by a state machine with guards and reset maps on the edges. Each state of the state machine is refined into a continuous time system that is an interconnection of continuous time actors. The topological sort of the actor graph gives their order of execution. Also, since guards have a triggering semantics, a transition must be taken as soon as a guard is verified (i.e., there is a domain change as soon as the values of the variables fall outside a domain). Modal models can be connected together as indicated by the dotted lines in Figure 3(c). Charon [16] programs lead to a similar structure but guard conditions have different enabling semantics: these impact the way in which the time stamper processes the `domainchange` condition in order to decides whether a pair (val, t) is valid or not.

The interchange of models between simulation tools like HyVisual or Modelica and verification tools like CheckMate, requires to check several conditions. First, the pair (C, S) of component and scheduler trees must be compacted into only three nodes: one component, one scheduler and a time stamper. This implies the explicit computation of the parallel composition defined in Section 3. Second, the domains must be defined as intersection of polyhedra. The inverse translation leaves many choices among which, the most natural would be to have a root node connected to as many dynamical systems as there are domains in the original CheckMate model.

The interchange format representation also highlights the semantic and structural properties of each language like scheduling decisions, transition semantics, composition, representation of discrete and continuous dynamics interaction, hierarchy and solution methods. Some of these properties could be unspecified or not supported in a particular language and such information is directly reflected in the interchange format. Hierarchy is one example that we have already discussed. Ordering of equations and scheduling of hybrid systems is another good example. For instance, Modelica does not define how a system of differential and algebraic equations is sorted and solved. A Modelica model represented in the interchange format would have $\pi(e) = 1,\ \forall e \in H.E$. The translation of such model to HSIF would require first the reduction of the tree representation to a one-level tree and then the decision on how automata and equations are ordered. On the other hand, the inverse translation would disregard such order.

6 Conclusions

We presented an interchange format for hybrid systems that facilitates the interoperability of tools and the exchange of design informations for the hybrid system research community. Interchange formats are especially needed when the maturity of a field is such that no standard for data format has yet emerged. To do so in the hybrid system domain, the interchange format has to be "rich" in the sense that has to be able to support the large variety of semantics embedded in the tools that we are interested in connecting. In this particular case, where the semantics of the models used by the various tools can be very complex, the interchange format itself must "contain" the semantics of the tools it is supposed to support. For this reason, we base our approach on an abstract semantics that serves as the foundation of our interchange format for hybrid system design. In particular, we discussed in this paper an abstract semantics for the interchange format that we first proposed in [5].

The abstract semantics can in fact be refined into various concrete semantics, each capturing the model used by a different language for the specification of hybrid systems. We also showed how a structural representation that keeps semantics and structure clearly separated is effective in highlighting the differences among the various hybrid system languages. We illustrated the use of the abstract semantics and its structural representation by applying them to various existing languages. We implemented the proposed interchange format within the Metropolis framework and we verified with a simple example the viability of our approach. In particular, thanks to its modularity, this approach makes it possible not only to translate the model of an hybrid system from one language to another, but also to combine models written in different languages.

References

1. Alur, R., Courcoubetis, C., Halbwachs, N., Henzinger, T.A., H, P., Nicollin, X., Sifakis, J., Yovine, S.: The algorithmic analysis of hybrid systems. Theoretical Computer Science **138** (1995) 3–34
2. Henzinger, T.A.: The theory of hybrid automata. In: Logic in Computer Science, IEEE Computer Society Press (1996) 278–292
3. Maler, O., Manna, Z., Pnueli, A.: From timed to hybrid systems. In: Real-Time: Theory in Practice, REX Workshop. Volume 600 of Lecture Notes in Computer Science., Springer-Verlag (1991) 447–484
4. Carloni, L.P., Benedetto, M.D., Passerone, R., Pinto, A., Sangiovanni-Vincentelli, A.: Modeling techniques, programming languages and design toolsets for hybrid systems. Technical report, IST - Columbus Project (2004) available at `www.columbus.gr/documents/public/WPHS/Columbus_DHS4_0.2.pdf`.

5. Pinto, A., Sangiovanni-Vincentelli, A.L., Carloni, L.P., Passerone, R.: Interchange formats for hybrid systems: review and proposal. In Morari, M., Thiele, L., eds.: HSCC 05: Hybrid Systems—Computation and Control. Volume 3414 of Lecture Notes in Computer Science., Springer-Verlag (2005) 526–541
6. Angermann, A., Beuschel, M., Rau, M., Wohlfarth, U.: MATLAB, Simulink, Stateflow: grundlagen, toolboxen, beispiele (MATLAB, Simulink, Stateflow: fundamentals, toolboxes, examples). Oldenbourg-Verlag (2003)
7. Rivoire, M., Ferrier, J.: MATLAB Simulink Stateflow avec des exercices d'automaticue resolus (MATLAB Simulink Stateflow with solved exercises in automatic control). Editions TECHNIP (2001)
8. Fritzson, P., Engelson, V.: Modelica - a unified object-oriented language for system modeling and simulation. In: ECCOP '98: Proc. of the 12th Eur. Conf. on Object-Oriented Programming, London, UK, Springer-Verlag (1998) 67–90
9. Tiller, M.M.: Introduction to physical modeling with Modelica. Kluwer Academic Publishers (2001)
10. Fritzson, P.: Principles of object-oriented modeling and simulation with Modelica 2.1. J. Wiley & Sons (2004)
11. Djenidi, R., Lavarenne, C., Nikoukhah, R., Sorel, Y., Steer, S.: From hybrid simulation to real-time implementation. In: ESS'99 11th European Simulation Symposium and Exhibition. (1999) 74–78
12. Nikoukhah, R., Steer, S.: SCICOS - a dynamic system builder and simulator user's guide - version 1.0. Technical Report 0207, INRIA, Rocquencourt, France (1997)
13. Hylands, C., Lee, E.A., Liu, J., Liu, X., Neuendorffer, S., Zheng, H.: Hyvisual: A hybrid system visual modeler. Technical Report UCB/ERL M03/1, UC Berkeley (2003)
14. Alur, R., Dang, T., Esposito, J., Fierro, R., Hur, Y., Ivancic, F., Kumar, V., Lee, I., Mishra, P., Pappas, G., , Sokolsky, O.: Hierarchical hybrid modeling of embedded systems. In Henzinger, T.A., Kirsch, C.M., eds.: EMSOFT 2001: Embedded Software, First International Workshop. Volume 2211 of Lecture Notes in Computer Science., Tahoe City, CA, USA, Springer-Verlag (2001)
15. Alur, R., Dang, T., Esposito, J., Hur, Y., Ivancic, F., Kumar, V., Lee, I., Mishra, P., Pappas, G.J., Sokolsky, O.: Hierarchical modeling and analysis of embedded systems. Proc. of the IEEE (2002)
16. Alur, R., Grosu, R., Hur, Y., Kumar, V., Lee, I.: Modular specification of hybrid systems in Charon. In Lynch, N., B.H, K., eds.: Proc. of the Third Intl. Work. on Hybrid Systems: Computation and Control. Volume 1790 of Lecture Notes in Computer Science., Springer-Verlag (2000) 6–19
17. Silva, B.I., Richeson, K., Krogh, B., Chutinan, A.: Modeling and verifying hybrid dynamic systems using CheckMate. In: Proc. of 4th Intl. Conf. on Automation of Mixed Processes. (2000) 323–328
18. Torrisi, F.D., Bemporad, A., Bertini, G., Hertach, P., Jost, D., Mignone, D.: Hysdel 2.0.5 - user manual. Technical report, ETH Zurich (2002)
19. Torrisi, F.D., Bemporad, A.: HYSDEL - a tool for generating computational hybrid models for analysis and synthesis problems. IEEE Transactions on Control Systems Technology **12** (2004) 235–249
20. Alur, R., Henzinger, T.A., Ho, P.H.: Automatic symbolic verification of embedded systems. IEEE Transactions on Software Engineering **22** (1996) 181–201
21. Henzinger, T., Ho, P.H., Wong-Toi, H.: A user guide to HyTeCH. In Brinksma, E., Cleaveland, W., Larsen, K., Margaria, T., Steffen, B., eds.: TACAS 95: Tools and Algorithms for the Construction and Analysis of Systems. Volume 1019 of Lecture Notes in Computer Science. Springer-Verlag (1995) 41–71
22. Henzinger, T.A., Ho, P.H., Wong-Toi, H.: HyTech: A model checker for hybrid systems. International Journal on Software Tools for Technology Transfer **1** (1997) 110–122
23. Dang, T.: Verification and synthesis of hybrid systems. PhD thesis, INPG (2000)
24. Asarin, E., Bournez, O., Dang, T., , Maler, O.: Approximate reachability analysis of piecewise linear dynamical systems. In Krogh, B., Lynch, N., eds.: HSCC 00: Hybrid Systems—Computation and Control. Volume 1790 of Lecture Notes in Computer Science., Springer-Verlag (2000) 20–31
25. Asarin, E., Bournez, O., Dang, T., Maler, O., Pnueli, A.: Effective synthesis of switching controllers for linear systems. Proceedings of the IEEE **88** (2000) 1011–1025
26. Pinto, A., Carloni, L., Passerone, R., Sangiovanni-Vincentelli, A.L.: Interchange formats for hybrid systems: Abstract semantics. In Hespanha, J.P., Tiwari, A., eds.: Proc. of the The 9th Intl. Workship on Hybrid Systems: Computation and Control (HSCC 2006), Santa Barbara, California, Springer Verlag (2006) to appear
27. Davis, J., Goel, M., Hylands, C., Kienhuis, B., Lee, E., Liu, J., Liu, X., Muliadi, L., Neuendorffer, S., Reekie, J., Smyth, N., Tsay, J., Xiong, Y.: Overview of the Ptolemy project. Technical Report UCB/ERL M99/37, Univ. of California at Berkeley (1999)
28. Asarin, E., Dang, T., Maler, O.: The d/dt tool for verification of hybrid systems. In: Proc. of the 14th Intl. Conf. on Computer-Aided Verification. (2002) 365–370
29. Group, M.: HSIF semantics (version 3, synchronous edition). Internal document, The University of Pennsylvania (August 22, 2002)
30. Brooks, C., Cataldo, A., Lee, E.A., Liu, J., Liu, X., Neuendorffer, S., Zheng, H.: Hyvisual: A hybrid system visual modeler. Technical Report UCB/ERL M04/18, UC Berkeley (2004) available at `http://ptolemy.eecs.berkeley.edu/hyvisual/`.