# Interchange Format for Hybrid Systems: Abstract Semantics

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

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

**Abstract.** In [1] we advocated the need for an interchange format for hybrid systems that enables the integration of design tools coming from many different research communities. In deriving such interchange format the main challenge is to define a language that, while presenting a particular formal semantics, remains general enough to accommodate the translation across the various modeling approaches used in the existing tools. In this paper we give a formal definition of the syntax and semantics for the proposed interchange format. In doing so, we clearly separate the structure of a hybrid system from the semantics attached to it. The semantics can be considered an "abstract semantics" in the sense that it can be refined to yield the 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 the interchange format can be used to capture the essential information across different modeling approaches and how such information can be used in the translation process.

## 1 Introduction

While the main concept behind the term *hybrid system* is commonly accepted by the control theory community and the computer science community, there is a mismatch in the interpretation of hybrid system models. The original definition of hybrid systems captures the discrete dynamics as a graph representing a state machine [2]. A function associates a continuous dynamics to each discrete state. These dynamics, which are expressed in terms of differential equations, may vary across different states. Transitions from a source state to a target state are enabled, or triggered, by the continuous evolution of the system's variables and each transition can also set the initial conditions for the system of differential equations associated with the target state. Following a *denotational* approach, control theorists use such model to complete a formal analysis of a hybrid system and derive necessary and/or sufficient conditions for its stability, safety, and reachability. Computer scientists, instead, use such model as a reference while

J. Hespanha and A. Tiwari (Eds.): HSCC 2006, LNCS 3927, pp. 491–506, 2006.

following an *operational* approach. Their main concern is to develop software programs that designers can use to simulate and verify hybrid systems. Generally, this entails the definition of a language, whose syntax defines the words and sentences that can be written in a program while its semantics defines their meaning. In fact, the language semantics should formally define the steps that an idealized computer must follow in order to produce a meaningful result while processing the program. In particular, for tools that target simulation, to define the semantics of their language corresponds to formally specify the algorithm that will produce the simulation trace. An excellent example of the definition of operational semantics of hybrid systems is given in [3].

Each language defines a *programming style* to describe hybrid systems based on its specific purpose, e.g. simulation, verification, or synthesis. Moreover, different algorithms require different data structures and the language is usually tailored to simplify the translation from the input language description to the internal data structure used by the algorithms. MODELICA, for instance, provides a language for describing systems in terms of implicit equations [4, 5]. The language is object-oriented and objects can be instantiated inside other object to model hierarchy. HYVISUAL gives a graphical syntax and a rich library of predefined actors that can be composed to model dynamical systems [6]. A hybrid system is described as a state machine in which states are refined into interconnection of continuous time actors. CHECKMATE [7], like HYSDEL [8], uses the interconnection of a state machine and a set of dynamical systems where the state machine selects one of the dynamics depending on the value of the system variables. Finally, a language also defines the class of hybrid systems that can be described. For instance, tools that target verification only allow linear dynamics and convex guards and invariants.

A system is usually described as a composition of objects. Compositionality and hierarchy are desirable features for the design of complex systems. While composing objects at the denotational level corresponds to composing functions, giving a semantically sound definition of composition in terms of a programming language is not a trivial task. The semantics of CHARON, a high-level language for modular specification of multiple, interacting hybrid systems, is indeed *compositional* in the sense that the semantics of one of its components (possibly the entire hybrid system) is entirely specified in terms of the semantics of its subcomponents [9, 10]. An interesting aspect of composition for simulation purposes is how to schedule the execution of a system across multiple interacting components. Consider, for instance, a system where component $A$ feeds two components $B$ and $C$ and, furthermore, $C$ also receives the output of $B$ as input. After executing $A$, the simulator must choose whether to execute $B$ or $C$ first. The two possible choices would likely give different simulation results.

Another interesting issue involves solving a system of differential algebraic equations. The solution is typically represented inside a computer as a finite subset of value-time pairs $(x, t)$. Since the computer resources are discrete and finite, two problems must be addressed: (1) how to select a subset that makes the result meaningful and (2) how to compute the value of $x$ at time $t$ for a

generic system of differential equations without relying on the analytical solution [3]. Further, if instead of a single equation we have a system of differential and algebraic equations, then there are many variables that must be computed and the order in which equations are evaluated becomes relevant. Finally, support for expressing algebraic equations makes things more complex due to the possibility of generating algebraic loops. In fact, some languages like Modelica do not define the meaning of an algebraic loop and leave the decision of how to compute the solution of such equations to the simulation engine. Other tools like Simulink/Stateflow and HyVisual return an error message whenever they detect the presence of algebraic loops.

*Contributions.* Researchers in industry and academia have developed several tools for the simulation, verification and synthesis of hybrid systems. In their development efforts, they had to address all the important issues mentioned above and, generally, they have made different implementation decisions. In [1] we advocated the need for an interchange format for hybrid systems that makes it possible to integrate design tools coming from many different research communities. While to define the syntax of the interchange format is an important step, and there are already interesting approaches in this direction [11], the definition of its semantics is the key to enable unambiguous translation of models across tools. In order to capture all the different models, we define an *abstract* semantics that can be refined in the *concrete* semantics of each language, we specify a set of functions that can be applied to perform such refinement, and we show the effectiveness in translating to and from the interchange format. Our approach allows us to better understand the structure of the exisisting languages for hybrid systems, to capture the semantic differences among them, and to develop algorithms for interchanging models.

## 2 Preliminaries

*Metropolis Meta-Model Interchange Format.* In [1] we reviewed a number of languages and tools for hybrid systems. Based on the outcome of our comparative summary, we highlighted the differences among tools and also a set of desirable features that a language for hybrid systems should provide. We then offered a proposal for an interchange format for hybrid systems whose formal semantics is based on the Metropolis Meta-Model [12]. The main challenge in defining an interchange format is to define a language with a formal semantics that remains general enough as it provides and easy translation path to/from all other languages of interest. Accordingly, the proposed interchange format defines *processes* for the solution of equations and *media* for communicating results among processes. The way in which the computation is performed is described in a separate view of the specification that consists of a collection of schedulers. Processes, media and schedulers can be hierarchically organized as shown in Figure 1. The hierarchy of a hybrid system has three levels: the *transition level*, *the dynamical system level* and the *equation level*. At the transition level, a scheduler (`TM`) selects a set of continuous-time processes whose composition forms

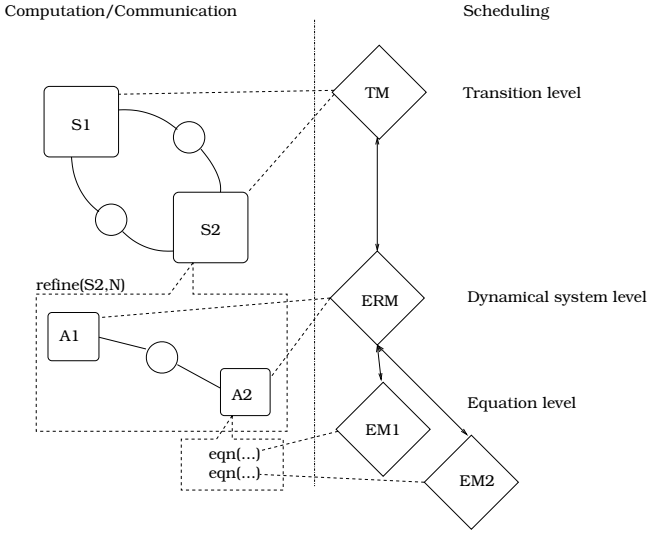


**Fig. 1.** Organization of the interchange format presented in [1]

a dynamical system. At the dynamical system level, a scheduler (`ERM`) selects a set of equations and orders their execution. At the equation level, the execution of each equation is governed by equation managers `EM`. Across the three hierarchical levels, the network of schedulers restricts the possible executions of the process network by (1) selecting a set of active processes at the transition level, (2) scheduling the execution of continuous time processes at the dynamical system level and (3) scheduling the solution of the equations at the equation level.

*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 $\mathcal{L}(V')(D) = \{p' \in \mathcal{R}(V') : p'|_V \in \mathcal{R}V\}$, where $p'|_V$ denotes the restriction of the valuation $p'$ to only the variables in $V$.

*Running Example.* The diagram in Figure 2 represents a *half-wave rectifier circuit*, a simple electronic circuit that can be modeled as a hybrid system and will be used 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 3. The three currents $i_d$, $i_R$ and $i_C$ must satisfy the Kirchoff's

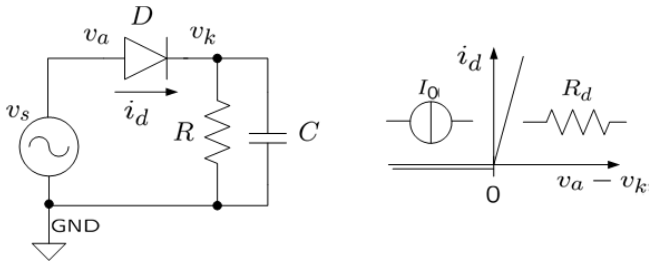


**Fig. 2.** Half-wave rectifier used as running example in this paper

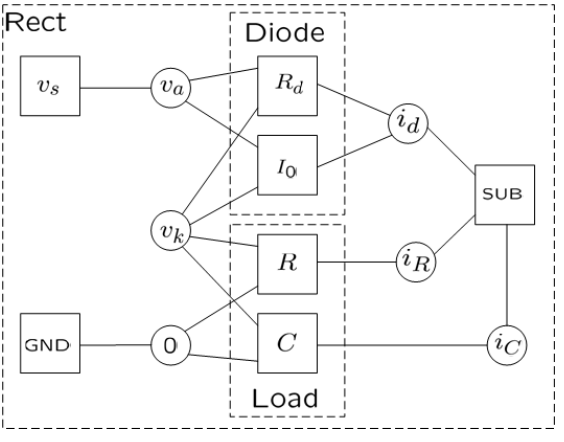


**Fig. 3.** Block diagram representing the half-wave rectifier

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 2.

## 3 Interchange Format 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 [1]. 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 $\sigma$ and $\omega$. For each domain, $\sigma$ 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. The components that define these functions can be easily identified in the interchange format structure of Figure 1. Function $\sigma$ is implemented at the transition level while function $\omega$ is implemented at the dynamical system level. The reset function describes what happens to the values of the variables when the active domain changes.

*Example 1.* The `Load` component instantiated in the `Rect` component of Figure 2 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 $\rho$ 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 $\epsilon$, 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 sets.

**Equation ordering and temporary variables.** 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 $\pi : E \to \{1, 2, \ldots, |E|\}$ that fixes an order on the set of equations[1]. Hence, the tuple denoting a hybrid system that was defined in the previous section is extended as follows: $H = (V, V_t, E, \mathcal{D}, I, \sigma, \omega, \rho, \pi)$.

Temporary variables are used in algorithms like fixed-point computation or event detection, i.e., whenever the system of equations must be solved multiple times before reaching the desired result. Also, as discussed in the introduction, an important task in solving the systems of equations is to properly order them.

**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 $V_2$;
- the index set is the juxtaposition of the two index sets

$$I = \{1, ..., |I_1| + |I_2|\}$$

  which takes into account the fact that the number of dynamics and components is equal to the sum of the number of the dynamics and components coming from the two hybrid systems $H_1$ and $H_2$;
- 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;
- the set of equations associated with each given index (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):

$$\begin{array}{ll} \omega(i) = \omega_1(i), & \text{if } 1 \le i \le |I_1|, \\ \omega(i) = \omega_2(i - |I_1|), & \text{if } |I_1| + 1 \le i \le |I_1| + |I_2| \end{array}$$

[1] Note that $\pi$ 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 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 $E_2$:

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

– the two reset functions $\rho_1$ and $\rho_2$ give a set of new possible values for the variables as a function of the domains and the variables themselves. If the two hybrid systems share the same variables and if the two reset functions assign different values for the same domain transition, then both resets should be considered. If the two reset functions agree on the resets then only one value should be considered. This operation is implemented by the set union. Given $D_i,\ D_j \in 2^{\mathcal{R}(V)}$

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

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 $H_1, \ldots, H_n$ is another hybrid system $H = H_1 \,||\, \ldots \,||\, H_n = (((H_1 \,||\, H_2) \,||\, H_3) \,||\, \ldots H_n)$.

*Example 3.* We model here the diode of Figure 2. 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 on 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. The following example, instead, is a case where the order is relevant.

*Example 4.* The entire rectifier 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 Semantics

We define the semantics of a hybrid system $H$ with a tuple $(H, B, T, \texttt{resolve}, \texttt{init}, \texttt{update})$. The set $B$ is a set of pairs $(\gamma, t)$ 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 4. Furthermore, $T$ governs the valuation of the system variables for a given time stamp. In other words, $T$ is in charge of both selecting the next time stamp and deciding whether the pair $(val, t)$ can be added to the set $B$. Both tasks are performed by $T$ through the invocation of three algorithms (`init`, `resolve` and `update`). This invocation follows a specific sequence that is encoded in the transition diagram. For different time-stamp-control methods, predicates and actions on the arcs of the abstract state machine change, while the three algorithms remain the same.

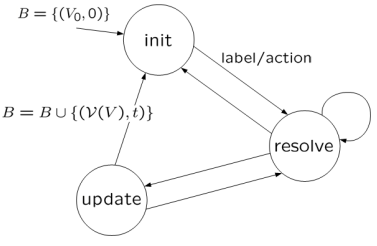


**Fig. 4.** Time stamper finite state machine

The set of actions that can be used to "customize" the time stamper are: `next`, which selects the next time stamp, and `resolve`, `init` and `update` that are each used to invoke the execution of the corresponding homonymous algorithms. The set of predicates that can be used are `true`, `false`, thresholds on the integration error, and `domainchange`, which checks if the values of the variables $H.V$ have caused a domain change. Depending on how such predicates and actions are positioned on the arcs of the state machine, and depending on the implementation of `next`, several execution semantics can be implemented that lead to different sets $B$.

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 $(\gamma, t)$ to the set $B$.

In the `resolve` algorithm (Algorithm 4), the `solve` method takes an equation and computes the value of the unknown variables at time $t$. Computation is done on the auxiliary set $V_t$. Depending on the equation ordering, it might happen that the equation admits more than one solution. In this case, `solve`

**Algorithm 1.** `resolve` algorithm

```
resolve(t)
D' ⇐ {D ∈ 𝒟 | val(V_t) ∈ D}                    // Compute the set of active domains.
I ⇐ ∅, E_t ⇐ ∅
I ⇐ ∪_{D∈𝒟'} σ(D)                               // Collect all active dynamics and components.
for all i ∈ I do
   E_t = E_t ∪ ω(i)                             // Collect all active equations.
end for
sort(E_t, π)                                    // Order the equations.
for all e_i ∈ E_t do
   solve(e_i,t)
end for
D'' ⇐ {D ∈ 𝒟 | val(V_t) ∈ D}                   // Set of active domains after the computation.
markchange(D', D'')                             // Check if the set of active domains has changed.
```

has several options: it could assign a special value *any* to all variables to indicate that a unique solution could not be computed; it could return a set of solutions; it could pick one solution depending on specific criteria. In fact, `solve` can be seen as another interface that can be customized depending on the source-language semantics. Finally, the function `markchange` checks if during the equation resolution phase a domain change has happened. This decision also depends on the semantics of the source language. Algorithm `init` initializes the auxiliary variables $V_t$ to a value that depends on the reset function $H.\rho$ and on the algorithm implemented by the time stamper. Algorithm `update` executes $val(V) = val(V_t)$, which assigns the intermediate-computation values to the system variables.

The abstract semantics can be refined into a concrete semantics by fully specifying the algorithms and functions that we have described in this section. Some of them, e.g. equation ordering, are easy to specify while others like `solve` and `next` have usually very complicated implementations. Consequently, for these functions we foresee the development of standard libraries that can be selected in the translation from one language to the interchange format. Tools for simulation map directly onto the scheduling specification. Tools for verification and synthesis can also be applied by taking advantage of the trace semantics $B$ discussed in Section 4 and of the underlying Metropolis Meta-Model [12], which defines a formal semantics for the schedulers that is suitable for analysis. The use of libraries can further simplify the analysis with the use of pre-characterized components. The Meta-Model also supports declarative properties and constraints, which can be used as links to tools and components described in other models of computation.

*Back-tracking and Algebraic Loops.* As shown in Figure 4, 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 [13].

## 5 Partitioning Structure and Semantics

In Section 3 and 4 we defined the syntax and abstract semantics of the interchange format and we showed how the abstract semantics can be refined into many concrete semantics. In this section we show 1) how the semantics can still be formally defined by partitioning the `resolve` algorithm among components and 2) how structure and semantics can be clearly separated such that it is possible to assign different semantics to hybrid systems having the same structure. 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. This section formalizes and justifies the structure of the interchange format presented in [1] and shown in Figure 1.

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 for a hybrid system $H$. Then, $\mathcal{I} : \mathcal{C} \rightarrow \mathcal{S}$ is a bijection that for each component $c$ returns its associated scheduler. Note that we use instances of components and schedulers instead of objects. Also note that the same symbol $H$ has been used here and in Section 3, but this should not confuse the reader since the object and the elements in the tuple are the same, while the tuple is just partitioned in a component and a scheduler.

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. Given 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 pair 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 binary relation (the edges of the tree). If $r = (c_i, c_j) \in C_E$ we say that $c_j$ is instantiated in $c_i$.

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 time-stamper. The subtree induced[2] 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 $T$. The input degree of $T$ is always equal to zero.

We illustrate this concept using the example in Figure 2.

*Example 5.* Figure 5, which shows the structure of the rectifier, has two interpretations:

[2] 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.

– it captures the organization of a design. For instance, component `Diode` contains two instances: component $R_d$ and component $I_0$;
– it represents the parse tree of the algebraic composition

$$Rect = v_s \,||\, Diode \,||\, \text{GND} \,||\, \text{SUB} \,||\, Load = v_s \,||\, (R_d \,||\, I_0) \,||\, \text{GND} \,||\, \text{SUB} \,||\, (R \,||\, C)$$

□

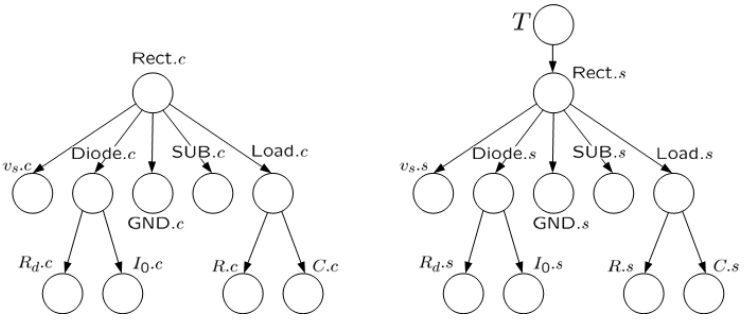


**Fig. 5.** Structural representation of the half-wave rectifier

Being able to capture hierarchies in a formal way is extremely important for an interchange format in order to retain the structure of the original specification and to allow "back translation" without loss of information.

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`.

The time stamper, which has been presented in Section 4, invokes the `init`, `resolve` and `update` functions on the "root scheduler" of $S_N$ for a given time

**Algorithm 2.** `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
```

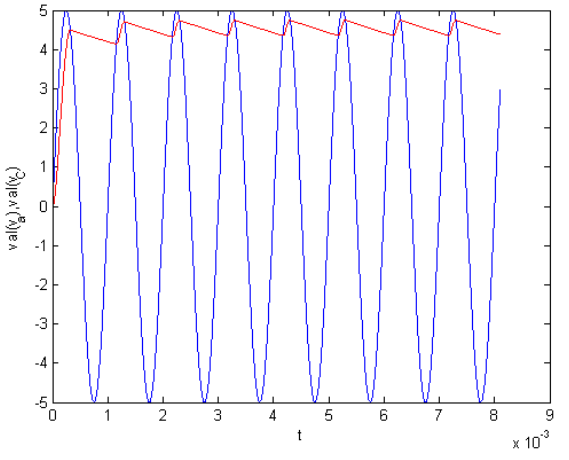


**Fig. 6.** Simulation result of the rectifier circuit

stamp $t$. In particular, the `resolve` algorithm (Algorithm 2) 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 $\Pi$. Notice that $\Pi$ together with ordering $\pi$ defined in the leaves implement the ordering $H.\pi$.

The `init` and `update` algorithms recursively call the `init` and `update` along the tree using the ordering in $\Pi$. They simply initialize variables to a given value and copy the auxiliary variable $V_t$ into $V$, respectively.

We have implemented the rectifier circuit in the METROPOLIS framework and the simulation results can be observed in Figure 6. We used a fixed step size solver as a time stamper and simulated the rectifier for $C = 10^{-4}\mu F, R = 100\Omega$, and for an input voltage $v_s(t) = 5\sin(2\pi 10^3 t)$.

## 6 Applications

The structure of the interchange format introduced in Section 5 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.

Figure 7 a) shows the structure of a language that supports neither hierarchy nor composition. Examples of languages belonging to this class are CHECKMATE [7], `d/dt` [14], and HYSDEL [8]. 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 inequalities 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.

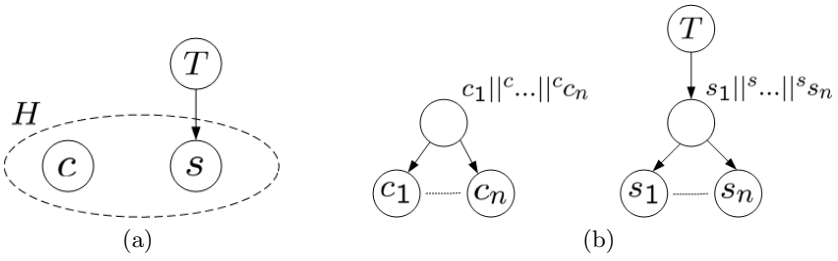


**Fig. 7.** Structure of the programs that do not support a) hierarchy and composition, b) hierarchy

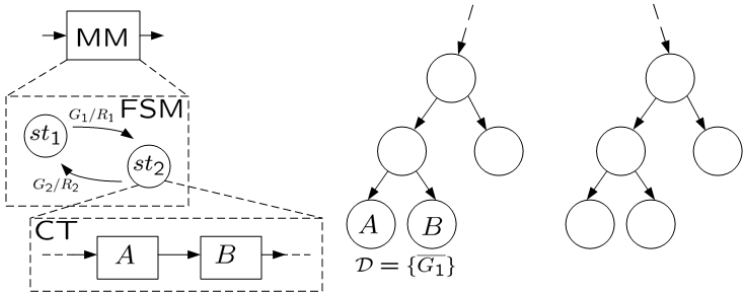


**Fig. 8.** Structure of a HyVisual modal model

Figure 7 b) shows the structure of programs that support composition but not hierarchy. Examples of languages belonging to this class are HyTech [15] and HSIF [16]. Each child of the root node is a hybrid system. For HSIF programs, hybrid automata are ordered with respect to a dependency graph. The graph nodes are 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 8 shows the structure of a HyVisual *modal model* [17]. 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 satisfied (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 8. Charon [9] 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 decide 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, the most natural among which would be to have a root node connected to as many dynamical systems as there are domains in the original CheckMate model.

For each language, the interchange format representation also highlights semantic and structural properties such as 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 first require 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.

## 7 Conclusions

We discussed the importance of an abstract semantics as the foundation of an interchange format for hybrid system design. In particular, we defined an abstract semantics for the interchange format that we first proposed in [1]. The abstract semantics can 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 such 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. 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

2. Lygeros, J., Tomlin, C., Sastry, S.: Controllers for reachability specifications for hybrid systems. In: Automatica. Volume 35. (1999)
3. Lee, E.A., Zheng, H.: Operational semantics of hybrid systems. In: HSCC. (2005) 25–53
4. Fritzson, P.: Principles of object-oriented modeling and simulation with Modelica 2.1. J. Wiley & Sons (2004)
5. Tiller, M.M.: Introduction to physical modeling with Modelica. Kluwer Academic Publishers (2001)
6. 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) available at `http://ptolemy.eecs.berkeley.edu/hyvisual/`.
7. Silva, B.I., Richeson, K., Krogh, B., Chutinan, A.: Modeling and verifying hybrid dynamic systems using CheckMate. In: Proceedings of 4th International Conference on Automation of Mixed Processes. (2000) 323–328
8. 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**(2) (2004) 235–249
9. 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
10. Alur, R., Grosu, R., Lee, I., Sokolsky, O.: Compositional refinement for hierarchical hybrid systems. In Benedetto, M.D., Sangiovanni-Vincentelli, A., eds.: Hybrid Systems: Computation and Control. Volume 2034 of Lecture Notes in Computer Science., Springer-Verlag (2001)
11. Sprinkle, J., Ames, A.D., Pinto, A., Zheng, H., Sastry, S.S.: On the partitioning of syntax and semantics for hybrid systems tools. In: 44th IEEE Conference on Decision and Control and European Control Conference ECC 2005 (CDC-ECC'05), (accepted for publication) (2005)
12. Team, T.M.P.: The Metropolis meta model version 0.4. Technical Report UCB/ERL M04/38, University of California, Berkeley (2004)
13. 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)
14. 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
15. 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**(1–2) (1997) 110–122
16. Group, M.: Hsif semantics (version 3, synchronous edition). Internal document, The University of Pennsylvania (August 22, 2002)
17. 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/`.