# Foundations of a Compositional Interchange Format for Hybrid Systems *

D.A. van Beek, M.A. Reniers, R.R.H. Schiffelers, and J.E. Rooda

Eindhoven University of Technology (TU/e)
P.O. Box 513, NL-5600 MB Eindhoven, The Netherlands.
{D.A.v.Beek,M.A.Reniers,R.R.H.Schiffelers,J.E.Rooda}@tue.nl

**Abstract.** A compositional interchange format for hybrid systems is defined in terms of an interchange automaton, allowing arbitrary differential algebraic equations, including fully implicit or switched DAEs, discrete, continuous and algebraic variables, that can be internal or external, urgency conditions, and operators for parallel composition, action hiding, variable hiding and urgent actions. Its compositional semantics is formally defined in terms of a hybrid transition system. This allows development of transformations to and from other formalisms that can be proven to preserve essential properties, and it allows a clear separation between the mathematical meaning of a model and implementation aspects such as algorithms used for solving differential algebraic equations.

## 1 Introduction

Our intention is to establish inter-operability of a wide range of tools by means of model transformations to and from a compositional interchange format that is defined in terms of an *interchange automaton*. The domain of the interchange automaton format consists of languages and tools from computer science and from dynamics and control for modeling, simulation, analysis, controller synthesis, and verification in the area of hybrid and timed systems. The purpose of an interchange format is to avoid the implementation of many bi-lateral translators between specific formalisms. Instead, the translation from a formalism A to a formalism B is divided in two steps: first, the model in formalism A is translated into a representation (model) in the interchange automaton format, then, this representation is translated into a model in formalism B [1].

Our main requirements for the interchange format are summarized below. A more detailed discussion of these requirements follows in Sections 2 and 3.

1. It should have a formal and compositional semantics, based on (hybrid) transition systems, and allow property preserving model transformations.

* Work partially done in the framework of the HYCON Network of Excellence, contract number FP6-IST-511368

2. Its concepts should be based on mathematics, and independent of implementation aspects such as equation sorting, and numerical equation solving algorithms.
3. It should support arbitrary differential algebraic equations (DAEs), including fully implicit equations, higher index systems, algebraic loops, steady state initialization, switched systems such as piecewise affine systems, and DAEs with discontinuous right hand sides.
4. It should support a wide range of concepts originating from hybrid automata, including different kinds of urgency, such as 'urgency predicates', 'deadline predicates', 'triggering guard semantics', and 'urgent actions'.
5. It should support parallel composition with synchronization by means of shared variables and shared actions.
6. It should support hierarchy and modularity to allow the definition of parallel modules and modules that can contain other modules (hierarchy), and to allow the definition of variables and actions as being local to a module, or shared between modules.

Other work on interchange formats for hybrid systems has been carried out in different projects: in the MoBIES project, the Hybrid System Interchange Format (HSIF) [2] is defined; in [3] an 'abstract semantics' of an interchange format based on the Metropolis meta model is defined (this work is a continuation of the COLUMBUS project [4]); and in the HYCON NoE [5], an interchange format for switched linear systems [6] in the form of piecewise affine systems (PWAs) is defined.

In HSIF, a network of hybrid automata is used for model representation. The network behaves as a parallel composition of its automata, without hierarchy or modules. Variables can be shared or local, and the communication mechanism is based on broadcasting of boolean 'signals', where signals are partitioned in input and output signals. Each signal is required to be either a global input to the network or to be modified by exactly one automaton. The semantics is defined only for 'acyclic dependency graphs' with respect to the use of signals. The time dependent behavior is specified by means of ordinary differential equations (ODEs), together with algebraic relations of the form $x = f(x_1, \ldots, x_n)$, and invariants. The equation $\dot{x} = 0$ is assumed for each shared variable. Circular dependencies of the algebraic equations, i.e. algebraic loops, are not allowed, and urgency predicates or urgent actions are not available [2]. The interchange automaton format defined in this article aims to be more general than HSIF, and does not incorporate tool limitations, such as restrictions on circular dependencies, or restrictions on shared variables or algebraic loops, in its compositional formal semantics.

The 'abstract semantics' presented in [3], takes implementation considerations into account, such as equation sorting, iterations that may be required for state-event detection, and iterations for reaching a fixed-point in case of algebraic loops. The semantics is defined in terms of functions and algorithms such as init, markchange, and solve. This is different from the compositional formal semantics as defined in Section 5, which aims at defining the *mathematical* mean-

ing of interchange automata, independently of implementation aspects such as equation sorting or state-event detection. For example, the semantics defines the mathematical meaning of a switched system of equations, such as a PWA system, but an implementation may choose to implement such switching behavior with or without state-event detection.

A transformation from the PWA-based interchange format [6] to the interchange automaton format will be developed. Based on this transformation, several tools, based on among others PWA, HYSDEL, MLD (see [7] for an overview relating these languages) can then be connected to the interchange automaton format.

The remainder of this article is organized as follows: Section 2 discusses the importance of a compositional formal semantics, Section 3 discusses the concepts present in the interchange automaton format, Sections 4 and 5 define the syntax and semantics of interchange automata, respectively, and Section 6 presents concluding remarks.

## 2 Importance of a Compositional Formal Semantics

To use the interchange automaton format for verification purposes, translations from models to the interchange format, and vice versa, should preserve essential properties. I.e., verification results obtained for a derived model should also be valid for the original model specified in another language.

The different languages may have different features and semantics, complicating translations between them. To keep the translations between the interchange automaton format and the other languages manageable, in terms of complexity, it is important that transformations between parts of specifications within the interchange format itself can be defined. To allow such transformations, it is essential that the semantics of the interchange automaton format is *compositional*. I.e., that the notion of equivalence is a congruence for all operators of the interchange automaton format, see [8]. Parts of a model can then be replaced by equivalent parts without changing the meaning of the model.

Consider, for instance, a transformation of a model in a simulation language, such as Modelica [9] or EcosimPro [10], to a verification tool, such as PHAVer [17] or HyTech [16]. The simulation languages use a triggering guard semantics (see Section 3.4), whereas the verification tools use invariants to force switching to a different location. Defining direct translations from the simulation languages to the verification tools and reasoning about the correctness of the translations would be difficult, since the simulation languages do not have a formal semantics and the urgent guards in two languages would need to be transformed into invariants in combination with non-urgent guards in two other languages. By using the compositional interchange format as an intermediate, the complicated direct translations can be replaced by more straightforward translations from the simulation languages to the interchange format, using urgent guards, and from the interchange format to the verification tools, using invariants; in combination with a transformation in the interchange format from urgent guards to invariants.

# 3 Concepts in the Interchange Automaton Format

## 3.1 Differential Algebraic Equations

Modeling of physical systems, such as mechanical or chemical systems frequently leads to DAEs. Algebraic constraints can also be the result of stateless components such as proportional controllers. DAEs can be modeled and simulated using languages such as Modelica and EcosimPro. DAEs can be specified in the invariants of an interchange automaton, since such invariants are predicates over all variables, including the dotted variables. Flow clauses are supported for reasons of compatibility with existing hybrid automata. The reason for not enforcing a separation between invariants (over non-dotted variables) and flow clauses (over dotted variables), as in existing hybrid automata, is that such a separation is absent in the mathematical theory of dynamical systems, including control theory. In many cases, fully implicit DAEs, such as $\mathbf{f}(\dot{\mathbf{x}}, \mathbf{x}, \mathbf{y}, t) = \mathbf{0}$, cannot even be rewritten to a form where the algebraic constraints and the differential constraints are separated, such as the semi-explicit form $\dot{\mathbf{x}} = \mathbf{g}(\mathbf{x}, \mathbf{y}, t)$, $\mathbf{h}(\mathbf{x}, \mathbf{y}, t) = \mathbf{0}$, where $\mathbf{x}$ and $\mathbf{y}$ are the continuous and algebraic variables, respectively. The generalized invariant allows us to consider the four expressions $x = 1 \wedge x = 2$, $\dot{x} = 1 \wedge \dot{x} = 2$, $\dot{x} = y \wedge \dot{x} = 2y \wedge y = 1$ and 'false' to be equivalent (bisimilar): no behavior is possible.

The initialization clause of the interchange automaton is also defined as a predicate over all variables, including the dotted variables. This allows more general initializations than usually allowed in hybrid automata. In particular, steady state initialization, as available in Modelica and EcosimPro, is supported. E.g. by defining $\dot{\mathbf{x}} = \mathbf{0}$ as initialization predicate for a location with invariant $\mathbf{f}(\dot{\mathbf{x}}, \mathbf{x}, \mathbf{y}, t) = \mathbf{0}$, the initial state is defined as the 'steady state', that is the solution of the set of DAEs such that all derivatives are zero: $\mathbf{f}(\mathbf{0}, \mathbf{x}, \mathbf{y}, t) = \mathbf{0}$.

## 3.2 Discrete, Continuous and Algebraic Variables

The interchange automaton defines three classes of variables: the discrete and continuous variables, and in addition the algebraic variables. The differences are as follows: Continuous variables are the only variables for which dotted variables (derivatives) can be used in models. The values of discrete variables remain constant when model time progresses, the values of continuous variables may change according to a continuous function of time when model time progresses, and the values of algebraic variables may change according to a discontinuous function of time. Finally, there is a difference between the different classes of variables with respect to how the resulting values of the variables in a transition relate to the starting values of the variables in the next transition. The resulting value of a discrete or continuous variable in a transition always equals its starting value in the next transition. For algebraic variables there is no such relation, because algebraic variables are not part of the state.

The state of an interchange automaton consists of, among others, the interchange automaton itself, and a valuation of the discrete and continuous variables

(see Section 5 for a more precise definition of the state). The values of the dotted variables and the algebraic variables are not contained in the state. The reason for this is that the state of an interchange automaton represents all information needed to determine future behavior, i.e., the state of a system makes the system's history irrelevant. The dotted and algebraic variables are not needed in the state, because their values are determined completely by the interchange automaton: in particular by the initial conditions, the flow conditions, the invariants and the jump predicates as defined in Section 4.

In most languages that allow (implicit) DAEs, such as Modelica [9], EcosimPro [10], and Simulink [11], the distinction between continuous and algebraic variables is implicitly made by considering all continuous variables that do not occur differentiated as algebraic.

### 3.3 Automata Related Concepts

Many different hybrid automaton definitions exist. Some definitions require solutions for the continuous variables to be differentiable functions, e.g. in [12, 13]. Other definitions allow the more general case of piecewise differentiable or piecewise continuous functions, e.g. in [14]. Such restrictions can be realized in the interchange automaton format by means of the parameters $F$ and $G$ as defined in Section 5. In [15], for each variable a 'dynamic type' can be defined. However, since we did not find such expressivity in tools, the interchange automaton format allows the definition of the dynamic type for the algebraic and continuous variable classes, not for each individual variable.

With respect to the meaning of jump predicates, that define the behavior of the variables in action transitions, differences also occur: in [12] the variables can in principle perform arbitrary jumps unless restricted by the jump predicate, in [16], variables in principle remain unchanged unless changes are enforced by the jump predicate by means of primed variables. The first behavior is obtained by an interchange automaton that defines the set of jumping variables $W$ (see Sections 4 and 5.1) at each edge to be equal to the set of all variables. The second kind of behavior is obtained by defining the set $W$ as the union of all primed variables of the jump predicate. The specification of a set of jumping variables and a jump predicate for each edge of an interchange automaton is based on [13].

The interchange automaton format is expressive enough to deal with verification tools such as PHAVer [17] and HyTech [16]. The behavior of the algebraic variables from the interchange automaton is related to the external variables from the semantical hybrid I/O automaton defined in [15]. In this I/O automaton, the external variables are also not part of the state, and they can have a dynamic type that allows discontinuous trajectories. The state is defined by the values of the internal variables, and discrete transitions (action transitions) are defined only on internal variables. The interchange automaton format can express this as a special case, since the different classes of variables, action transitions, and hiding/abstraction are orthogonal concepts in the interchange automaton format.

The basic elements of hierarchy and modularity that are supported by the interchange format are parallel composition, and hiding of variables and actions. Interchange automata can be grouped by means of parallel composition, and variables and/or actions that are meant to be local to that group can be hidden from the environment of the group. The concrete interchange format, that will be developed, will define modularity in terms of the basis elements of the abstract format.

### 3.4 Urgency

The concept of urgency allows the passing of time up to a certain point. There are essentially two different kinds of urgency:

1. Urgency that is defined for an atomic automaton by means of one or more predicates. Such predicates can be associated to a location, or to outgoing edges of the location.
2. Urgency that is defined as an operation on a composition of one or more automata. Such an operation defines a set of actions as urgent for the composition. The operation allows the passing of time up to the point when one or more of the urgent actions can be executed.

The first kind of urgency is defined in many different forms. The `tcp` (*time can progress*) predicate [18], is a predicate over the variables of the automaton and time. The predicate is associated to a location. It allows passing of time in a location for as long as the predicate is true. Related to the `tcp` predicate is the *stopping condition* [19], which is a predicate on the variables of the automaton, also associated to a location, and which allows passing of time in a location for as long as the stopping condition is false, or in other words, until the time-point when the stopping condition is true. *Deadline predicates* [20] and *urgency predicates* [19] are associated to the edges of an automaton. Deadline predicates allow passing of time in a location until the time-point that one or more deadline predicates of the outgoing edges of the location become true. Whenever a deadline predicate of an edge becomes true, the guard associated to that edge must also be true: the deadline predicate must imply the guard. Urgency predicates are similar to deadline predicates; the only difference is that they do not have the restriction that the urgency predicate should imply the guard. Urgency predicates allow passing of time in a location until the point of time that for one or more of the outgoing edges, the guard and the urgency predicate are both true.

Restricting a `tcp` predicate as a predicate over the variables of an automaton makes it equal to the negation of a stopping condition. Deadline predicates and urgency predicates are less expressive. They can both be expressed in terms of stopping conditions, see [19], or as `tcp` predicates. E.g. the stopping condition of a location corresponds to the disjunction of all deadline predicates of the outgoing edges of the location. Note that a flow condition which is false in a hybrid automaton is equivalent to a stopping condition that is true, or a `tcp` predicate that is false. The interchange automaton format adopts stopping conditions, which we refer to as *urgency conditions*.

In simulation languages, such as Modelica, EcosimPro, and HyVisual, usually a triggering or urgent guard semantics is used, meaning that the passing of time in a location is allowed until the time-point that any of the guards of the outgoing edges becomes true. This is equivalent to a stopping condition associated to the location that is the disjunction of the guards of all outgoing edges.

The second kind of urgency, as for example defined in [21] and [22], is often available with restrictions only. E.g. in HyTech, edges can be defined as urgent. The composition of urgent actions is required to be well-formed: 'whenever two components synchronize on a label, if one transition is urgent then the other must either be urgent, or have a jump condition expressible as a guarded command with its guard being either the predicate true or the predicate false' [16]. A second restriction is that 'if there exists an urgent transition from a location $v$ to a location $v'$, then for all valuations satisfying the invariant of $v$, an urgent transition to $v'$ should exist'.

The interchange automaton format defines the second kind of urgency by means of the urgent action operator. Note that defining this kind of urgency by means of labeling certain edges or actions as urgent may lead to bisimulation not being a congruence for parallel composition, as described in [23]. Another example is the ASAP flag that can be attached to an edge in HyTech. In such cases, replacing a part of a specification by another part with the same behavior may lead to different behavior of the complete system. Straightforward translations of such languages to and from the interchange format is in principle possible if each action (or edge with a certain action) is either always urgent or never urgent in a model. If the same action (or edge with a certain action) occurs both as urgent and not urgent, it may be necessary to eliminate parallel composition, as for example described in [8], before translation to the interchange format is possible.

## 4 Syntax of Interchange Automata

**Notation 1** *The following notations are defined:*

- *A set $\mathcal{X}$ of variables and a set of action labels $\mathcal{L}$, which does not include the predefined non-synchronizing action $\tau$, are assumed. The set $\mathcal{L}_\tau$ denotes the set $\mathcal{L} \cup \{\tau\}$.*
- *For a set of variables $S \subseteq \mathcal{X}$, $\dot{S} = \{\dot{x} \mid x \in S\}$ denotes the set of dotted variables.*
- *For a set of variables $S \subseteq \mathcal{X}$,* Pred($S$) *denotes the set of all predicates over variables from $S$.*
- *$f : A \mapsto B$ and $g : A \to B$ define a partial function $f$ and a total function $g$, both with domain $A$ and range $B$.*

**Definition 1 (Atomic Interchange Automaton).** *An* atomic interchange automaton *is a tuple* ($X$, $X_\mathrm{i}$, dtype, $V$, $v_0$, init, flow, inv, urgent, $L$, $E$) *where*

- *$X \subseteq \mathcal{X}$ is a finite set of variables, $X_\mathrm{i} \subseteq X$ is the set of* internal *variables, and $X_\mathrm{e} = X \setminus X_\mathrm{i}$ is the set of* external *variables.*

- $\mathrm{dtype} : X \to \{\mathrm{disc}, \mathrm{cont}, \mathrm{alg}\}$ *is a function that associates to each variable a dynamic type:* discrete*,* continuous *or* algebraic*. The sets* $X_{\mathrm{disc}}, X_{\mathrm{cont}}, X_{\mathrm{alg}}$ *are defined as* $X_t = \{x \in X \mid \mathrm{dtype}(x) = t\}$ *for* $t \in \{\mathrm{disc}, \mathrm{cont}, \mathrm{alg}\}$*, and* $X_{\mathrm{state}} = X_{\mathrm{disc}} \cup X_{\mathrm{cont}}$ *is the set of* state *variables.*
- $V$ *is a finite non-empty set of* vertices*, called* locations*, and* $v_0 \in V$ *is the initial location.*
- $\mathrm{init} \in \mathrm{Pred}(\tilde{X})$ *is the initial condition. For* $Y \subseteq X$*,* $\tilde{Y} = Y \cup \{\dot{y} \mid y \in Y \cap X_{\mathrm{cont}}\}$ *is the extension of* $Y$ *with the dotted versions of the continuous variables in* $Y$*.*
- $\mathrm{flow}, \mathrm{inv}, \mathrm{urgent} : V \to \mathrm{Pred}(\tilde{X})$*, are functions that each associate to each location* $v \in V$ *a predicate describing the* flow condition*, the* invariant*, and the* urgency condition*, respectively.*
- $L \subseteq \mathcal{L}$ *is a finite set of action labels.*
- $E = V \times \mathrm{Pred}(\tilde{X}) \times (L \cup \{\tau\}) \times (\mathcal{P}(\tilde{X}) \times \mathrm{Pred}(\tilde{X} \cup \tilde{X}^-)) \times V$ *is a finite set of* edges*, such that for each element* $(v, g, \ell, (W, r), v') \in E$*,* $v$ *and* $v'$ *are the* source *and* target *locations, respectively,* $g$ *is the* guard*,* $\ell$ *is the* action label*,* $W \subseteq \tilde{X}$ *is a set of jumping variables (the value of which may change as a result of an action transition), and* $r$ *is the* jump predicate*, also called* reset map*. For any* $Y \subseteq \tilde{X}$*,* $Y^- = \{y^- \mid y \in Y\}$ *denotes the set of minus superscripted variables that represent the values of variables before an action transition.*

Note that the *dynamic* type of a variable gives information about its time dependent behavior. E.g. the value of a discrete variable remains constant when time passes, whereas the value of a continuous variable changes as a continuous function of time. The *static* type, such as real, integer or boolean, mainly gives information about the domain in which the variable takes values.

The interchange automaton format consists of automata, and operators for parallel composition, for hiding of action labels and variables, and for the definition of urgent actions. The automata and operators can be freely combined:

**Definition 2 (Interchange automaton).** *The set of interchange automata* $A$ *is defined by the following grammar for the interchange automata* $\alpha \in A$*:*

$$
\begin{array}{lll}
\alpha ::= \alpha_{\mathrm{atom}} & & \textit{atomic interchange automaton} \\
\quad \mid \alpha \parallel \alpha & & \textit{parallel composition} \\
\quad \mid \mathrm{hide}_{\mathrm{act}}(L_{\mathrm{h}}, \alpha) & & \textit{action hiding operator} \\
\quad \mid \mathrm{hide}_{\mathrm{var}}(X_{\mathrm{h}}, \alpha, \sigma_{\mathrm{h}}) & & \textit{variable hiding operator} \\
\quad \mid \mathrm{urgent}(L_{\mathrm{u}}, \alpha) & & \textit{urgent action operator,}
\end{array}
$$

*where*

- $\alpha_{\mathrm{atom}}$ *denotes an arbitrary atomic interchange automaton;*
- $L_{\mathrm{h}} \subseteq \mathcal{L}$ *denotes a set of actions to hide;*
- $X_{\mathrm{h}} \subseteq \mathcal{X}$ *denotes a set of variables to hide and* $\sigma_{\mathrm{h}} : X_{\mathrm{h}} \mapsto \Lambda$ *denotes a (partial) valuation for the hidden state variables of interchange automaton* $\alpha$*;*
- $L_{\mathrm{u}} \subseteq \mathcal{L}$ *denotes a set of urgent actions.*

In the next sections, two auxiliary functions on interchange automata are used. These are defined below. The functions $\mathrm{var}_{\mathrm{e,st}}$ and act extract the sets of external state variables and external (non-hidden) action labels[1], respectively, from an interchange automaton:

**Definition 3.** *For* $\alpha_{\mathrm{a}} = (X, X_{\mathrm{i}}, \mathrm{dtype}, V, v, \mathrm{init}, \mathrm{flow}, \mathrm{inv}, \mathrm{urgent}, L, E)$*, and* $\alpha$*,* $\alpha_1$*,* $\alpha_2 \in A$ *we define the functions* $\mathrm{var}_{\mathrm{e,st}} : A \to \mathcal{P}(\mathcal{X})$ *and* $\mathrm{act} : A \to \mathcal{P}(\mathcal{L})$ *as follows:*

$$\begin{array}{ll}
\mathrm{var}_{\mathrm{e,st}}(\alpha_{\mathrm{a}}) = X_{\mathrm{state}} \cap X_{\mathrm{e}} & \mathrm{act}(\alpha_{\mathrm{a}}) = L \\
\mathrm{var}_{\mathrm{e,st}}(\alpha_1 \parallel \alpha_2) = \mathrm{var}_{\mathrm{e,st}}(\alpha_1) \cup \mathrm{var}_{\mathrm{e,st}}(\alpha_2) & \mathrm{act}(\alpha_1 \parallel \alpha_2) = \mathrm{act}(\alpha_1) \cup \mathrm{act}(\alpha_2) \\
\mathrm{var}_{\mathrm{e,st}}(\mathrm{hide}_{\mathrm{act}}(L_{\mathrm{h}}, \alpha)) = \mathrm{var}_{\mathrm{e,st}}(\alpha) & \mathrm{act}(\mathrm{hide}_{\mathrm{act}}(L_{\mathrm{h}}, \alpha)) = \mathrm{act}(\alpha) \setminus L_{\mathrm{h}} \\
\mathrm{var}_{\mathrm{e,st}}(\mathrm{hide}_{\mathrm{var}}(X_{\mathrm{h}}, \alpha, \sigma_{\mathrm{h}})) = \mathrm{var}_{\mathrm{e,st}}(\alpha) \setminus X_{\mathrm{h}} & \mathrm{act}(\mathrm{hide}_{\mathrm{var}}(X_{\mathrm{h}}, \alpha, \sigma_{\mathrm{h}})) = \mathrm{act}(\alpha) \\
\mathrm{var}_{\mathrm{e,st}}(\mathrm{urgent}(L_{\mathrm{u}}, \alpha)) = \mathrm{var}_{\mathrm{e,st}}(\alpha) & \mathrm{act}(\mathrm{urgent}(L_{\mathrm{u}}, \alpha)) = \mathrm{act}(\alpha)
\end{array}$$

## 5 Semantics of Interchange Automata

The formal semantics associates to each interchange automaton an action transition relation, a time transition relation, and a consistency predicate on states. A different way of looking at such a semantics is as a labeled transition system with two types of transitions and a predicate. The states $S$ of the labeled transition system associated to an interchange automaton consist of an interchange automaton, a valuation of the external state variables of that automaton, and a set of jumping external state variables: i.e., $S = A \times \mathit{Val} \times \mathcal{P}(\mathcal{X})$, where $\mathit{Val} = \mathcal{X} \cup \dot{\mathcal{X}} \mapsto \Lambda$ is the set of all partial mappings from $\mathcal{X} \cup \dot{\mathcal{X}}$ to the set of values $\Lambda$. The set of jumping variables $J$ is defined by other automata executing in the environment of (in parallel to) automaton $\alpha$. The valuation $\sigma$ of a state of a transition system defines values for precisely the externally visible state variables, i.e., $\mathrm{dom}(\sigma) = \mathrm{var}_{\mathrm{e,st}}(\alpha)$ for all $(\alpha, \sigma, J) \in S$.

The intuition of an action transition $(\alpha, \sigma, J) \xrightarrow{\xi, \ell, W, \xi'} (\alpha', \sigma', J)$ is that the state $(\alpha, \sigma, J)$ executes a discrete action (with action label) $\ell$ with visible valuations $\xi$, $\xi'$, before and after execution of the action, respectively, and thereby transforms into the state $(\alpha', \sigma', J)$, where $\sigma'$ denotes the accompanying valuation of the automaton $\alpha'$, after the discrete action $\ell$ is executed. The set $W$ represents the external state variables that are allowed to change (jump) in this action transition. They need to be visible for synchronization in a parallel composition of interchange automata.

The intuition of a time transition $(\alpha, \sigma, J) \overset{t, \rho}{\mapsto} (\alpha', \sigma', J)$ is that model time passes for $t$ time units, and the valuation at each time-point $s \in [0, t]$ is given by $\rho(s)$ for the externally visible variables. At the end-point $t$, the resulting state is $(\alpha', \sigma', J)$.

The intuition of the consistency predicate $(\alpha, \sigma, J) \overset{\xi}{\rightsquigarrow}$ is that the interchange automaton $\alpha$ is consistent with extended valuation $\xi$, which means that the invariants of all active locations of $\alpha$ are satisfied in $\xi$.

[1] This does not mean that these actions are actually used. It is allowed to specify the set of actions much broader than the actions that appear on transitions.

**Notation 2** *In this section, some notations and operators are used. These are defined as follows:*

- $\restriction$ *is the restriction operator on functions. If* $f$ *is a function, and* $S$ *is a set,* $f \restriction S$ *denotes the restriction of* $f$ *to* $S$*, that is, the function* $g$ *with* $\mathrm{dom}(g) = \mathrm{dom}(f) \cap S$*, such that* $g(c) = f(c)$ *for each* $c \in \mathrm{dom}(g)$*.*
- $\downarrow$ *is the projection operator on functions, which is used here on trajectories. For* $\rho : T \mapsto (Y \to \Lambda)$*,* $S \subseteq Y$ *and* $x \in Y$*,* $\rho \downarrow S$ *denotes the function* $\rho' : T \mapsto (S \to \Lambda)$ *such that* $\rho'(t) = \rho(t) \restriction S$ *for each* $t \in T$*; and* $\rho \downarrow x$ *denotes the function* $f : T \mapsto \Lambda$ *such that* $f(t) = \rho(t)(x)$ *for each* $t \in \mathrm{dom}(\rho)$*.*
- *For atomic interchange automaton* $\alpha_{\mathrm{atom}}$ *given by* $(X,\ X_{\mathrm{i}},\ \mathrm{dtype},\ V,\ v,\ \mathrm{init},\ \mathrm{flow},\ \mathrm{inv},\ \mathrm{urgent},\ L,\ E)$*,* $\alpha_{\mathrm{atom}}[v', \mathrm{init}'/v, \mathrm{init}]$ *denotes the atomic interchange automaton obtained from atomic interchange automaton* $\alpha_{\mathrm{atom}}$ *by replacing* $v$ *by* $v'$ *and* init *by* init$'$*, and* $\alpha_{\mathrm{atom}}[v'/v] = \alpha_{\mathrm{atom}}[v', \mathrm{init}/v, \mathrm{init}]$*.*

### 5.1 Semantics of Atomic Interchange Automata

**Definition 4 (Action transitions).** *Consider an atomic interchange automaton* $\alpha = (X,\ X_{\mathrm{i}},\ \mathrm{dtype},\ V,\ v,\ \mathrm{init},\ \mathrm{flow},\ \mathrm{inv},\ \mathrm{urgent},\ L,\ E)$*. The* action transition relation $\_ \xrightarrow{\cdot} \_ \subseteq S \times (\mathit{Val} \times \mathcal{L}_\tau \times \mathcal{P}(\mathcal{X}) \times \mathit{Val}) \times S$ *is for* $(\alpha, \sigma, J), (\alpha', \sigma', J) \in S$*,* $\xi_{\mathrm{e}}, \xi'_{\mathrm{e}} \in \mathit{Val}$*,* $\ell \in \mathcal{L}_\tau$*, and* $W_{\mathrm{e}} \subseteq \mathcal{X}$*, defined as follows:* $(\alpha, \sigma, J) \xrightarrow{\xi_{\mathrm{e}}, \ell, W_{\mathrm{e}}, \xi'_{\mathrm{e}}} (\alpha', \sigma', J)$*, if and only if there exist an edge* $(v, g, \ell, (W, r), v') \in E$ *and* $\xi, \xi' \in \mathit{Val}$ *with* $\mathrm{dom}(\xi) = \mathrm{dom}(\xi') = \tilde{X}$ *such that*

- $\xi \restriction \tilde{X}_{\mathrm{e}} = \xi_{\mathrm{e}}$ *and* $\xi' \restriction \tilde{X}_{\mathrm{e}} = \xi'_{\mathrm{e}}$*;*
- $\xi_{\mathrm{e}} \restriction X_{\mathrm{state}} = \sigma$ *and* $\xi'_{\mathrm{e}} \restriction X_{\mathrm{state}} = \sigma'$*;*
- $\xi \models \mathrm{init}$ *and* $\xi \models g$*;*
- $\xi \models \mathrm{inv}(v)$ *and* $\xi' \models \mathrm{inv}(v')$*;*
- $\xi' \cup \xi^- \models r$*;*
- $\xi \restriction X_{\mathrm{nonjmp}} = \xi' \restriction X_{\mathrm{nonjmp}}$*, where* $X_{\mathrm{nonjmp}} = X_{\mathrm{state}} \setminus (W \cup (J \cap X_{\mathrm{e}}))$*;*
- $W_{\mathrm{e}} = W \cap X_{\mathrm{state}} \cap X_{\mathrm{e}}$*;*
- $\alpha' = \alpha[v', \mathrm{init}'/v, \mathrm{init}]$*,* $\mathrm{init}' = \left( \bigwedge_{x \in X_{\mathrm{state}} \cap X_{\mathrm{i}}} x = c_x \right)$*, and* $c_x \in \Lambda$ *is given by* $c_x = \xi'(x)$*.*

Here, *val* $\models$ *pred*, where *val* is a valuation and *pred* a predicate, means that *pred* is satisfied when all variables occurring in it are substituted by their values as defined in *val*. Minus superscripted variables, such as $x^-$, occurring in $r$ are evaluated in $\xi^-$, which is defined as $\mathrm{dom}(\xi^-) = \{x^- \mid x \in \mathrm{dom}(\xi)\}$, and $\xi^-(x^-) = \xi(x)$. The 'non-jumping' variables in the set $X_{\mathrm{nonjmp}} = X_{\mathrm{state}} \setminus (W \cup (J \cap X_{\mathrm{e}}))$ are the variables the values of which are not allowed to change in an action transition. These variables are the discrete and continuous variables apart from two sets of variables: the variables from set $W$ and the externally visible variables from set $J$ ($J \cap X_{\mathrm{e}}$). The jumping variables in set $J$ are the result of changes in external variables of synchronizing automata, as defined in the semantics of parallel composition in [8]. The updated initial condition init$'$ acts

as a local valuation which ensures that for each local state variable $x$, its starting value for the next transition equals its resulting value (here: $\xi'(x)$, in Definition 5: $\rho(t)(x)$) for the current transition. Note that we do not combine the valuation of the internal variables with the valuation of the external variables in $\sigma$. Having the valuations of the local variables in $\sigma$ would lead to restrictions on the parallel composition of two automata, namely that the sets of internal variables of two automata in a parallel composition would need to be disjoint. Otherwise, local variables with the same names in parallel automata would become shared. See for example [15], where the local variables and local actions of automata in a parallel composition are required to be disjoint.

**Definition 5 (Time transitions).** *Consider an atomic interchange automaton* $\alpha = (X,\ X_{\mathrm{i}},\ \mathrm{dtype},\ V,\ v,\ \mathrm{init},\ \mathrm{flow},\ \mathrm{inv},\ \mathrm{urgent},\ L,\ E)$. *The* time transition relation $\_ \overset{\cdot}{\longmapsto} \_ \subseteq S \times (T \times (T \mapsto \mathit{Val})) \times S$ *is for* $(\alpha, \sigma, J), (\alpha', \sigma', J) \in S$, $t \in T$, *and* $\rho_{\mathrm{e}} : [0,t] \to \mathit{Val}$, *defined as follows:* $(\alpha, \sigma, J) \overset{t,\rho_{\mathrm{e}}}{\mapsto} (\alpha', \sigma', J)$, *if and only if there exists a* $\rho : [0,t] \to \mathit{Val}$ *with* $\mathrm{dom}(\rho(s)) = \tilde{X}$ *for all* $s \in [0,t]$ *such that*

- $\rho \downarrow \tilde{X}_{\mathrm{e}} = \rho_{\mathrm{e}}$;
- $\rho_{\mathrm{e}}(0) \upharpoonright X_{\mathrm{state}} = \sigma$ *and* $\rho_{\mathrm{e}}(t) \upharpoonright X_{\mathrm{state}} = \sigma'$;
- $\rho(0) \models \mathrm{init}$;
- $\rho \downarrow x$ *is a constant function for all* $x \in X_{\mathrm{disc}}$;
- $(\rho \downarrow x) \in F$ *for all* $x \in X_{\mathrm{alg}}$;
- $\rho \downarrow \dot{x}$ *is an integrable function in the Lebesgue sense for all* $x \in X_{\mathrm{cont}}$;
- $\rho(s) \models \mathrm{flow}(v)$ *and* $\rho(s) \models \mathrm{inv}(v)$ *for all* $s \in [0,t]$;
- $(\rho \downarrow x)(s) = (\rho \downarrow x)(0) + \int_0^s (\rho \downarrow \dot{x})(s')ds'$ *for all* $x \in X_{\mathrm{cont}}$ *and* $s \in [0,t]$;
- $(\rho \downarrow x, \rho \downarrow \dot{x}) \in G$ *for all* $x \in X_{\mathrm{cont}}$;
- $\rho(s) \models \neg\mathrm{urgent}(v)$ *for all* $s \in \{0\} \cup [0,t)$;
- $\alpha' = \alpha[\mathrm{init}'/\mathrm{init}]$, $\mathrm{init}' = \left( \bigwedge_{x \in X_{\mathrm{state}} \cap X_{\mathrm{i}}} x = c_x \right)$, *and* $c_x \in \Lambda$ *is given by* $c_x = \rho(t)(x)$.

Item $(\rho \downarrow x) \in F$ for all $x \in X_{\mathrm{alg}}$, requires the trajectories of the algebraic variables to be functions of type $F$. This set of functions is a global parameter of the solution concept of an interchange automaton specification.

The relation between the trajectory of a continuous variable $x$ and the trajectory of its 'derivative' $\dot{x}$ is given by the Caratheodory solution concept [24]: $(\rho \downarrow x)(s) = (\rho \downarrow x)(0) + \int_0^s (\rho \downarrow \dot{x})(s')ds'$. This integral relation can hold only for those continuous variables for which $\rho \downarrow x$ is an absolutely continuous function, but it does allow a non-smooth trajectory for a continuous variable in the case that the trajectory of its 'derivative' $\rho \downarrow \dot{x}$ is non-smooth or even discontinuous, as in, for example, in the solution of the invariant $\dot{y} = \mathrm{step}(t-1) \wedge \dot{t} = 1$, where $t$ and $y$ are continuous variable with initial value of 0, and $\mathrm{step}(x)$ equals 0 for $x \leq 0$ and 1 for $x > 0$.

In hybrid automata, the solution concept usually defines the function $\rho \downarrow \dot{x}$ to be the derivative function of $\rho \downarrow x$ for continuous variables $x \in X_{\mathrm{cont}}$. This

can be realized for the interchange automaton format semantics by restricting the set $G$, which is used in the requirement $(\rho \downarrow x, \rho \downarrow \dot{x}) \in G$ for all $x \in X_{\mathrm{cont}}$, as $G = \{(f, f') \mid f \text{ is differentiable, and } f' \text{ is the derivative function of } f\}$. In this way, the semantics of the interchange automaton format corresponds to the usual semantics of hybrid automata.

**Definition 6 (Consistency predicates).** *Consider an atomic interchange automaton* $\alpha = (X, X_{\mathrm{i}}, \mathrm{dtype}, V, v, \mathrm{init}, \mathrm{flow}, \mathrm{inv}, \mathrm{urgent}, L, E)$*. The* consistency predicate $\_ \tilde{\rightsquigarrow} \subseteq S \times \mathit{Val}$ *is for* $(\alpha, \sigma, J) \in S$ *and* $\xi_{\mathrm{e}} \in \mathit{Val}$*, defined as follows:* $(\alpha, \sigma, J) \overset{\xi_{\mathrm{e}}}{\tilde{\rightsquigarrow}}$*, if and only if there exists a valuation* $\xi \in \mathit{Val}$ *with* $\mathrm{dom}(\xi) = \tilde{X}$ *such that* $\xi \restriction \tilde{X}_{\mathrm{e}} = \xi_{\mathrm{e}}$*,* $\xi_{\mathrm{e}} \restriction X_{\mathrm{state}} = \sigma$*,* $\xi \models \mathrm{init}$*, and* $\xi \models \mathrm{inv}(v)$*.*

### 5.2 Semantics of the Operators

The informal semantics of the operators is defined below. The formal semantics of the operators is defined in a structured operational semantics in [8].

**Parallel composition** The most common operator for composing hybrid automata is parallel composition. There are no compatibility requirements for the parallel composition of interchange automata: any pair of interchange automata can be composed by the parallel composition operator. The parallel composition operator synchronizes on all external actions that the arguments share and allows interleaving of any other actions (under the condition that they maintain the consistency of the other automaton). Time transitions must be synchronized, and consistency is established only if both automata agree on it. The external state variables that are shared by the argument automata need to have the same values (all the time).

**Hiding** The action hiding operator applied to an automaton, $\mathrm{hide}_{\mathrm{act}}(L_{\mathrm{h}}, \alpha)$, hides (abstracts from) the actions from set $L_{\mathrm{h}}$ by replacing them by the internal action $\tau$. This only affects the action behavior of $\alpha$; its delay behavior and consistency remain unchanged.

The variable hiding operator applied to an automaton, $\mathrm{hide}_{\mathrm{var}}(X_{\mathrm{h}}, \alpha, \sigma_{\mathrm{h}})$, hides the variables from set $X_{\mathrm{h}}$ by removing information about them from the action and time transitions of $\alpha$. The values of the hidden state variables are stored in valuation $\sigma_{\mathrm{h}}$.

**Urgent action operator** The urgent action operator applied to an automaton, $\mathrm{urgent}(L_{\mathrm{u}}, \alpha)$, gives actions from the set $L_{\mathrm{u}}$ priority over time passing. The action behavior and consistency of $\alpha$ are not affected by the urgent action operator. Time transitions are allowed only if at the current state, and at each intermediate state while delaying, no actions from set $L_{\mathrm{u}}$ are possible.

## 6 Concluding Remarks

The proposed interchange automaton format integrates formalisms rooted in computer science with those rooted in dynamics and control. It is indeed compositional, since bisimilarity is proved to be a congruence for all operators of the interchange format in [8]. Future work entails, among others, adding the notion of input/output variables and input/output actions, adding channels as communication mechanism between interchange automata in a parallel composition, adding additional operators, such as sequential composition, and possibly extending the interchange format with stochastic model primitives. The development of translations and simulator implementations will be done by different partners in Work Package 3 of the HYCON NoE [5]. The translations that are to be developed should also specify incompatibilities, if present, or subsets for which property preserving translations are possible.

The atomic interchange automata as introduced syntactically in Definition 1 and for which the semantics has been introduced in Section 5 are very expressive. For any application of an action or variable hiding operator on an atomic interchange automaton, it is possible to obtain an equivalent atomic interchange automaton. Also, the parallel composition of any two atomic interchange automata for which the shared variables have compatible types can be replaced by an equivalent atomic interchange automaton. The only operator that cannot be eliminated in all relevant cases is the urgent action operator. Further substantiation of these claims and ideas can be found in [8].

## References

1. Remelhe, M.A.P., Beek, D.A.v.: Requirements for an interchange format for nonlinear hybrid systems. Technical Report D 3.6.1, HYCON NoE (2006)
2. MoBIES team: HSIF semantics. Technical report, University of Pennsylvania (2002) internal document.
3. Pinto, A., Carloni, L.P., Passerone, R., Sangiovanni-Vincentelli, A.L.: Interchange format for hybrid systems: Abstract semantics. In Hespanha, J.P., Tiwari, A., eds.: Hybrid Systems: Computation and Control, 9th International Workshop. Volume 3927 of Lecture Notes in Computer Science., Santa Barbara, Springer-Verlag (2006) 491–506
4. Columbus IST project: http://www.columbus.gr (2006)
5. HYCON NoE: http://www.ist-hycon.org/ (2005)
6. Cairano, S.D., Bemporad, A., Kvasnica, M.: An architecture for data interchange of switched linear systems. Technical Report D 3.3.1, HYCON NoE (2006)
7. Heemels, W.P.M.H., Schutter, B.d., Bemporad, A.: Equivalence of hybrid dynamical models. Automatica **37**(7) (2001) 1085–1091
8. Beek, D.A.v., Reniers, M.A., Schiffelers, R.R.H., Rooda, J.E.: A compositional interchange format for hybrid systems. Technical Report SE-Report 2006-05, Eindhoven University of Technology, Department of Mechanical Engineering, The Netherlands (2006) http://se.wtb.tue.nl/sereports/.
9. Tiller, M.: Introduction to Physical Modeling with Modelica. Volume 615 of The International Series in Engineering and Computer Science. Springer-Verlag (2001)

10. EcosimPro: http://www.ecosimpro.com (2006)
11. The MathWorks, Inc: Using Simulink, version 6, http://www.mathworks.com. (2005)
12. Henzinger, T.A.: The theory of hybrid automata. In Inan, M., Kurshan, R., eds.: Verification of Digital and Hybrid Systems. Volume 170 of NATO ASI Series F: Computer and Systems Science. Springer-Verlag, New York (2000) 265–292
13. Alur, R., Henzinger, T.A., Ho, P.H.: Automatic symbolic verification of embedded systems. IEEE Transactions on Software Engineering **22**(3) (1996) 181–201
14. Schaft, A.J.v.d., Schumacher, J.M.: An Introduction to Hybrid Dynamical Systems. Volume 251 of Lecture Notes in Control and Information Sciences. Springer-Verlag (2000)
15. Lynch, N., Segala, R., Vaandrager, F.: Hybrid I/O automata. Information and Computation **185**(1) (2003) 105–157
16. Henzinger, T.A., Ho, P.H., Wong-Toi, H.: A user guide to HyTech. In Brinksma, E., Cleaveland, R., Larsen, K.G., Margaria, T., Steffen, B., eds.: Tools and Algorithms for Construction and Analysis of Systems, First International Workshop. Volume 1019 of Lecture Notes in Computer Science., Aarhus, Denmark, Springer-Verlag (1995) 41–71
17. Frehse, G.: PHAVer: Algorithmic verification of hybrid systems past HyTech. In Morari, M., Thiele, L., eds.: Hybrid Systems: Computation and Control, 8th International Workshop. Volume 3414 of Lecture Notes in Computer Science. Springer-Verlag (2005) 258–273
18. Nicollin, X., Olivero, A., Sifakis, J., Yovine, S.: An approach to the description and analysis of hybrid systems. In Grossman, R.L., Nerode, A., Ravn, A.P., Rischel, H., eds.: Hybrid Systems. Volume 736 of Lecture Notes in Computer Science., Springer (1993) 149–178
19. Gebremichael, B., Vaandrager, F.: Specifying urgency in timed i/o automata. In: Proc. Third IEEE Conference on Sofware Engineering and Formal Methods. (2005) 64–74
20. Bornot, S., Sifakis, J.: An algebraic framework for urgency. Information and Computation **163**(1) (2000) 172–202
21. Bolognesi, T., Lucidi, F.: Timed process algebras with urgent interactions and a unique powerful binary operator. In de Bakker, J.W., Huizing, C., de Roever, W.P., Rozenberg, G., eds.: Real-Time: Theory in Practice, REX Workshop. Volume 600 of Lecture Notes in Computer Science., Mook, The Netherlands, Springer-Verlag (1991) 124–148
22. Beek, D.A.v., Man, K.L., Reniers, M.A., Rooda, J.E., Schiffelers, R.R.H.: Syntax and consistent equation semantics of hybrid Chi. Journal of Logic and Algebraic Programming **68**(1-2) (2006) 129–210
23. Bohnenkamp, H.C., D'Argenio, P.R., Hermanns, H., Katoen, J.P.: Modest: A compositional modeling formalism for real-time and stochastic systems. Technical Report Technical Report TR-CTIT-04-46, University of Twente, Centre for Telematics and Information Technology, The Netherlands (2004)
24. Filippov, A.F.: Differential Equations with Discontinuous Right Hand Sides. Kluwer Academic Publishers, Dordrecht (1988)