New Concepts in the Abstract Format of the Compositional Interchange Format ⋆

D.A. van Beek * **P. Collins** † **D.E. Nadales** * **J.E. Rooda** *
R.R.H. Schiffelers *

* *Eindhoven University of Technology, P.O.Box 513, 5600 MB Eindhoven, The Netherlands*
{*d.a.v.beek, d.e.nadales.agut, j.e.rooda, r.r.h.schiffelers*}*@tue.nl*
† *Centrum Wiskunde en Informatica*
Postbus 94079, 1090 GB Amsterdam, The Netherlands
pieter.collins@cwi.nl

Abstract: The compositional interchange format for hybrid systems (CIF) supports inter-operability of a wide range of tools by means of model transformations to and from the CIF. Work on the CIF takes place in the FP7 Multiform project, and in several other European projects. The CIF consists of an abstract and a concrete format, used for defining a formal semantics and for modeling, respectively. This paper discusses the results of a redesign of the abstract format as previously published, leading to the following main changes: variables are introduced using scoping operators; the abstract language is made more orthogonal by providing an operator for each concept in the language; parallel composition has been defined in such a way that compositional verification (assume/guarantee reasoning) is supported; and the concept of urgent actions has been properly defined. As a result, the expressivity and semantics of the abstract language have been considerably improved.

Keywords: Modeling, Automata, Hybrid Systems, Formal Semantics.

1. INTRODUCTION

The main purpose of the Compositional Interchange Format (CIF), that has originally been developed in HYCON, see HYCON Network of Excellence (2005) and Beek et al. (2007b,a, 2008b), is to establish inter-operability of a wide range of tools by means of model transformations to and from the CIF. In addition, the CIF provides a generic modeling formalism and tools for a wide range of untimed, timed and hybrid systems. Fig. 1 gives an overview of work on the CIF in different projects. In the EU FP7 project Multiform, see MULTIFORM consortium (2008), bidirectional transformations between the CIF and several languages/tools are developed. In the EU FP7 project C4C, see C4C consortium (2008), work on the CIF is mainly on compositional verification, whereas in the EU ITEA2 Twins project, see ITEA Twins consortium (2009), the CIF is connected to tools for supervisory control synthesis. The CIF is used in several industrial projects. Some examples are presented in italics in Fig. 1. Not represented in Fig. 1 is work in the EU FP7 project DISC, see DISC consortium (2009), on possible connections between the CIF and Petri nets. For an overview on previous related work on interchange formalisms, such as found in MoBIES team (2002), Pinto et al. (2006), Cairano et al. (2006), we refer to Beek et al. (2007b,a).

⋆ This work was partially done as part of the European Community's Seventh Framework Programme (FP7/2007-2013) projects MULTIFORM and C4C, contract numbers FP7-ICT-224249 and FP7-ICT-223844, respectively, as part of the European Community's EUREKA cluster program ITEA 2 project Twins 05004, and as part of the Darwin project under the responsibility of the Embedded Systems Institute, partially supported by the Netherlands Ministry of Economic Affairs under the BSIK program.

The CIF consists of an *abstract* format, which is specified using mathematical notation and is used for the definition of the formal semantics and analysis system properties, and a *concrete* format, as defined in Beek et al. (2008b), which is specified in the ASCII character set by a formal grammar and is used as a modeling language. The semantics of a model in the concrete format is formally defined by means of a mapping to the abstract format. The advantage of having two formats is that each can be tailored to its specific purpose. In general, the abstract format has fewer concepts in order to simplify the semantics, while the concrete format has 'syntactic sugar' and more emphasis on backward compatibility in order to facilitate modeling.

In this article, the abstract format is redefined. The main changes are as follows: variables are introduced using scoping operators; the abstract language is made more orthogonal by providing an operator for each concept in the language; parallel composition has been defined in such a way that compositional verification (assume/guarantee reasoning) is supported; and urgency of actions has been redefined.

The remainder of this article is organized as follows. Section 2 defines the syntax along with an informal description of the semantics; Section 3 presents the new concepts in the abstract format, together with the most important deduction rules, and Section 4 presents concluding remarks.

2. REVISED SYNTAX AND INFORMAL SEMANTICS

2.1 Syntax

An atomic interchange automaton α_{atomic} is a tuple

$$\alpha_{\text{atomic}} = (V, v_0, \text{flow}, \text{inv}, \text{tcp}, E),$$

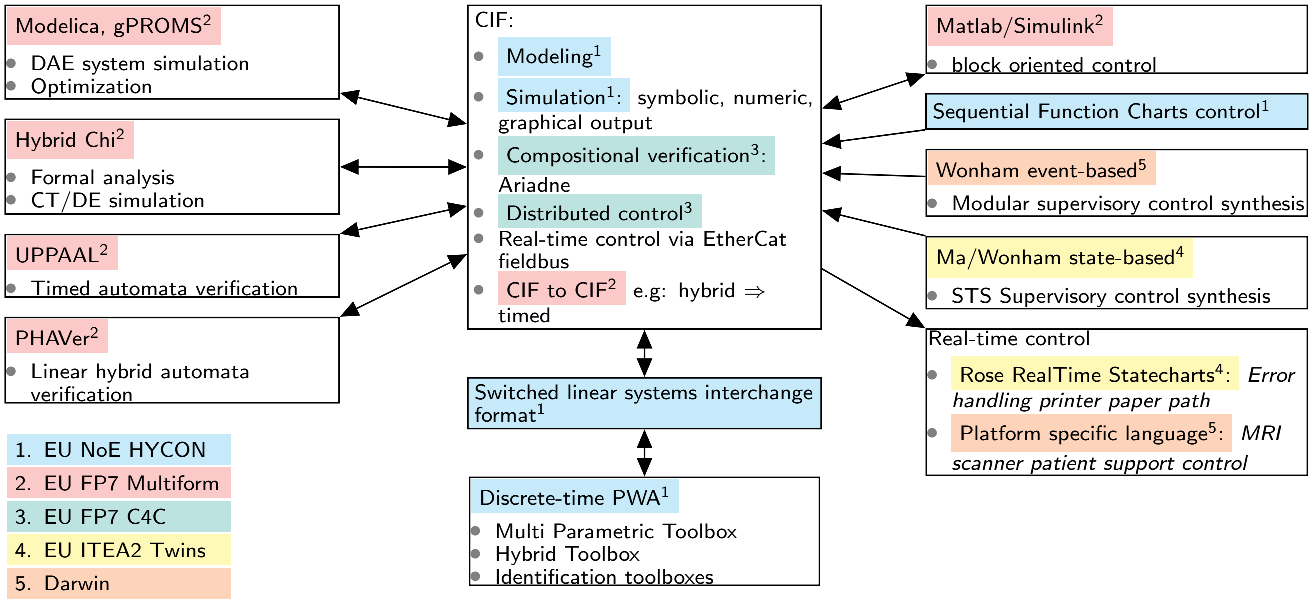

Fig. 1. Overview of work on the CIF

where V is a set of locations (vertices); $v_0 \in V$ is the initial location, also called active location; flow, inv, tcp are functions of the set $V \to \mathrm{P}(\dot{\mathcal{V}})$, which associate to each location a predicate describing the flow condition, the invariant, and the urgency condition respectively. Here, $\mathcal{V}$ denotes the set of all variables, $\dot{\mathcal{V}} = \{\dot{x} \mid x \in \mathcal{V}\} \cup \mathcal{V}$ denotes the set of variables plus their dotted versions, and $\mathrm{P}(X)$ denotes the set of all predicates over the variables from set X;

$$E \in V \times \mathrm{P}(\dot{\mathcal{V}}) \times \mathcal{L}_{\text{basic}} \cup \mathrm{CE}(\dot{\mathcal{V}}) \times (2^{\dot{\mathcal{V}}} \times \mathrm{P}(\dot{\mathcal{V}} \cup \dot{\mathcal{V}}^{+})) \times V$$

denotes the discrete transitions (edges) of the automaton, where $\mathcal{L}_{\text{basic}}$ denotes the set of basic action labels, $\mathrm{CE}(\dot{\mathcal{V}})$ denotes the set of all send and receive labels, formally defined as $\mathrm{CE}(\dot{\mathcal{V}}) = \{h!es, h?xs \mid h \in \mathcal{H} \wedge es \in \mathrm{Expr}(\dot{\mathcal{V}})^* \wedge xs \in \mathcal{V}^*\}$ where X^* denotes the set of lists whose elements are taken from set X, $\mathrm{Expr}(X)$ denotes the set of all expressions over variables from set X, and $\mathcal{H}$ is the set of all channels, 2^X denotes the powerset of X, and $X^+ = \{v^+ \mid v \in X\}$ is used to refer to the values of the variables after the execution of an action.

The set of interchange automata $\mathcal{A}$ is defined by the following grammar:

$\alpha ::= \alpha_{atom}$	Atomic interchange automaton
$\mid \alpha \parallel \alpha$	Parallel composition operator
$\mid \gamma_a(\alpha)$	Synchronizing action operator
$\mid u \gg \alpha$	Initialization operator
$\mid [\![_{\mathrm{V}}\, x = e_0, \dot{x} = e_1 :: \alpha]\!]$	Variable scope operator
$\mid [\![_{\mathrm{A}}\, a :: \alpha]\!]$	Action scope operator
$\mid [\![_{\mathrm{H}}\, h :: \alpha]\!]$	Channel scope operator
$\mid \partial_h(\alpha)$	Channel encapsulation operator
$\mid \mathrm{st}_x(\alpha)$	State variable operator
$\mid \mathrm{own}_x(\alpha)$	Ownership operator
$\mid \mathrm{U}_w(\alpha)$	Urgent action operator
$\mid \mathrm{D}_{x:G}(\alpha)$	Dynamic type operator

where $a \in \mathcal{L}_{\text{basic}}$ is a basic action label; $u \in \mathrm{P}(\dot{\mathcal{V}})$ is a predicate over variables and dotted variables; x is a variable, and $e \in \{\perp\} \cup \mathrm{Expr}(\dot{\mathcal{V}})$ is an expression, where $\perp$ denotes the undefined value; $h \in \mathcal{H}$ is a channel; $w \in \mathcal{L}_{\text{basic}} \cup \mathcal{H}$ is a basic action label or a channel; and G is a set of solutions (see Section 3.2 for more detail on solutions).

2.2 *Informal semantics*

The semantics of an interchange automaton is defined in terms of action transitions and time transitions between states consisting of the interchange automaton itself, a valuation of the variables and dotted variables of the automaton, and a set of state variables. These variables are the only ones that are not allowed to change arbitrarily in action transitions. By default all variables can change after the execution of an action.

The edges of the *atomic interchange automaton* are used to specify which actions can be executed. An edge $(v, g, l, (W, r), v')$ can be chosen to perform an action in a given valuation σ if v is the active location; the predicates g, $\mathrm{flow}(v)$, and $\mathrm{inv}(v)$ are satisfied in σ; and it is possible to find a new valuation σ' in which the predicates r, $\mathrm{flow}(v')$, and $\mathrm{inv}(v')$ are satisfied. After the execution of the action specified by this edge, the active location is updated to v'.

Time can pass in an active location v as long as the predicates $\mathrm{flow}(v)$, $\mathrm{inv}(v)$, and $\mathrm{tcp}(v)$ hold, and no urgent actions become enabled. Section 3.2 describes how model variables are allowed to change when time passes. The active location is not modified.

Parallel composition operator $\alpha_0 \parallel \alpha_1$ allows to execute two automata in parallel, which can interact by means of synchronizing actions, communication of values via channels, and shared variables. By default the actions of α_0 and α_1 are interleaved.

Synchronizing action operator $\gamma_a(\alpha)$ specifies that actions a have to be synchronized with other actions named a that were declared as synchronizing in a parallel context.

Initialization operator $u \gg \alpha$ restricts the initial valuations in which α can start its execution to those which satisfy u.

Variables, actions, and channels can be introduced using *scope operators*, which make the identifiers being declared invisible outside of the scope. In particular, the term $[\![_{\mathrm{V}}\, x = e_0, \dot{x} = e_1 :: \alpha]\!]$ declares a local variable x (and associated 'derivative' $\dot{x}$), the initial value of which is given by the expression e_0 (e_1), unless $e_0 = \bot$ ($e_1 = \bot$) in which case x ($\dot{x}$) can have any value.

Channel encapsulation operator $\partial_h(\alpha)$ forces send and receive actions via channel h to execute synchronously, by blocking their individual occurrences.

State variable operator $\text{st}_x(\alpha)$ declares variable x as a state variable for automaton α, which means that x cannot change arbitrarily in action transitions. In other words, if α does not change x explicitly, for instance by means of an assignment, its value remains the same after the execution of an action.

Ownership operator $\text{own}_x(\alpha)$, declares the variable x as belonging to automaton α. This means that only α can change the value of x. Other automata cannot change the value of x, unless this change is allowed by α in a synchronizing action.

Urgent action operator $\text{U}_w(\alpha)$ declares action w (or the send/receive actions via w iff w is a channel) as urgent. This means that time passing is allowed iff action w is not enabled during the time interval.

2.3 Description of Hybrid Transition System

The structured operational semantics (SOS) of the CIF associates a labeled transition system (LTS) with every interchange automaton. There are three kind of transitions: action transitions, time transitions, and environment transitions.

An *action transition* $(\alpha, \sigma, X) \xrightarrow{l,A} (\alpha', \sigma', X')$ models the execution of an action with label l, starting in state (α, σ, X), and resulting in a new state (α', σ', X'). The set A contains the set of synchronizing actions of α, and the sets X and X' contain the set of state variables (see section 3.2) of α and α', respectively.

A *time transition* $(\alpha, \sigma, X) \overset{t,\rho,\theta}{\longmapsto} (\alpha', \sigma', X')$ models the passing of t time units, starting in state (α, σ, X), and resulting in a new state (α', σ', X'). Function ρ contains for each variable the trajectories on time interval $[0, t]$, and function θ contains for each action the trajectory of the guard(s) associated with this action on time interval $[0, t]$.

A *environment transition* $(\alpha, \sigma, X) \overset{A}{\dashrightarrow} (\alpha, \sigma', X')$ states that 1) σ (σ') is consistent with the initial conditions and active invariants of α (α'); 2) the values of the variables owned by α remain unchanged; and 3) the set of synchronizing actions of α is A.

3. NEW CONCEPTS

This section presents the new concepts of the abstract CIF, along with the most important deduction rules. We have chosen to structure this section based on the concepts, introducing the most important deduction rules only. Furthermore, examples are specified using some abuse of notation. For example, an assignment to a variable x on an edge of an automaton can be specified as $x := 1$, thus omitting the locations, guards and action labels.

3.1 Introducing variables, actions, and channels

In the concrete format, variables are introduced by means of *closed scopes* and *open scopes*. Both concepts of scoping can be found in modeling languages. In the abstract format, as defined in Beek et al. (2007b), however, the notion of an interchange automaton was formalized using concepts from hybrid automaton theory (e.g. see Alur et al. (1995)). This means that variables, action labels and channels were defined in interchange automata, and hiding operators were used instead of scoping operators.

This mismatch in the way variables were introduced in the abstract and concrete level added considerable complexity to the function that mapped the concrete CIF format to the abstract format. In the new CIF, the sets of internal and external variables have been removed from the atomic automata. External variable are introduced in the valuations, and internal variables are introduced using scope operators.

3.2 Action and delay behavior of variables

In the concrete format of the CIF, variables are declared as discrete, continuous or algebraic. Such a declaration defines the behavior of variables in the following way.

For actions, the value of a discrete or continuous variable does not change in an action transition unless it is explicitly specified to be allowed to change. E.g, incrementing the value of variable x by one is expressed on an edge of an automaton by $\{x\} : x^+ = x + 1$. The set of variables that is allowed to change is $\{x\}$, and in predicate $x^+ = x + 1$, x and x^+ refer to the values of x before and after the transition, respectively. All other discrete and continuous variables *are not* allowed to change, and all other algebraic variables *are* allowed to change.

For delays, the value of a discrete variable is a constant function; continuous variables change according to a continuous function such that the solution function for x is the integral of the solution function for $\dot{x}$ (and $\dot{x}$ is the derivative of x, if such a derivative exists); algebraic variables may behave according to a discontinuous function.

To allow an improved separation of concerns, the semantics of the discrete, continuous and algebraic variables is defined by two operators in the abstract CIF: the state variable operator defines the action behavior, and the dynamic type operator defines the delay behavior. Furthermore, the ownership operator own() provides new functionality: it prevents automata from making changes to variables that are owned by other automata.

State variable operator Application of the state variable operator $\text{st}_x(\alpha)$ adds x to the set of state variables X, thus preventing changes in action transitions. This is formalized in rule 1:

$$\frac{(\alpha, \sigma, X \cup \{x\}) \xrightarrow{a,A} (\alpha', \sigma', X')}{(\text{st}_x(\alpha), \sigma, X) \xrightarrow{a,A} (\text{st}_x(\alpha), \sigma, X)} \; 1$$

In principle, the translation function from the concrete CIF format to the abstract format ensures that the state variable operator is applied for the discrete and continuous variables.

Ownership operator Formally, this operator only affects environment transitions. Here, the variable that is declared as owned by the automaton is not allowed to change. This is used in parallel composition, see Section 3.5.

$$X, X' \Vdash \frac{(\alpha, \sigma) \overset{A}{\dashrightarrow} (\alpha', \sigma'),\ \sigma(x) = \sigma'(x)}{(\text{own}_x(\alpha), \sigma) \overset{A}{\dashrightarrow} (\text{own}_x(\alpha'), \sigma')} \; 2$$

The difference between the ownership operator and the state variable operator is subtle; the state variable operator prevents a variable from changing during an action defined within the automaton itself, whereas the ownership operator prevents a variable from jumping during an action defined within another automaton.

Dynamic type operator The dynamic type operator $\mathrm{D}_{x:G}(\alpha)$ has no effect on action and environment transitions. For time transitions, the operator ensures that the trajectories for the variable x and the dotted version of the variable $\dot{x}$ is restricted to the behavior that is specified by the set of pairs of solution functions G. E.g. for a discrete variable x, the set G would consist of pairs of a constant function as first element, and a zero function as second element. For a continuous variable, for each pair, the second function would be the derivative of the first (if existing), and the first would be the integral of the second.

$$X, X' \Vdash \frac{(\alpha, \sigma) \overset{t,\rho,\theta}{\longmapsto} (\alpha', \sigma'),\ (\rho \downarrow x, \rho \downarrow \dot{x}) \in G}{(\mathrm{D}_{x:G}(\alpha), \sigma) \overset{t,\rho,\theta}{\longmapsto} (\mathrm{D}_{x:G}(\alpha'), \sigma')}\ 3$$

3.3 Initialization

In the previously published abstract CIF, initialization was specified by means of the init predicate of an atomic interchange automaton. The init predicate was also used to store the values of local variables. The initialization predicate that was present in the 'old' interchange automata is specified by means of the initialization operator in the new abstract CIF, and the value of each local variable is stored in a corresponding variable scope operator. To capture the fact that all initialization predicates are taken into account simultaneously, as is the case in hybrid automata, the following properties should hold: $u \gg (u' \gg \alpha) \leftrightarrow u \wedge u' \gg \alpha$ and $(u \gg \alpha) \parallel (u' \gg \alpha') \leftrightarrow u \wedge u' \gg (\alpha \parallel \alpha')$.

To ensure that the initialization predicate is removed after it has been taken into account, the deduction rules are:

$$X, X' \Vdash \frac{(\alpha, \sigma) \xrightarrow{a,A} (\alpha', \sigma'),\ \sigma \models u}{(u \gg \alpha, \sigma) \xrightarrow{a,A} (\alpha', \sigma')}\ 4$$

$$X, X' \Vdash \frac{(\alpha, \sigma) \overset{A}{\dashrightarrow} (\alpha', \sigma'),\ \sigma \models u}{(u \gg \alpha, \sigma) \overset{A}{\dashrightarrow} (\alpha', \sigma')}\ 5$$

and likewise for time transitions. Rule 8 for the parallel composition ensures that when automaton α_0 executes an action, the initialization predicates of automaton α_1, executing in parallel, are also taken into account. Furthermore, the resulting (right hand side of the transition) parallel composition of the automata in the conclusion is constructed as the parallel composition of the resulting automata in the premises, to ensure that the initialization predicates are removed after they have been taken into account. Note that the environment transitions avoid the need for functions that depend on the syntax of the automata, which may endanger compositionality of the semantics.

3.4 Synchronization

In the previously defined CIF semantics, actions with the same name were synchronizing by default. However, common practice has shown that this is inconvenient, see Theunissen et al. (2008, 2009). This is illustrated by means of the following example. Consider motor M_0 with two controllers C_0 and C_1. Controller C_0 stops the motor if it detects a low oil pressure. Similarly, controller C_1 stops the motor if it detects overheating. The motor and controllers can be modeled separately by the automata specified in Fig. 2. Using the previously defined semantics, the motor can be stopped only if an abnormal event is detected by both controllers, since the three automata synchronize on the 'stop' event.

In the revised CIF semantics, actions having the same name are non-synchronizing by default. If synchronization is intended, this can be achieved by using the synchronization operator $\gamma()$. Informally, the automaton $\gamma_a(\alpha)$ declares the action a as synchronizing, which means the the execution of action a in α synchronizes with other automata that have also declared a as synchronizing. Therefore, in $\gamma_a(\alpha_0) \parallel \gamma_a(\alpha_1)$ action a must be executed simultaneously in both automata. If, on the other hand, in $\gamma_a(\alpha_0) \parallel \alpha_1$, action a is not declared as synchronizing in α_1, then execution of actions a in both automata will be interleaved. The desired behavior in the example represented in Fig. 2 can be specified in the new CIF by $\gamma_{\{stop\}}(Motor) \parallel \gamma_{\{stop\}}(Controller_0 \parallel Controller_1)$.

Note that synchronization by default, as defined in the previous CIF semantics, can be modeled in the revised abstract language by enclosing every atomic automaton by appropriate $\gamma()$ operators. The converse is, however, not true. Therefore, the revised abstract format is more expressive than the previously defined one.

The semantics of the synchronization operator is defined using the set of action labels from the action and environment transitions. Thus, the effect of the operator $\gamma_a(\alpha)$ is to add the action a to this set, as shown in the following rule:

$$X, X' \Vdash \frac{(\alpha, \sigma) \xrightarrow{l,A} (\alpha', \sigma')}{(\gamma_a(\alpha), \sigma) \xrightarrow{l,A\cup\{a\}} (\gamma_a(\alpha'), \sigma')}\ 6$$

In the SOS rules for parallel composition, the set of synchronizing actions (on the action and environment transitions) is used to determine whether an action is synchronizing.

3.5 Parallel composition

Consider a parallel composition of two automata, each with two locations connected by means of one edge, with a shared synchronizing action a, and two shared variables x and y. The automata assign the value 1 to the variables x and y, respectively. With some abuse of notation this can be specified as $\gamma_a(a : \{x\} : x^+ = 1) \parallel \gamma_a(a : \{y\} : y^+ = 1)$. In the previously

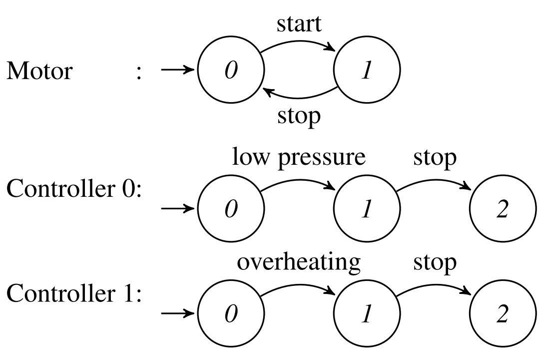

Fig. 2. Motor and controller automata

published semantics of the CIF this was equivalent to one automaton with one edge: $\gamma_a(a : \{x, y\} : x^+ = 1 \wedge y^+ = 1)$. In more general terms: $\gamma_a(a : W : r) \parallel \gamma_a(a : W' : r')$ was equivalent (bisimilar) to $\gamma_a(a : W \cup W' : r \wedge r')$. This functionality was achieved by means of a set of jumping variables W on action transitions, and by means of injecting the set of jumping variables of an action of one automaton into the environment set of jumping variables of the other automaton, see Beek4:RevHSIntFormatD363Tech2007.

In the new semantics, parallel composition is strictly a restriction for synchronizing behavior. This means that there is a difference between the behavior of state variables (discrete and continuous variables) and the other variables (algebraic variables) in the example: $\gamma_a(a : \{x\} : x^+ = 1) \parallel \gamma_a(a : \{y\} : y^+ = 1)$ cannot do any of the two actions if x or y is a state variable (or if they both are state variables).

In more general terms, we get the following equivalence: $\gamma_a(a : W : r) \parallel \gamma_a(a : W' : r') \;\underline{\leftrightarrow}\; \gamma_a(a : W \cap W' : r \wedge r')$. This means that for state variables x and y, the example should be modeled using the new semantics as: $\gamma_a(a : \{x, y\} : x^+ = 1) \parallel \gamma_a(a : \{x, y\} : y^+ = 1)$, allowing both variables to change in both processes; or as: $\gamma_a([\![_{\mathrm{v}}\, x = \bot :: \mathrm{st}_x(a : \{x\} : x^+ = 1)\,]\!]) \parallel \gamma_a([\![_{\mathrm{v}}\, y = \bot :: \mathrm{st}_y(a : \{y\} : y^+ = 1)\,]\!])$, defining both variables as local.

In the new semantics, the set of jumping variables has been removed from the action transitions. Synchronizing action behavior is specified by means of the following rule:

$$X, X' \Vdash \frac{a \in A \cap B,\quad (\alpha_0, \sigma) \xrightarrow{a,A} (\alpha_0', \sigma'),\ (\alpha_1, \sigma) \xrightarrow{a,B} (\alpha_1', \sigma')}{(\alpha_0 \parallel \alpha_1, \sigma) \xrightarrow{a,A\cup B} (\alpha_0' \parallel \alpha_1', \sigma')} \quad 7$$

Clearly, the parallel composition is now the intersection of the behaviors of the synchronizing components: changes in the valuations are allowed only if allowed by both partners of the parallel composition.

For non-synchronizing behavior, the partners of a parallel composition also need to agree on changes in the values of variables. In this case, however, the non-synchronizing partners do an environment transition. An environment transition does not allow any changes in the values of the 'owned' variables, see Rule 2. It allows all other variables to change arbitrarily, as long as the active invariants and equations remain satisfied. This is expressed by the following rule for interleaving parallel composition:

$$X, X' \Vdash \frac{a \notin A \cap B,\quad (\alpha_0, \sigma) \xrightarrow{a,A} (\alpha_0', \sigma'),\ (\alpha_1, \sigma) \overset{B}{\dashrightarrow} (\alpha_1', \sigma')}{(\alpha_0 \parallel \alpha_1, \sigma) \xrightarrow{a,A\cup B} (\alpha_0' \parallel \alpha_1', \sigma')} \quad 8$$

Consider the example $x = 0 \gg x := 1 \parallel \mathrm{own}_x(a)$, where a denotes a (non-synchronizing) action label. The ownership operator prevents changes to variable x by parallel components, as defined by Rules 2 and 8. Therefore, the assignment to x cannot be executed. When variable x is defined as owned at the top level, as in $\mathrm{own}_x(x = 0 \gg x := 1 \parallel a)$, the assignment to x *can* be executed.

The new syntax and semantics facilitates compositional verification, because the ownership operator can be applied to an automaton α in a parallel composition (e.g. $\mathrm{own}_x(\alpha) \parallel \alpha'$) to prevent changes to its state variable x by parallel automaton α'.

3.6 Urgency

Urgent actions were introduced in timed and hybrid formalisms to allow easy modeling of greedy, or eager, behavior. In the CIF, the passing of time can be restricted by means of the tcp predicate, or by means of the urgent action operator $\mathrm{U}_a(\alpha)$ that declares action a to be urgent.

The following example shows a relation between the tcp predicate and the urgent action operator. Let $\alpha = (\{v\}, v, \{v \mapsto \mathrm{true}\}, \{v \mapsto \mathrm{true}\}, \{v \mapsto \mathrm{true}\}, \{e\})$ denote an atomic interchange automaton consisting of a single location v with invariant, flow, and tcp predicates all true, and one self loop edge e with guard $\mathrm{time} \geq 1$ and action label a: $e = (v, \mathrm{time} \geq 1, a, (\emptyset, \mathrm{true}), v)$. Then defining the action a as urgent by application of the urgent action operator $\mathrm{U}_a(\alpha)$ is equivalent to the automaton α' obtained from α by replacing the tcp predicate by the negation of the guard: $\alpha' = (\{v\}, v, \{v \mapsto \mathrm{true}\}, \{v \mapsto \mathrm{true}\}, \{v \mapsto \neg(\mathrm{time} \geq 1)\}, \{e\})$. However, in the case that an action a synchronizes in a parallel context, it is not possible to express urgency of such an action using tcp predicates in a compositional way.

The urgent action operator applied to a parallel composition of two automata $\mathrm{U}_a(\alpha_0 \parallel \alpha_1)$ aims to express that time can pass only for as long as no action a becomes enabled in $\alpha_0 \parallel \alpha_1$. For a synchronizing action a to become enabled, the guards of the action a need to be true in both automata α_0 and α_1. For a non-synchronizing action a to become enabled, it is sufficient that a guard in one of the automata α_0 or α_1 is enabled. Note that time cannot pass anymore when an urgent action becomes enabled, independently of whether such an action can actually be executed. If an invariant of a target location would be false, then deadlock could be the result.

The required semantics is obtained by augmenting the time transitions by a pair of *guard trajectories* $(\theta_{\mathrm{y}}, \theta_{\mathrm{n}})$, as defined in the hybrid Chi formalism (see Beek et al. (2008a)). Namely θ_{y} for the synchronizing actions, and θ_{n} for the non-synchronizing actions. Note that for the purpose of simplicity, urgent channels are not discussed here. A guard trajectory is defined as a mapping from time-points to guard valuations: $\theta_{\mathrm{y}}, \theta_{\mathrm{n}} \in [0, t] \to ((\mathcal{L}_{\mathrm{basic}} \cup \{\tau\}) \rightharpoonup \mathbb{B})$, and for all $s \in [0, t]$, $a \in \mathrm{dom}(\theta_{\mathrm{x}}(\mathrm{s}))$, $x \in \{\mathrm{y}, \mathrm{n}\}$: $\theta_x(s)(a)$ iff the action a is enabled (the value of the associated guard is true) at time s.

Guard trajectories are defined in the time transition rule of the atomic interchange automaton:

$$\frac{\forall_{s\in[0,t]}\rho(s) \models f(v_0) \wedge i(v_0),\quad \forall_{s\in[0,t)}\rho(s) \models c(v_0),\ \mathrm{dom}(\rho) = [0, \mathrm{t}],\ \rho(0) = \sigma,\ \mathrm{t} > 0}{((V, v_0, f, i, c, E), \sigma, X) \overset{t,\rho,\theta}{\longmapsto} ((V, v_0, f, i, c, E), \rho(t), X)} \quad 9$$

where $\theta = (\theta_{\mathrm{y}}, \theta_{\mathrm{n}})$ and for all time points $s \in [0, t]$ and actions $a \in \mathcal{L}_{\mathrm{basic}}$, θ_{y} and θ_{n} are defined as

$$\mathrm{dom}(\theta_{\mathrm{y}}(\mathrm{s})) = \emptyset$$

$$\theta_{\mathrm{n}}(s)(a) = \rho(s)(\vee_{g:g\in\{g|(v_0,g,l,(W:r),v')\in E\}}\, g)^{1}.$$

To be robust for variable abstraction, the *value* $\rho(s)(g)$ of guard g is used in the guard valuation instead of the guard expressions itself. If one would choose to use the guard expressions,

[1] Note that $\vee_{g:g\in\emptyset}\, g$ denotes the predicate false

abstraction from variables may unexpectedly change the way in which actions are enabled and disabled, and hence may change the nature of the urgent behavior of those actions.

The urgent action operator $\mathrm{U}_a(\alpha)$ allows for time passing as long as the guard valuation of the action a remains false.

$$\frac{\begin{array}{c}(\alpha, \sigma, X) \overset{t,\rho,\sigma}{\longmapsto} (\alpha', \sigma', X'),\\ \forall_{s\in[0,t)}(a \in \mathrm{dom}(\theta_{\mathrm{y}}(\mathrm{s})) \Rightarrow \neg\theta_{\mathrm{y}}(\mathrm{s})(\mathrm{a}),\\ a \in \mathrm{dom}(\theta_{\mathrm{n}}(\mathrm{s})) \Rightarrow \neg\theta_{\mathrm{n}}(\mathrm{s})(\mathrm{a}))\end{array}}{(\mathrm{U}_a(\alpha), \sigma, X) \overset{t,\rho,\sigma}{\longmapsto} (\mathrm{U}_a(\alpha'), \sigma', X')} \quad 10$$

The urgent action operator does not affect the action and environment behavior of its operand.

The difference between guard trajectories of synchronizing actions and non-synchronizing actions becomes clear in the time transition rule of the parallel composition operator.

$$\frac{\begin{array}{c}X, X' \Vdash\\ (\alpha_0, \sigma) \overset{t,\rho,(\theta_{\mathrm{y}0},\theta_{\mathrm{n}0})}{\longmapsto} (\alpha_0, \sigma'),\ (\alpha_1, \sigma) \overset{t,\rho,(\theta_{\mathrm{y}1},\theta_{\mathrm{n}1})}{\longmapsto} (\alpha_1, \sigma')\end{array}}{(\alpha_0 \parallel \alpha_1, \sigma) \overset{t,\rho,(\theta_{\mathrm{y}0}\wedge\theta_{\mathrm{y}1},\theta_{\mathrm{n}0}\vee\theta_{\mathrm{n}1})}{\longmapsto} (\alpha_0 \parallel \alpha_1, \sigma')} \quad 11$$

For synchronizing actions, the resulting guard trajectory is defined as the *conjunction* of the guard trajectories $\theta_{\mathrm{y}0}$ and $\theta_{\mathrm{y}1}$, whereas for non-synchronizing actions, the resulting guard trajectory is defined as the *disjunction* of the guard trajectories $\theta_{\mathrm{n}0}$ and $\theta_{\mathrm{n}1}$.

Finally, for the synchronizing action operator $\gamma_a(\alpha)$ we can informally state that it 'moves' the guard trajectory for action a from θ_{n} to θ_{y}, resetting the guard trajectory for a in θ_{n} to false.

4. CONCLUDING REMARKS

The main advantages of the redesign of the abstract format are as follows: the abstract and concrete languages are now based on similar concepts so that the transformation of the concrete language to the abstract language will be straightforward; the different concepts in the abstract language are easier to identify and understand, because each concept is defined by an operator in an orthogonal way; compositional verification (assume/guarantee reasoning) is facilitated; urgent actions are properly defined, and the SOS rules have been considerably simplified. We expect that the work on the CIF in the different (EU) projects will benefit accordingly. As a final remark, please note that all deduction rules of the abstract CIF satisfy the *process-tyft* format of Mousavi et al. (2005). Therefore, stateless bisimilarity is a congruence for all operators.

ACKNOWLEDGEMENTS

The authors thank Pieter Cuijpers, Jasen Markovski, and Michel Reniers for their contributions in developing the abstract CIF, and they thank the anonymous reviewers for helpful comments on the draft of this article.

REFERENCES

Alur, R., Courcoubetis, C., Halbwachs, N., Henzinger, T.A., Ho, P.H., Nicollin, X., Olivero, A., Sifakis, J., and Yovine, S. (1995). The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138(1), 3–34.

Beek, D.A.v., Hofkamp, A.T., Reniers, M.A., Rooda, J.E., and Schiffelers, R.R.H. (2008a). Syntax and formal semantics of Chi 2.0. SE Report 2008-01, Eindhoven University of Technology, Systems Engineering Group, Department of Mechanical Engineering, Eindhoven, The Netherlands. URL `http://se.wtb.tue.nl/sereports`.

Beek, D.A.v., Reniers, M.A., Rooda, J.E., and Schiffelers, R.R.H. (2007a). Revised hybrid system interchange format. Technical Report HYCON Deliverable D3.6.3, HYCON NoE.

Beek, D.A.v., Reniers, M.A., Rooda, J.E., and Schiffelers, R.R.H. (2008b). Concrete syntax and semantics of the compositional interchange format for hybrid systems. In *17th Triennial World Congress of the International Federation of Automatic Control*, 7979–7986. Seoul, Korea.

Beek, D.A.v., Reniers, M.A., Schiffelers, R.R.H., and Rooda, J.E. (2007b). Foundations of an interchange format for hybrid systems. In A. Bemporad, A. Bicchi, and G. Butazzo (eds.), *Hybrid Systems: Computation and Control, 10th International Workshop*, volume 4416 of *Lecture Notes in Computer Science*, 587–600. Springer-Verlag, Pisa.

C4C consortium (2008). Control for coordination of distributed systems. http://www.c4c-project.eu/.

Cairano, S.D., Bemporad, A., and Kvasnica, M. (2006). An architecture for data interchange of switched linear systems. Technical Report D 3.3.1, HYCON NoE.

DISC consortium (2009). Distributed supervisory control of large plants. http://www.disc-project.eu/.

HYCON Network of Excellence (2005). http://www.ist-hycon.org/.

ITEA Twins consortium (2009). Optimizing HW-SW co-design flow for sofware intensive system development. http://www.twins-itea.org/.

MoBIES team (2002). HSIF semantics. Technical report, University of Pennsylvania. Internal document.

Mousavi, M.R., Reniers, M.A., and Groote, J.F. (2005). Notions of bisimulation and congruence formats for SOS with data. *Information and Computation*, 200(1), 107–147.

MULTIFORM consortium (2008). Integrated multi-formalism tool support for the design of networked embedded control systems MULTIFORM. http://www.multiform.bci.tu-dortmund.de.

Pinto, A., Carloni, L.P., Passerone, R., and Sangiovanni-Vincentelli, A.L. (2006). Interchange format for hybrid systems: Abstract semantics. In J.P. Hespanha and A. Tiwari (eds.), *Hybrid Systems: Computation and Control, 9th International Workshop*, volume 3927 of *Lecture Notes in Computer Science*, 491–506. Springer-Verlag, Santa Barbara.

Theunissen, R.J.M., Schiffelers, R.R.H., Beek, D.A.v., and Rooda, J.E. (2008). Supervisory control synthesis for a patient support system. SE Report 2008 – 08, Eindhoven University of Technology, Systems Engineering Group, Department of Mechanical Engineering, Eindhoven, The Netherlands. URL `http://se.wtb.tue.nl/sereports`.

Theunissen, R.J.M., Schiffelers, R.R.H., Beek, D.A.v., and Rooda, J.E. (2009). Supervisory control synthesis for a patient support system. In *Proceedings of the European control conference*. Budapest, Hungary.