A Compositional Interchange Format for Hybrid Systems

Ramon Schiffelers
joint work with Bert van Beek, Michel Reniers, Koos Rooda

Systems Engineering Group
Dept. Mechanical Engineering

and

Formal Methods Group
Dept. Mathematics and Computer Science
Eindhoven University of Technology
The Netherlands

May 8, 2008
Contents

• Foundations of the CIF

• The CIF language
  – Abstract format
  – Concrete format
  – Transfer format

• Current status of the CIF (available translations, tools)

• Future of the CIF
Purpose of the CIF

- Establish inter-operability of a wide range of tools by means of model transformations to and from the CIF.
  
  Avoid the implementation of many bi-lateral translators between specific formalisms.

- Provide a generic modeling formalism and tools (such as a simulator) for a wide range of hybrid systems.
Examples of translations

Without the CIF:

<table>
<thead>
<tr>
<th>Language A</th>
<th>Language B</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_0$</td>
<td>$B_0$</td>
</tr>
<tr>
<td>$A_1$</td>
<td>$B_1$</td>
</tr>
<tr>
<td>$A_2$</td>
<td>$B_2$</td>
</tr>
<tr>
<td>$A_3$</td>
<td>$B_3$</td>
</tr>
</tbody>
</table>
Examples of translations

With the CIF:

- Language $A_0$
- Language $A_1$
- Language $A_2$
- Language $A_3$
- CIF
- Language $B_0$
- Language $B_1$
- Language $B_2$
- Language $B_3$
Examples of applying the CIF

- Simulate Language $S$
- CIF
- Verify Model checking language $M_1$
- Model checking language $M_2$
Examples of applying the CIF
Requirements of the CIF

- Formal and compositional semantics, based on (hybrid) transition systems, allowing property preserving model transformations.

- Concepts based on mathematics, independent of implementation aspects such as equation sorting, and numerical equation solving algorithms.

- 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.

- Support hierarchy and modularity to allow 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.
Requirements of the CIF (cnt.)

- Support a wide range of urgency concepts, such as used in hybrid automata, including
  - urgency predicates
  - deadline predicates
  - triggering guard semantics
  - urgent actions.

- Support parallel composition with interaction by means of
  - CSP-like communication by means of channels
  - synchronization by means of shared actions
  - shared variables.
Three formats of the CIF

Abstract format: Facilitates mathematical definition of the formal semantics.

Concrete format: Provides user friendly syntax for the language elements of the abstract format
Suited for modeling directly in the CIF.
Extends the abstract format with (possibly non-orthogonal) language elements

Transfer format: Facilitates the file generation and parsing process. E.g. XML format, supported by means of libraries in many languages.
Syntax of the abstract format of the CIF

\[ \alpha ::= \alpha_{\text{atom}} \quad \text{atomic interchange automaton where} \]

<table>
<thead>
<tr>
<th>\alpha \parallel \alpha \quad \text{parallel composition}</th>
</tr>
</thead>
<tbody>
<tr>
<td>hide_{\text{var}}(X_h, \alpha, \sigma_h) \quad \text{variable hiding operator}</td>
</tr>
<tr>
<td>hide_{\text{act}}(L_h, \alpha) \quad \text{action hiding operator}</td>
</tr>
<tr>
<td>urgent(L_u, \alpha) \quad \text{urgent action operator}</td>
</tr>
<tr>
<td>encap(L_e, \alpha) \quad \text{action encapsulation operator},</td>
</tr>
</tbody>
</table>

- \( L_h \): actions to hide,
- \( X_h \): variables to hide,
- \( L_u \): urgent actions,
- \( L_e \): actions to encapsulate.
Syntax atomic interchange automaton

\[ \alpha_{\text{atom}} ::= (X, X_i, \text{dtype}, V, v_0, \text{init}, \text{flow}, \text{inv}, \text{tcp}, L, E). \]

- \( X \): all variables, \( X_i \) internal variables.
- \( \text{dtype} \): function from variable to dynamic type: discrete, continuous or algebraic.
- \( V \): locations, \( v_0 \): initial location, \( \text{init} \): initial condition.
- \( \text{flow}, \text{inv}, \text{tcp} \): functions from location to flow condition, invariant, time-can-progress condition.
- \( L \): action labels, \( E \): edges.
Compositional formal semantics of the CIF

The formal semantics are defined in terms of a *Hybrid Transition System*

Two types of transitions:

- discrete behavior by means of *action* transitions, labeled with action label and valuation.
- continuous behavior by means of *time* transitions, labeled with trajectories and duration.
Example of a hybrid transition system

\[
\begin{align*}
\{ & \text{time} \mapsto 4, \quad & \text{time} \mapsto 4 \\
, & n \mapsto 0, \quad & n \mapsto 1 \\
, & V \mapsto 2, \quad & V \mapsto 2 \\
, & \dot{V} \mapsto -2 \} & , & \dot{V} \mapsto 1 \}
\end{align*}
\]
Compositional formal semantics of the CIF

Notion of equivalence proved to be a congruence for all operators of the CIF. For instance, if \( A \) and some transformation \( T(A) \) are bisimilar in the CIF, then \( A \parallel B \) is bisimilar to \( T(A) \parallel B \).

In this way, property preserving translations between parts of specifications within the interchange format itself can be defined. E.g. Language with a triggering guard semantics can be transformed to a verification tool language that uses invariants to force switching to a different location.

In addition, 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. E.g. internal variables of CIF automata in parallel may have the same names.
Differential algebraic equations (DAEs) in the CIF

- Invariant and initial condition are both predicates over variables and differential variables. E.g. steady state initialization requires $\dot{x} = 0$ as initial condition.

- Fully implicit DAEs, including higher index systems, can be specified as invariant. E.g. $f(\dot{x}, x, y, t) = 0$.

Reason for not separating (algebraic) invariant and (differential) flow clause:

- No such separation in mathematics of dynamical systems / control systems theory.

- Many implicit DAE systems such as $f(\dot{x}, x, y, t) = 0$ cannot be rewritten into (semi-)explicit form
  
  $\dot{x} = g(x, y, t), h(x, y, t) = 0$. 
Discrete, continuous and algebraic variables in the CIF

<table>
<thead>
<tr>
<th>Variable Type</th>
<th>Derivative</th>
<th>Trajectory</th>
<th>Part of state</th>
</tr>
</thead>
<tbody>
<tr>
<td>Discrete var</td>
<td>no</td>
<td>constant</td>
<td>yes</td>
</tr>
<tr>
<td>Continuous var</td>
<td>yes</td>
<td>continuous</td>
<td>yes</td>
</tr>
<tr>
<td>Algebraic var</td>
<td>no</td>
<td>discontinuous</td>
<td>no</td>
</tr>
</tbody>
</table>

*Changes in variable values in action transitions*

Discrete and continuous var: no changes, unless explicitly specified (e.g. by means of assignment).

Algebraic var: arbitrary changes, unless explicitly restricted.
Urgency

Urgency allows the passing of time up to a certain point. In principle, two kinds of urgency can be distinguished:

- Urgency defined for an atomic automaton by means of predicates associated to a location or to outgoing edges of the location.

- Urgency defined by means of an operator on (a parallel composition of) automata.

Operator defines a set of actions as urgent for the composition.
Urgency defined for atomic automaton

Abstract format of the CIF:

- The time-can-progress (tcp) condition is a predicate associated to a location.
- Passing of time allowed until the tcp condition becomes false.

For modeling, a triggering guard semantics is often used (Modelica, Simulink, HyVisual, Chi, concrete format of the CIF):

- Time passing in a location allowed until one of the (urgent) guards becomes true.
- Location with $n$ outgoing edges with urgent guards $b_1 \ldots b_n$ (and tcp condition true) is equivalent to location with non-urgent guards $b_1 \ldots b_n$ and tcp condition $\neg b_1 \land \cdots \land \neg b_n$. 
Urgency defined for a composition of automata

Let $\alpha_1$ and $\alpha_2$ denote two CIF automata and let $a$ and $b$ denote two action labels. Then

$$\text{urgent}_{\{a, b\}}(\alpha_1 \parallel \alpha_2)$$

denotes the parallel composition of the automata $\alpha_1$ and $\alpha_2$ such that whenever the parallel composition can execute the action $a$ or $b$, time passing is not allowed.

Defining this kind of urgency by

- referring directly to guards (MoDeST)
- assigning urgency flags to actions of edges (HyTech)

can easily lead to *non-compositionality* (bisimulation no longer being a congruence for parallel composition).
Syntax of the concrete format of the CIF

The concrete format consists of user-friendly syntax for the language elements from the abstract format. Furthermore, it extends the abstract format with:

- *clocks* that are added for compatibility with timed automata,
- *input and output variables* that are added for compatibility with languages such as SIMULINK and PHAVER
- *open and closed scopes* that allow the definition of variables, channels, clocks and actions as being local to facilitate hierarchy and modularity,
- *automaton definition and instantiation* that facilitate re-use of automata.
Example tank controller in concrete CIF
model TankController =
[[
  connect {tank.V, controller.VT},
  {tank.alpha, controller.valve}
]
:: tank:
  [[
    extern var V : cont real = 10,
    alpha: disc nat,
    intern var Qin: alg real
  :: ( mode physics = inv dot V = Qin - 2
      & Qin = alpha * 3
       :: physics
    )
  ]]
|| controller:
  [[
    extern var VT : cont real,
    valve: disc nat = 0
  :: ( mode closed = when VT <= 2 now do valve := 1 goto opened
      , opened = when VT >= 10 now do valve := 0 goto closed
       :: closed
    )
  ]]
]
model TankController

controller
init valve = 0

valve

VT

closed
tcp not VT <= 2

when VT >= 10
do valve:=0

when VT <= 2
do valve:= 1

opened
tcp not VT >= 10

tank
init V = 10

V

alpha

tank
init V = 10

V

alpha

physics

inv Qin = alpha* 3,
dot V = Qin − 2
Simulation CIF model: tank.cif
model TankController =
  extern var V : cont real = 10,
  alpha: disc nat = 0,
  :: tank:
    |( intern var Qin: alg real
    :: |( mode physics = inv dot V = Qin - 2
      & Qin = alpha * 3
      :: physics
    )|
  )|
  || controller:
    || [ [ :: |( mode closed = when V <= 2 now do alpha := 1 goto opened
      , opened = when V >= 10 now do alpha := 0 goto closed
      :: closed
    )|
    ]|
  ]|
model TankController

init alpha = 0
init V = 10

closed

when V >= 10
do alpha := 0

opened

tcp not V <= 2

when V <= 2
do alpha := 1

tcp not V >= 10

inv Qin = alpha * 3,
\[ \dot{V} = Qin - 2 \]

closed
tcp not V <= 2
opened
tcp not V >= 10

model TankController
tank

init V = 10

physics

inv Qin = alpha * 3,
\[ \dot{V} = Qin - 2 \]
Semantics of the concrete format of the CIF

The semantics of the concrete format is defined by means of a mapping from the concrete format to the abstract format.

See paper *Concrete syntax and semantics of the compositional interchange format for hybrid systems*, IFAC world conference, Seoul, Korea 2008
Current status of the CIF

• Language
  – Syntax and formal semantics of the abstract format
  – Syntax and formal semantics of the concrete format

• Tools
  – Translator from the concrete format to the abstract format
  – Simulator
  – Real-Time simulator in order to control hardware
  – Translator of Wonham automata (used for supervisor synthesis) to the CIF vice versa.
  – Visualisor of the abstract format

• Applications
  – Small examples
  – Industrial case study
Future of the CIF

Development of translations to and from the CIF, including the tools are embedded in the *MULTIFORM* project. 
*MULTIFORM* is a new FP7 European project on networked embedded and control systems - control of large-scale complex distributed systems. 
Start: 1 September 2008, duration 3.5 years. 
Participating partners:

<table>
<thead>
<tr>
<th>Name</th>
<th>Acronym</th>
<th>Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technische Universität Dortmund (coordinator)</td>
<td>TUDO</td>
<td>Germany</td>
</tr>
<tr>
<td>Technische Universiteit Eindhoven</td>
<td>TUE</td>
<td>The Netherlands</td>
</tr>
<tr>
<td>Université Joseph Fourier Grenoble</td>
<td>UJF</td>
<td>France</td>
</tr>
<tr>
<td>RWTH Aachen</td>
<td>RWTH</td>
<td>Germany</td>
</tr>
<tr>
<td>Aalborg Universitet</td>
<td>AAU</td>
<td>Denmark</td>
</tr>
<tr>
<td>Embedded Systems Institute</td>
<td>ESI</td>
<td>The Netherlands</td>
</tr>
<tr>
<td>VEMAC GmbH &amp; Co. KG</td>
<td>VEMAC</td>
<td>Germany</td>
</tr>
<tr>
<td>KVCA A/S</td>
<td>KVCA</td>
<td>Denmark</td>
</tr>
</tbody>
</table>
Multiform: CIF related activities

- Extension of the CIF (TUE)
- Translation of $\chi$ to and from the CIF (TUE)
- Translation of UPPAAL and PHAVER to and from the CIF (UJF, AAU, TUE)
- Translation of MODELICA and GPROMS to and from the CIF (TUDO)
- Translation of MATLAB/SIMULINK to and from the CIF (TUDO)
Future of CIF (cnt.)

AND perhaps YOU have developed a formalism:
Translation of your formalism to (and from) the CIF to use existing tools.

OR perhaps YOU have developed a tool:
The CIF might be useful as the input language of your tool, or a translation from CIF to the input language might enable application of your tool to models originally specified in some other formalism.
Questions