The Compositional Interchange Format: concepts, formal basis, and applications

Bert van Beek

5 November 2009
Purpose of the Compositional Interchange Format

Background

- For the development of high tech industrial systems many different formalisms and tools are required
- To ensure consistency of results across different models and tools, tool-integration and model reuse is essential

The main purpose of CIF is to

- Establish inter-operability of a wide range of tools by means of model transformations to and from CIF

In addition, tools that operate directly on the CIF are available for

- simulation of untimed, timed and hybrid CIF models
- supervisory control synthesis of untimed and timed models
CIF transformation examples

- **UPPAAL**
  - Timed automata verification

- **PHAVer**
  - Linear hybrid automata verification

- **Modelica, gPROMS**
  - CT plant design

- **Chi**
  - DE plant design

- **CIF**
  - ASCII plant specification
  - Graphical plant specification

- **CIF XML/Ecore controller**
  - Timed/hybrid/realtime simulation

- **CIF XML/Ecore controlled plant**

- **CIF XML/Ecore plant**

- **Rose RT Statecharts**
  - Real-time DE control

- **Matlab/Simulink**
  - CT controller design

- **Sequential Function Charts**
  - Graphical specification

- **Supervisory control tools**
  - DE controller synthesis

- **Modelica**
  - High speed hybrid simulation

---

CT = Continuous-time
DE = Discrete-event

Bert van Beek, FMES in FMWeek 2009
Transformations without CIF

Language $A_0$ → Language $B_0$

Language $A_1$ → Language $B_1$

Language $A_2$ → Language $B_2$

Language $A_3$ → Language $B_3$
Transformations with CIF

Language $A_0$ -> CIF -> Language $B_0$
Language $A_1$ -> CIF -> Language $B_1$
Language $A_2$ -> CIF -> Language $B_2$
Language $A_3$ -> CIF -> Language $B_3$
Properties of CIF

- Based on automata
- *Formal and compositional semantics* allowing *property preserving model transformations*
- *Differential algebraic equations* (possibly discontinuous)
- *Hierarchy and modularity* to deal with *large scale systems*: model re-use, parallel models, and nested models

**Process interaction:**
- Communication via *shared channels*
- Synchronization by means of *shared actions*
- *Shared variables*

**Support for *urgency***:
- urgent actions and channels
- urgent locations
Development of CIF

Modelica, gPROMS
- DAE system simulation
- Optimization

Chi
- CT/DE simulation
- Process algebraic SCS

UPPAAL
- Timed automata verification
- Timed games SCS

PHAVer
- Linear hybrid automata verification

1. EU NoE HYCON1,2
2. EU FP7 Multiform
3. EU FP7 C4C
4. EU ITEA2 Twins
5. NL Darwin

CIF:
- Syntax and formal semantics
- Simulation
- Co-simulation
- Compositional verification: Ariadne
- Abstraction, refinement: E.g. and/or superstates
- CIF to CIF transforms
- Eclipse graphical IDE
- Real-time control via EtherCat

Switched linear systems interchange format

Discrete-time PWA
- Toolboxes: Multi-Param, Hybrid, Identification

Matlab/Simulink
- Block oriented control

Sequential Function Charts control

Wonham event-based
- Modular SCS

Ma/Wonham state-based
- STS SCS

Real-time control
- Rose RT Statecharts: Error handling printer paper path
- Platform specific: MRI scanner patient support control

Distributed control
- Printing process and job control

SCS = Supervisory Control Synthesis
Industrial applications of CIF
Real-time error handling of paper path control using supervisory control synthesis
Industrial applications of CIF

Distributed control of the printing process using supervisory control synthesis

Océ specializes in durable, high-end equipment, suitable for corporate publishing/reproduction centers, as well as commercial printing and copying operations. Most equipment produced by Océ is high-speed (50 pages per minute and over) and has very high duty cycles (half a million pages a month and higher).

The process part, where the toner images are being formed and transferred to the paper, is a yet mostly unexplored area for control generation. Here we need to deal with heaters, temperature sensors, set-up and cleaning mechanisms, transport belts, etc.

Unlike the paper path, which is a coherent constellation of motors and pinches, this is a more diverse grouping of rather loosely coupled components.

The embedded control software for the process faces the problem that it has to control a very large amount of possibly conflicting machine states (like temperature too high while cleaning drum while solving error).

As illustration of undesired emergent behavior, consider the following example: It may happen that the imaging drum becomes too hot, in that case it should be cooled (by a fan), which can only happen in standby. However, a transition from running to
Industrial applications of CIF

Model based engineering of a document inserter
Industrial applications of CIF

Real-time control of a patient support system

Supervisory control synthesis using:
- modular supervisory control
- state-based supervisory control

Uncontrolled system: 6.3 billion states

PICU:

Tabletop sensor (on/off)
Position encoder (on/off)
Horizontal brake (on/off/stopped)
Clutch (on/off)
Max out sensor (on/off)
TTR button (on/off)
Vertical motor (up/down/stepped)
Vertical brake (on/off)
Max up sensor (on/off)
Max down sensor (on/off)

Light-visor
Bore
Patient support table
Tabletop
Why use CIF for model transformations?

CIF is more general than other languages. E.g:

- UPPAAL has point to point synchronization by means of channels; PHAVer has multi-process synchronization by means of actions; CIF has both
- Block diagram languages (e.g. Matlab/Simulink) and hybrid IO automata deal with input-output systems; Behavioral languages (e.g. Modelica) deal with acausal systems; CIF deals with both
- Simulation languages deal with urgent (triggering guard semantics) and deterministic systems; Verification formalisms deal with non-urgent (using invariants to force actions) and non-deterministic systems; CIF deals with both
Why use CIF for model transformations?

CIF has a formal compositional semantics:

- The meaning of any CIF component is defined independently of its environment (bisimulation proven to be a congruence for all operators)
- E.g. if a hybrid CIF component $\alpha_{\text{hybrid}}$ with local variables can be simplified as an equivalent timed component $\alpha_{\text{timed}}$, then

$$\alpha_{\text{hybrid}} \parallel C$$

is equivalent to

$$\alpha_{\text{timed}} \parallel C$$

for all CIF components $C$
Model transformations

- In many cases uni-directional transformations are sufficient (e.g. real-time code generation, or transformation for verification)
- Transformation may require model rewrite (e.g. elimination of parallel composition for non-compositional languages)
- Transformations may be defined on language subsets
Bottle filling line example

\[ Q_{u}, c_{u} \quad Q_{a}, c_{a} \]

\[ V_{T}, n, c, pH \]

\[ Q_{F1} \quad Q_{Fr} \]
Bottle filling line: CIF graphical model

tank

var alpha, beta: disc nat = 0, 0;
  n: cont real;
  pH: alg real = 7;
  c, Qa, Qu: alg real

physics
  inv
  n = c*V,
  dot V = Qu + Qa - QFl - QFr,
  dot n = cu*Qu + ca*Qa - c*QFl
     - c*QFr - Kloss*V,
  pH = - log c/1000,
  Qa = alpha*Qseta,
  Qu = beta*Qsetu

when pH >= 7.1 do alpha:= 1
when pH <= 7 do alpha:= 0
when V >= 10 do beta:= 0
when V <= 2 do beta:= 1

when V >= 10 do beta:= 0
when V <= 2 do beta:= 1

when pH >= 7.1 do alpha:= 1
when pH <= 7 do alpha:= 0
when V >= 10 do beta:= 0
when V <= 2 do beta:= 1

var QFl: alg real
var QFr: alg real
var V: cont real

Bottle_Filling_System

Bottle_Filling_Line

bottles?

Bottle_Filling_Line

bottles?

Bottle_Supply

bottles!
Bottle filling line: CIF textual model

model Bottle_Filling_System =
  ![ default urgent ]
  ![ connect {tank.V, left.VT, right.VT} ]
  ![ {tank.QFl, left.QF} ]
  ![ {tank.QFr, right.QF} ]
  ![ { bs.bottles
    ! left.bottles
    ! right.bottles
  ]

:: tank:
  ![ output var V: cont real = 10 ]
  ![ extern var QFl, QFr: alg real ]
  ![ intern var alpha: disc nat = 0 ]
  ![ ; beta: disc nat = 0 ]
  ![ ; n: cont real ]
  ![ ; pH: alg real ]
  ![ ; c, Qa, Qu: alg real ]
  init pH = 7
:: ![ ( mode physics = eqn ]
  ![ dot V = Qu + Qa - QFl - QFr ]
  ![ , dot n = cu*Qu + ca*Qa - c*QFl ]
  ![ - c*QFr - Kloss*V ]
  ![ , n = c*V ]
  ![ , pH = - log(c/1000) ]
  ![ , Qa = alpha*Qseta ]
  ![ , Qu = beta*Qsetu ]
:: physics
)
|| |( mode closed =
  ![ when pH >= 7.1 ]
  ![ do alpha := 1 goto opened ]
  ![ mode opened =
    ![ when pH <= 7 ]
    ![ do alpha := 0 goto closed ]
:: closed
  )|
|| |( mode closed =
  ![ when V <= 2 ]
  ![ do beta := 1 goto opened ]
  ![ mode opened =
    ![ when V >= 10 ]
    ![ do beta := 0 goto closed ]
:: closed
  )|
|| left : Bottle_Filling_Line
|| right : Bottle_Filling_Line
|| bs : Bottle_Supply
)
Different representations of CIF

Concrete CIF: User friendly syntax for modeling directly in CIF. Available in textual or graphical form

Abstract CIF: XML or ecore (for Eclipse platform) representation of CIF for model transformations

Core CIF: Smallest set of CIF primitives for mathematical definition of the formal semantics
CIF redesign in Multiform

- Complete redesign of the core language and formal semantics in cooperation with C4C
- Improvements of concrete CIF, and addition of data types
- Operators of CIF are now restricting behavior, which facilitates compositional verification
- Concrete CIF and Core CIF now based on similar concepts
- Considerable simplification of the formal semantics (SOS rules)
Mathematical syntax of atomic interchange automaton

An atomic interchange automaton is a tuple

\[(V, v_0, \text{flow}, \text{inv}, \text{tcp}, E)\]

where

- \(V\): set of locations (vertices)
- \(v_0\): initial location
- flow, inv, tcp: functions from location to flow predicate, invariant, time-can-progress predicate, respectively
- \(E\): set of edges
- An edge can have an basic action label \(\ell\), a send action \(h!e\), or a receive action \(h?x\):
  - \((v, g, \ell, (W, r), v')\)
  - \((v, g, h!e, (W, r), v')\)
  - \((v, g, h?x, (W, r), v')\)
Mathematical syntax of core CIF

\[ \alpha ::= \alpha_{atom} \]
\[ \begin{array}{l}
\alpha \mid \alpha \parallel \alpha \\
\gamma_{a}(\alpha) \quad \text{synchronising action operator} \quad a \text{ action label} \\
u \gg \alpha \quad \text{initialisation operator} \quad u \text{ predicate} \\
\begin{bmatrix} V \ x = e :: \alpha \end{bmatrix} \quad \text{variable scope operator} \quad x \text{ variable}, e \text{ expression} \\
\begin{bmatrix} A \ a :: \alpha \end{bmatrix} \quad \text{action scope operator} \quad a \text{ action label} \\
\begin{bmatrix} H \ h :: \alpha \end{bmatrix} \quad \text{channel scope operator} \quad h \text{ channel} \\
\partial_{h}(\alpha) \quad \text{channel encapsulation operator} \quad h \text{ channel} \\
U_{z}(\alpha) \quad \text{urgency operator} \quad z \text{ action or channel} \\
D_{x:G}(\alpha) \quad \text{dynamic type operator} \quad G \text{ set of pairs of trajectories} \\
st_{x}(\alpha) \quad \text{state variable operator} \quad x \text{ variable} \\
\text{own}_{x}(\alpha) \quad \text{ownership operator} \quad x \text{ variable}
\end{array} \]

Parallel composition now restrictive for synchronizing behavior. Other operators restrictive for all behavior.
Formal semantics

States

\[ \mathcal{A} \times \Sigma \times V \]

Action transitions

\[ (\alpha, \sigma, X) \xrightarrow{a,A} (\alpha', \sigma', X') \subseteq \text{States} \times (\mathcal{L}_\tau \times 2^\mathcal{L}) \times \text{States} \]

Time transitions

\[ (\alpha, \sigma, X) \xrightarrow{t,\rho,\theta} (\alpha, \sigma, X') \subseteq \text{States} \times (T_{\geq 0} \times (T \rightarrow \Sigma) \times (T \rightarrow \text{val}_{\text{ah}}^4)) \times \text{States} \]

where \( \text{val}_{\text{ah}} = (\mathcal{L}_{\text{basic}} \cup \{ \tau \} \cup \mathcal{H}) \rightarrow \mathbb{B} \)

Environment transitions

\[ (\alpha, \sigma, X) \xrightarrow{A} (\alpha', \sigma', X') \subseteq \text{States} \times 2^{\mathcal{L}_{\text{basic}}} \times \text{States} \]
Synchronization

All actions are non-synchronizing by default. Synchronization can be achieved by means of the synchronization operator $\gamma_l(\alpha)$. The behavior of the operator is formalized by manipulating the set of synchronizing variables in the labels of the transitions.

$$
(\alpha, \sigma, X) \xrightarrow{a, A} (\alpha', \sigma', X')
$$

$$
(\gamma_l(\alpha), \sigma, X) \xrightarrow{a, A \cup \{l\}} (\gamma_l(\alpha'), \sigma', X)
$$

$$
(\alpha, \sigma, X) \xrightarrow{A} (\alpha', \sigma', X')
$$

$$
(\gamma_l(\alpha), \sigma, X) \xrightarrow{A \cup \{l\}} (\gamma_l(\alpha'), \sigma', X)
$$

The sets of synchronizing actions are considered in the rules for parallel composition.
Parallel composition

Parallel composition is strictly a restriction for synchronizing behavior. Thus the parallel composition

\[ \gamma_\ell(\ell : \{x\} : x^+ = 1) \parallel \gamma_\ell(\ell : \{y\} : y^+ = 1) \]

cannot do an action transition unless \( x \) and \( y \) are already 1. More generally

\[ \gamma_\ell(\ell : W : r) \parallel \gamma_\ell(\ell : W' : r') \leftrightarrow \gamma_\ell(\ell : W \cap W' : r \land r') \]
SOS for synchronizing parallel composition

The synchronizing parallel composition is the intersection of the behaviors of the components.

\[ (\alpha_0, \sigma, X) \xrightarrow{a, A_0} (\alpha'_0, \sigma', X'), (\alpha_1, \sigma, X) \xrightarrow{a, A_1} (\alpha'_1, \sigma', X'), \ a \in A_0 \cap A_1 \]

Changes to variables are possible only if this is allowed by both components. For example:

\[ \gamma_\ell(\ell : \{x, y\} : x^+ = 1) \parallel \gamma_\ell(\ell : \{x, y\} : y^+ = 1) \]

is bisimilar to:

\[ \gamma_\ell(\ell : \{x, y\} : x^+ = 1 \land y^+ = 1) \]
SOS for interleaving parallel composition

The non-synchronizing partners do a consistency transition.

- Changes to variables owned by a non-synchronizing automaton are forbidden.
- All other variables can change arbitrarily as long as they are consistent with active invariants, flow conditions, and initialization predicates.

\[
(\alpha_0, \sigma, X) \xrightarrow{a, A_0} (\alpha'_0, \sigma', X'), (\alpha_1, \sigma, X) \xrightarrow{A_1} (\alpha'_1, \sigma', X'), a \notin A_0 \cap A_1
\]

\[
(\alpha_0 \parallel \alpha_1, \sigma, X) \xrightarrow{a, A_0} (\alpha'_0 \parallel \alpha'_1, \sigma', X)
\]
Concrete CIF provides user-friendly syntax for the language elements from core CIF.

Furthermore, it extends core CIF 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.
Concluding remarks CIF redesign

• Considerable simplification of the formal semantics (SOS rules)
• Each concept in the language modeled by means of an operator
• Concrete and core CIF now based on similar concepts
• Operators of CIF are now restricting behavior
• Formal semantics still compositional (bisimulation is a congruence for all operators)
Concluding remarks CIF tooling

- Translators available or under development for connecting the CIF to tools for:
  - Large scale DAE based hybrid system simulation
  - Hybrid system optimization
  - Verification of timed and hybrid systems
  - Supervisory control synthesis
  - Analysis and control of Discrete-time Piecewise Linear Affine systems
  - Block oriented analysis and (real-time) control system design

- Several tools available or under development for direct use of the CIF:
  - Graphical editor
  - Modeling and simulation
  - Hybrid system verification
  - Supervisory control synthesis and real-time control
Acknowledgements

- Dennis Hendriks, programmer SE
- Damian Nadalus, PhD student SE
- Jacobus (Koos) E. Rooda, group leader SE
- Ramon R.H. Schiffelers, postdoc SE