On the Connection of Equation- and Automata-based Languages: Transforming the Compositional Interchange Format to Modelica

Christian Sonntag, Martin Hühner

Process Dynamics and Operations Group
Dept. of Biochemical and Chemical Engineering
Technische Universität Dortmund, Germany
{christian.sonntag,martin.huefner}@bci.tu-dortmund.de.

Abstract: In recent years, the object-oriented MODELICA formalism for dynamic hybrid models has become a de-facto standard for the equation-based modeling, design, and analysis of complex, heterogeneous systems. It provides powerful mechanisms for model structuring and consistent model re-use, and a variety of tools are available that are based on the MODELICA formalism. In this paper, an algorithmic transformation from the automaton-based Compositional Interchange Format (CIF) for hybrid systems to the equation-based MODELICA language is defined. The transformation algorithm transforms the discrete automaton structure of the CIF into a set of Boolean equations in MODELICA, while the continuous parts are mapped into suitable MODELICA equation constructs. Using this algorithm, a very large subset of the CIF can be mapped to MODELICA, thereby significantly extending the capabilities of both. Through this transformation, the CIF is equipped with support for comfortable modeling and simulation environments and with a comprehensive model library, while MODELICA gains access to numerous model-based tools that are connected to the CIF. The transformation is illustrated on a realistic hybrid example that offers significant continuous dynamics as well as highly complex discrete dynamics.

Keywords: Hybrid systems, Model exchange, Equation-based languages, Automata-based languages, Modelica, Compositional Interchange Format

1. INTRODUCTION

Model-based methods for the design and analysis of technical systems are becoming increasingly important in industry since they offer many advantages over traditional design methodologies, such as increased safety, a more economical operation, and a significant reduction of the design duration which translates to significant financial savings. In recent years, sophisticated methods and tools have been developed for all stages of systems design, ranging from requirements analysis and simulation to automatic synthesis of optimal and robust controllers and implementation verification. A major obstacle in the application of model-based techniques is the incompatibility of the model formalisms that are employed by different tools and techniques, see e.g. [17]. Today, the transformation of models must usually be carried out manually, which is time-consuming, error-prone, and expensive. To overcome this obstacle, a framework for the integration of model-based design and analysis tools is currently developed in the scope of the European research project MULTIFORM [MULTIFORM]. In this framework, the interconnection of design and analysis tools via the exchange of models between a large number of formalisms, such as MATLAB/SIMULINK, MODELICA [2, 6], gPROMS [9], UP-PAAL [8], PHAVER [5], and others, assumes an important role. The model exchange is realized by defining bi-directional transformations between proprietary modeling formalisms and a general interchange format, the Compositional Interchange Format (CIF) for hybrid systems [21, 22, 23, 19, 3].

This paper presents an algorithmic transformation from the CIF to the object-oriented, equation-based MODELICA language that has become a de-facto standard for the equation-based modeling, design, and analysis of complex, heterogeneous systems. It provides powerful mechanisms for model structuring and consistent model re-use, and a variety of tools and libraries are available that are based on the MODELICA formalism. The integration of finite-state or mixed discrete-continuous automaton-based formalisms into MODELICA has been in the focus of research efforts for several years (see e.g. [15, 12, 14, 16]). The transformation from the CIF to MODELICA is inspired by the principles employed to implement Petri Nets [10] and StateGraphs [13] in MODELICA. In these approaches, the discrete structures of the source formalisms are modeled using discrete-time Boolean equations without the need for algorithmic constructs, which is very advantageous in
an equation-based modeling formalism, as discussed in e.g. [10]. The transformation developed in this work provides a general framework for hybrid automaton-based formalisms within Modelica in which the complex, discrete CIF structure is represented by a set of Boolean equations while the continuous parts are mapped into suitable Modelica equation constructs. Using this algorithm, a very large subset of the CIF can be mapped to Modelica, thereby significantly extending the capabilities both. Through this transformation (and a transformation from Modelica to the CIF that was developed in previous work), the CIF is equipped with support for comfortable modeling and simulation environments and with a comprehensive model library, while Modelica gains access to numerous model-based tools that are connected to the CIF. The transformation is illustrated on a realistic hybrid example that offers significant continuous dynamics as well as highly complex discrete dynamics.

2. THE COMPOSITIONAL INTERCHANGE FORMAT FOR HYBRID SYSTEMS

The Compositional Interchange Format (CIF) for hybrid systems is a modeling and model exchange formalism for general hybrid systems that encompasses many of the language concepts that are present in modern modeling formalisms. The CIF was initially developed within the European Network of Excellence HYCON [1] and has been improved and extended ever since, see e.g. [21, 22, 23, 19]. Recently, it has been redesigned to reduce its syntactic and semantical complexity and to simplify the definition of model transformations [3].

Fig. 1 gives a simplified overview of the CIF modeling framework. To provide a syntactically rich modeling format as well as a precise mathematical definition of the model semantics, three different CIF model representations have been developed. The CIF formalism is the syntactically most expressive of the three languages. Its (syntactic and semantic) expressivity is similar to that of modern general modeling languages such as Modelica, gPROMS, and ECOSIMPro. CIF models are based on communicating hybrid transition systems (atomic automata). It supports (deterministic or stochastic) variables, complex data types (including lists, arrays, matrices, dictionaries, enumerations, and tuples), internal and external function invocations, and communication by synchronizing actions or shared variables. Continuous dynamics can be modeled using fully implicit discontinuous DAE systems, and the CIF offers different concepts to specify properties that often arise in hybrid languages, such as transition urgency, invariants, and time-can-pass predicates. Since many modern modeling languages provide syntactical elements to facilitate and structure the modeling process, the CIF supports in addition hierarchy, modularization, and automaton instantiation. core CIF contains only a minimal set of syntactical concepts, but possesses the same behavioral expressivity as CIF. Thus, models can be transformed between the two representations without any change in model semantics. µCIF, on the other hand, is a mathematical formalism that is equivalent to core CIF, but that is defined in a mathematical way. Formal, compositional semantics have been defined for the µCIF language which enables correctness proofs if the transformed languages also possess formal semantics. Being a model exchange format, the CIF framework has been designed to make the definition of model transformations as simple as possible. For both, CIF and core CIF, abstract grammars and Ecore class diagram definitions [20] are available. The former allow to define concrete grammatical CIF representations (e.g. textual and graphical), while the latter enables a straightforward definition of languages and transformations (using dedicated transformation languages) within the Eclipse modeling framework, see e.g. [3, 7]. Besides the connection of languages to the CIF, such transformations are also used for CIF-to-CIF transformation (e.g. to transform between the three CIF formalisms, to replace modeling constructs that are not supported by a target language with equivalent, supported constructs, or to transform concrete-grammar models into class diagram form). The transformation algorithm presented in this paper is based on the µCIF formalism that is defined as follows 1:

**Def. 1 (Atomic µCIF Automaton 2):** An atomic CIF automaton is a tuple \( \alpha_{\text{atom}} = (V, \text{init}, \text{inv}, \text{tcp}, E, \text{varC}, \text{actS}, \text{dtype}) \) where:

- \( V \) is the set of discrete modes or locations. The active mode is the mode for which init evaluates to true.
- \( \text{init} \) (the initial conditions), \( \text{inv} \) (the invariants), and \( \text{tcp} \) (the time-can-pass predicates) are mappings of the type \( V \rightarrow \text{pred}(X \cup X)^3 \), where \( X \) is the set of all variables, and \( \hat{X} \) denotes the set of the time derivatives of all variables in \( X \), i.e. \( \hat{X} = \{ \dot{x} | x \in X \} \).
- \( E \) is the set of edges or discrete transitions. Each edge \( e \in E \) is of the form \( e = (v, g, a, (W, r), v') \), where
  - \( v, v' \in V \) are the source and target modes of \( e \),
  - \( g \in \text{pred}(X | \dot{X}) \) is the guard condition (a Boolean predicate that indicates if the edge is enabled),
  - \( a \in L_r \) is an action label, where the set \( L_r = L \cup \tau \) comprises all action labels and the non-synchronizing default label \( \tau \),
  - \( W \in 2^X \cup \dot{X} \) is the set of jumping variables (that can discontinuously change during a discrete transition), and
  - \( r \in \text{pred}(X | \dot{X} \cup X | \dot{X})^5 \) is the reset mapping, i.e. a predicate that indicates the possible changes of variables in \( W \).

1 The presentation of the transformation for hierarchical CIF models is not feasible due to space limitations. However, the extension to hierarchical models is straightforward, as outlined in Sec. 4.3.

2 Abbreviated and simplified version of Def. 1 in [3].

3 For a set of variables \( Y \), \( \text{pred}(Y) \) denotes the set of all predicates, and \( \text{expr}(Y) \) is the set of all expressions over the variables in \( Y \).

4 For a set of variables \( Y \), \( Y^+ = \{ y^+ | y \in Y \} \) are the evaluations of all \( y \in Y \) after the execution of a discrete transition.
\* varC ∈ 2\(X\) are the control variables of the automaton, i.e. those variables that can only be changed by the automaton that defines them. 
\* act ∈ 2\(\mathbb{S}\) is the set of synchronizing actions of the automaton. It contains those actions that synchronize actions with the same name in parallel automata. 
\* dtyp ∈ \(X \rightarrow \{\text{disc, cont}\}\) defines the dynamic type (discrete or continuous) of the variables in \(X\) (and their derivatives in \(X\))^5.

**Def. 2 (\(\mu\)CIF Interchange Automaton⁶):** The set of interchange automata \(A\) is recursively defined by the following grammar:
\[
\alpha ::= \text{atom} \mid [\alpha] \mid \gamma_\alpha(\alpha) \mid u \gg \alpha \mid [A \downarrow : \alpha] \mid U_\alpha(\alpha) \mid [v \leftarrow \text{expr}, \dot{x} \leftarrow \text{expr} : \alpha] \mid D_{\alpha,G}(\alpha) \mid c_\alpha(\alpha),
\]
where \([\cdots]\) denotes parallel composition, the operator \(\gamma_\alpha(\alpha)\) makes action \(a\) synchronizing within \(\alpha\), \(u \gg \alpha\) is the initialization operator, \([A \downarrow \cdots]\) is the action scope operator (used to define local actions), the operator \(U_\alpha(\alpha)\) states that action \(a\) is urgent within \(\alpha\) (i.e. must be fired as soon as it is enabled), \([v \leftarrow \text{expr}, \dot{x} \leftarrow \text{expr} : \alpha]\) is the variable scope operator (used to define local variables), \(D_{\alpha,G}(\alpha)\) is the dynamic type operator (used to equip variables with dynamic types), and \(c_\alpha(\alpha)\) is the control variable operator (states that the variables can only be modified by \(\alpha\)). □

The formal and the informal semantics of \(\mu\)CIF are described in detail in [3]⁷.

### 3. THE MODELICA LANGUAGE

The Modelica language [2] is receiving wide-spread attention as a de-facto language for the modeling and design of complex, heterogeneous hybrid systems in both, industry and academia. In contrast to the CIF, which is based on a hybrid automaton formulation, Modelica is inherently equation-based. This means that before execution, Modelica models are transformed into a single hybrid DAE system that comprises Boolean equations as well as implicit differential-algebraic continuous equations. The major advantage of this approach is that the hybrid DAE system can be transformed into a block-triangular form that allows for very efficient execution. Modelica is probably the modeling language that follows the object-oriented paradigm most closely, and it offers many of the concepts that also distinguish modern object-oriented programming languages such as C# or C++, including inheritance or polymorphism. Consequently, every element of Modelica is considered to be a class (or a class instance), down to the basic data types.

In this work, a subset of the Modelica language, termed Modelica\(_{\mu\text{CIF}}\), is defined that only contains those elements needed to represent CIF models:

**Def. 3 (Modelica\(_{\mu\text{CIF}}\) Model):** A Modelica\(_{\mu\text{CIF}}\) model \(m_{\mu\text{CIF}}\) is a set of classes \(C\), with subsets \(C_{\text{model}} \subseteq C\) (representing Modelica model entities), \(C_{\text{conn}} \subseteq C\) (representing Modelica connector entities), \(C_{\text{fn}} \subseteq C\) (representing Modelica functions), \(C_{\text{model}} \bigcup C_{\text{conn}} \bigcup C_{\text{fn}} = C\), and external or built-in classes \(C_{\text{ext}}\) (representing basic data types and classes in Modelica libraries that are imported):

- Each model entity \(c_{\text{model}} \in C_{\text{model}}\) is a tuple \(c_{\text{model}} = \{X_m, \text{init}_m, eq, \text{alg}, \text{dtyp}_m, \text{dir}\}\), where \(X_m\) are the components (i.e. variables, parameters, constants, or more complex class instances) within the entity. \(\text{init}_m\) are initialization predicates, \(eq\) is an equation section that contains a set of continuous, discrete, or instantaneous (when) equations, \(\text{alg}\) is an algorithm section that contains an ordered list of algorithmic statements, \(\text{dtyp}_m \in X_m \rightarrow \{\text{constant, parameter, discrete, continuous}\}\) defines the dynamic type of the elements of \(X_m\) (and their derivatives), and \(\text{dir} \in X_m \rightarrow \{\text{input, output, inout}\}\)⁸ represents the directionality of the elements of \(X_m\). Model entities are used to represent the dynamic behavior of a Modelica model.
- Each connector entity \(c_{\text{conn}} \in C_{\text{conn}}\) is a tuple \(c_{\text{conn}} = \{X_m, \text{dtyp}_m, \text{dir}\}\) whose elements are defined as above. Connector entities are used as interfaces between model entities.
- Each function \(c_{\text{fn}} \in C_{\text{fn}}\) is a tuple \(c_{\text{fn}} = \{X_m, \text{alg}, \text{dtyp}_m, \text{dir}\}\) whose elements \(X_m, \text{alg}, \text{dtyp}_m\) are defined as above, and \(\text{dir} \in \{\text{input, output}\}\). Functions are mathematical input/output mappings, similar to those in modern programming languages. □

The CIF has been designed to encompass most of the language constraints that are common in modeling formalisms, and its syntax is thus not strongly restricted. In contrast, Modelica was designed to enable the detection of many model errors already during the compilation phase. Consequently, its syntax is subject to numerous restrictions. In the following, some language concepts are presented that are important for the transformation algorithm described below, and that lead to restrictions throughout the language (some of these restrictions will be discussed in the next section).

Modelica is a synchronous language [2] which means that (1) all variables keep their values until these values are explicitly changed (i.e. by equations or algorithmic statements), (2) all active model equations at any point in time must be fulfilled concurrently, and (3) any computation at discrete events does not consume any (simulation) time. In addition, Modelica is restricted by two important rules:

- **Single-assignment rule:** The total number of equations must be identical to the total number of unknown variables at all times.
- **Balanced models [11]:** All model entities of a Modelica\(_{\mu\text{CIF}}\) model must be locally balanced, which means that the number of unknowns must at all times be equal to the number of active equations. A model is globally balanced if the number of unknowns of the complete model is at all times equal to the number of active equations.

---

5. The definition of \(\text{dtyp}\) has been simplified to only include the relevant dynamic types, compare to [3].

6. Abbreviated and simplified version of Def. 2 in [3].

7. Note that static types of variables are omitted throughout the paper for clarity, as these can be directly mapped from the CIF to Modelica.

8. The default value of \(\text{dir}\) is \(\text{inout}\).
4. THE TRANSFORMATION ALGORITHM

4.1 The Transformable \( \mu \text{CIF} \) Subset

The transformation algorithm presented here supports a large subset of \( \mu \text{CIF} \) (and also CIF) models, thus only few restrictions are needed to ensure that a model is transformable:

**Def. 4 (Transformable \( \mu \text{CIF} \) Model):** A \( \mu \text{CIF} \) model is transformable to Modelica\( \mu \text{CIF} \) if and only if:

- It is *simulatable*, which (among others) includes the requirements that it is *globally balanced* as defined above, that all variables are equipped with the correct dynamic and static types, that it is *deadlock-free* (unless a deadlock is introduced deliberately, e.g. to terminate the simulation in a certain error mode), and that its active equations possess a unique solution within the complete state space.
- It does not contain any stochastic constructs.
- It does not contain any *actions*, and all discrete transitions are *urgent*.
- Any *local variables* (i.e. variables defined in a variable scope operator) must not have the same name as any variable in the enclosing scope.
- At the first initialization, the *init* predicate must evaluate to *true* for a single discrete mode (this is necessary since Modelica is deterministic while \( \mu \text{CIF} \) supports non-deterministic initialization).
- \( \forall \alpha \in \mathcal{A}, \forall e \in E(\alpha) \): In any variable update \( (W, r) \) of \( e \) that is not an assignment, the predicates \( r \) must be solvable for the variables in \( W \).

Note that if a \( \mu \text{CIF} \) interchange automaton is transformable, it does not contain any operators of the types \( \gamma_n(\alpha) \) or \( |\mathcal{A}| \), and that all \( U_{\alpha} \) can be ignored since it is assumed that all transitions are urgent (see Def. 2).

4.2 Transforming \( \mu \text{CIF} \) to Modelica\( \mu \text{CIF} \)

The transformation of atomic \( \mu \text{CIF} \) automata to Modelica\( \mu \text{CIF} \) is based on six Modelica components that model the *discrete modes* \( V \), the *discrete transitions* \( E \), and *interfaces* between these. The following connector and model entities are added to the initially empty model \( m_{\mu \text{CIF}} \):

```
01 model CIFMode
02 parameter Integer n=0, m=0;
03 CIFStepInPort inPorts[n];
04 CIFStepOutPort outPorts[m];
05 parameter Boolean initStep = false;
06 Boolean invariant = true;
07 Boolean active;
08 Boolean newActive = (start=initStep, fixed=true);
09 equation
10   for i in 1:m loop
11      outPorts[i].fire = active and (if i=1 then
12         outPorts[i].enabled else (outPorts[i].enabled
13            and not outPorts[i-1].enabled));
14   end for;
```

It is not required that atomic automata are locally balanced.

**Action-based synchronization can be reformulated to synchronization by shared variables using a CIF-to-CIF transformation.**
the transition with the smallest index is chosen. The equations on lines 13-15 update the activity variable of the mode. On lines 16-18, the invariant is validated. If it evaluates to false while the mode is active, an error is triggered since this constitutes a deadlock.

A µCIF interchange automaton is mapped into a single Modelica model entity $e_{\mu\text{CIF}}$ that is added to the set $\mathcal{C}_\text{model}$. Starting from the innermost element of an interchange automaton $\mathcal{A}$, the transformation proceeds as follows:

1. Choose an element $\alpha_i \in \mathcal{A}$ that has not yet been transformed.
2. If $\alpha_i$ is of type $\alpha_{\text{atom}}$:
   a. Reproduce the discrete structure consisting of all $v \in V(\alpha_i)$ and $e \in E(\alpha_i)$ using the model entities described above. This step comprises an instantiation of $\text{CIFMode}$ in $e_{\mu\text{CIF}}.X_m$ for each $v \in V$ (setting the variables as invariant = $\text{inv}(v) \land \text{tcp}(v)$, where only inequalities in $\text{inv}$ are considered, and $\text{initStep} = \text{init}(v)$), an instantiation of $\text{CIFTransition}$ in $e_{\mu\text{CIF}}.X_m$ for each $e \in E$ including an equality constraint condition $\Rightarrow g$, and the correct connection of the connector ports using connect equations.
   b. For each variable update $(W, r)$ in each $e \in E$, include a when statement (if $(W, r)$ is an assignment) or a when equation (if $(W, r)$ is a reinitialization) into $e_{\mu\text{CIF}}.eq$ with the condition $\text{CIFTransition}.signalFired$, where $\text{CIFTransition}$ must be replaced with the instance name of the transition $e^{13}$.
   c. For each mode $v \in V$ of $\alpha_i$, add the equalities within the inv mapping to $e_{\mu\text{CIF}}.eq$ using conditional equation conditions that ensure that the equations only become active if the corresponding mode is active.
   d. Add all variables $x \in X$ to $e_{\mu\text{CIF}}$ using the following scheme: (a) if $\text{dtype}(x) = \text{disc}$ and $x$ is not modified anywhere within $\mathcal{A}$, $\text{dtype}_m(x) = \text{parameter}$, (b) if $\text{dtype}(x) = \text{disc}$ and $x$ is modified somewhere in $\mathcal{A}$, $\text{dtype}_m(x) = \text{discrete}$, (c) else $\text{dtype}_m(x) = \text{continuous}$.

3. If $\alpha_i$ is of type $\alpha_1 || \alpha_2$: Apply step 2 to $\alpha_1$ and $\alpha_2$.
4. If $\alpha_i$ is of type $\alpha \triangleright \triangleright \alpha$: Initialize the corresponding variables accordingly in Modelica.
5. If $\alpha_i$ is of type $[y \ldots z]$: Add the corresponding variables to $e_{\mu\text{CIF}}.X_m$.
6. If $\alpha_i$ is of type $D_{x_G}(\alpha)$: Set $\text{dtype}_m(x)$ as described in step 2.

11The $\mu$CIF chooses non-deterministically between several enabled transitions.
12Two variables must be used here, because without this construct, the system of Boolean equations with only one variable would contain algebraic loops that are not allowed in Modelica, see [10]. The $\text{pre}$ operator is used to break the algebraic loop. The Modelica function $\text{anyTrue} \in C_p$ ([10]) returns true if any element of its input vector is true, and the function $\text{fireSLE} \in C_f$ checks if any fired transitions are self-loops of the mode (in which case the mode must remain active).
13If a variable is modified several times within an equation section, when – elsewhen constructs must be employed (although this is currently not supported by most simulators).

14With the exception of external $\mu$CIF functions that must be connected manually to Modelica since the CIF does not yet offer a standardized interface.
15Since $\mu$CIF does not possess a concrete textual format, the CIF model is implemented in the concrete CIF format in a µCIF-like structure.

The transformation described in this paper has been applied to a highly challenging, large-scale hybrid model of a four-tank system under discrete control. The system was modeled in gPROMS, transformed algorithmically to the CIF (using the approach described in [7] and equipped with a logic controller in the Sequential Function Chart (SFC) formalism (see [4])) that was automatically transformed to the CIF. The major challenge posed by this system lies in the highly complex discrete dynamics, as the SFC model contains a Programmable Logic Control (PLC) hardware execution model which makes the discrete dynamics highly complex (see [18]). The CIF model and the transformed Modelica model are available for download here: http://goo.gl/xs2HV

5. APPLICATION EXAMPLE

The transformation described in this paper has been applied to a highly challenging, large-scale hybrid model of a four-tank system under discrete control. The system was modeled in gPROMS, transformed algorithmically to the CIF (using the approach described in [7] and equipped with a logic controller in the Sequential Function Chart (SFC) formalism (see [4])) that was automatically transformed to the CIF. The major challenge posed by this system lies in the highly complex discrete dynamics, as the SFC model contains a Programmable Logic Control (PLC) hardware execution model which makes the discrete dynamics highly complex (see [18]). The CIF model and the transformed Modelica model are available for download here: http://goo.gl/xs2HV.
very large subset of the Compositional Interchange Format (CIF) for Hybrid Systems to the Modelica language. The algorithm transforms the discrete automaton structure of the CIF into a set of Boolean equations, while the continuous parts are mapped into suitable Modelica equation constructs. Combined with a transformation from Modelica to the CIF that was developed in previous work, this transformation is beneficial for both formalisms: the CIF gains support for modeling and simulation environments with a large model library, while Modelica receives access to numerous model-based tools that are connected to the CIF. The transformation was illustrated on a realistic hybrid example that exhibits highly complex discrete dynamics.

Future work includes (among others) (a) the implementation of this transformation algorithm in the Eclipse-based CIF tool set (http://goo.gl/8swlC), (b) the development of methods to avoid the blow-up of model sizes when several transformations are applied in sequence, (c) the definition of an external function interface for the CIF, and (d) the connection of other languages (such as EcosimPro) to the CIF.

REFERENCES


