MODELING AND SIMULATION USING THE COMPOSITIONAL INTERCHANGE FORMAT FOR HYBRID SYSTEMS

C. Sonntag[1], R.R.H. Schiffelers[2], D.A. van Beek[2], J.E. Rooda[2], S. Engell[1]
[1]Technische Universität Dortmund, Germany, [2]Technische Universiteit Eindhoven, Netherlands

Corresponding author: C. Sonntag, Process Dynamics and Operations Group,
Dept. of Biochemical and Chemical Engineering, Technische Universität Dortmund 44227 Dortmund, Germany
c.sonntag@bci.tu-dortmund.de

Abstract. One of the major challenges towards a broad industrial acceptance of hybrid systems techniques and tools is the large number of distinct modeling formalisms and the resulting manual effort for the tool-based solution of many complex design or analysis tasks. A promising approach to achieve inter-operability between hybrid systems tools is to develop automatic translations of their formalisms via a general interchange format with sufficiently rich syntax and semantics. This paper gives an intuitive introduction to such a formalism, the *Compositional Interchange Format* (CIF) for general hybrid systems, that was recently developed within the European Network of Excellence HYCON. The concepts of the CIF are illustrated using an interesting example, the hybrid model of a supermarket refrigeration system. This system exhibits both, nonlinear DAE dynamics as well as significant discrete dynamics, and serves as a challenging case study for hybrid control techniques in several European research projects. Furthermore, the CIF tool set that provides simulation and visualization capabilities is introduced.

1 Introduction

In recent years, a large variety of techniques and tools have been developed to support the design and analysis of hybrid systems, ranging from rather prototypical academic tools that mostly serve as test beds for hybrid systems research to very mature and powerful modeling, simulation, design, and analysis environments. Most of these tools are based on different modeling formalisms that often possess specific features and semantics which are tailored to a certain application domain or to the methods that are implemented by the tools. Due to the lack of a common unifying formal basis, most tools can not be combined directly to solve a complex design or analysis problem. Hence, a tool-based solution of many complex design or analysis tasks involves the manual translation between different modeling formalisms. Such translations are usually very time-consuming and require a profound familiarity with the underlying modeling formalisms, and this problem is currently one of the major obstacles for a broader acceptance of hybrid systems tools in industrial applications.

One possibility to achieve inter-operability between hybrid systems tools is to define bi-lateral transformations between their underlying modeling formalisms. However, if the number of considered formalisms is large, this approach is infeasible due to the large number of bi-lateral transformations that must be defined (see Fig. 1 (a)). If instead a generic modeling formalism (an *interchange format*) is defined that is general and rich enough to represent the syntactic and semantic properties of the modeling languages under consideration, the implementation effort is reduced drastically, as shown in Fig. 1 (b).

The integration of hybrid systems tools via interchange formats has been investigated in several research projects. In the DARPA MOBIES project, the *Hybrid Systems Interchange Format* (HSIF) was defined [1] that employs a network of hybrid automata for model representation, and the abstract semantics of an interchange format based on the *Metropolis meta model* [2] is described in [3]. More recently, the *Compositional Interchange Format* (CIF) [4, 5] for a general class of hybrid systems has been developed within the European Network of Excellence HYCON [6].

The CIF has been developed with two major purposes in mind - to provide a generic modeling formalism (and appropri-

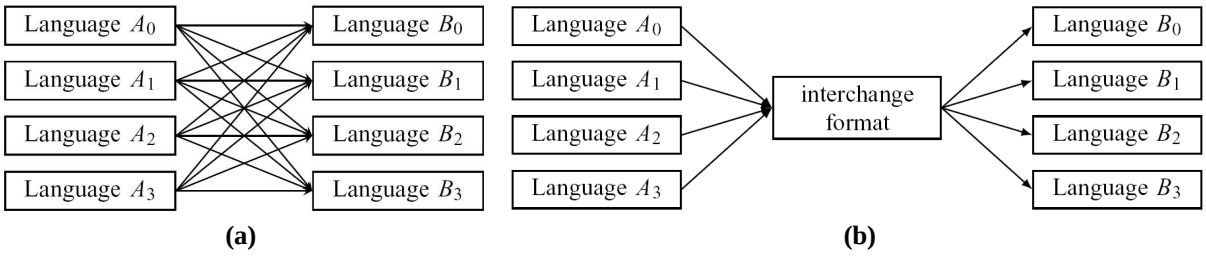


Figure 1: Model transformations without (a) and with (b) an interchange format.

ate tools) for a wide range of general hybrid systems, and to establish inter-operability of a wide range of tools by means of model transformations. The CIF language contains, among others, modeling primitives for the specification of:

- Different classes of variables (discrete, continuous, and algebraic) that differ in their allowed continuous-time behavior (trajectories) during time passing as well as during instantaneous changes (e.g. assignments).
- Continuous dynamics that are specified as fully implicit DAEs.
- Steady-state initialization, where the initial state is the solution of the set of DAEs such that all derivatives equal zero.
- Different kinds of urgency (urgency allows/restricts the passing of time up to a certain point), including many urgency concepts found in literature such as invariants from hybrid (and timed) automata, triggering guards, deadline predicates, and urgent actions.
- Interaction between parallel processes using different mechanisms, such as shared variables, synchronizing actions, and communication via channels.
- Structural information, such as classification of variables as input, output, and/or internal and external.
- Process re-use (automaton definition/instantiation) and hierarchy.

As the name indicates, the CIF is equipped with formal semantics that is *compositional*, i.e. the semantics of a model component is entirely specified in terms of the semantics of its subcomponents. The CIF mainly differs from the HSIF and from the METROPOLIS-based format in that its formal semantics defines the *mathematical meaning* of a hybrid model and is independent of implementation issues and limitations, such as e.g. circular dependencies and algebraic loops. The CIF serves as the basis of the new European research project MULTIFORM [7] whose main objective is the integration and the support for interoperability of tools and methods based on different modeling formalisms in order to make a significant step towards integrated coherent tool support for the design of large complex controlled systems from the first concept to the implementation and further on over their entire life cycle. Within MULTIFORM, algorithms and tools for the translation to/from the CIF will be defined for a large variety of modeling languages, including CHI, GPROMS, MATLAB/SIMULINK, MODELICA, MUSCOD-II, PHAVER, and UPPAAL.

The aim of this paper is twofold: (1) to give an intuitive introduction to the CIF concepts using a complex example, a supermarket refrigeration system under logic control, and (2) to introduce the freely available CIF tool set that was developed recently (see also [8]). In Section 2, the controlled supermarket refrigeration system is described. A hybrid CIF model of this system is specified in Section 3. The tool set and simulation results for the supermarket refrigeration system are detailed in Section 4, and Section 5 concludes the paper.

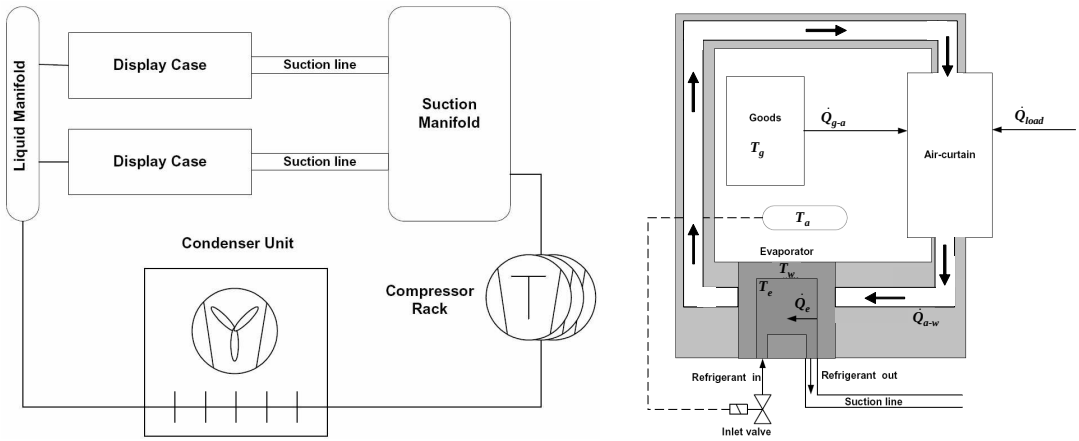
2 A Supermarket Refrigeration System under Logic Control

In this paper, the concepts of the CIF are illustrated using a complex case study, the hybrid model of an industrial supermarket refrigeration system under logic control. Supermarket refrigeration systems are used in most supermarkets to cool edible goods in (often open) display cases to avoid deterioration and to enable easy access for the customers. These systems are hybrid systems due to switching of the continuous dynamics and due to the presence of discretely switched actuators such as expansion valves and compressors. The supermarket refrigeration system considered here is a central industrial case study for hybrid control design techniques in both, the HYCON and the MULTIFORM projects (see e.g. [9, 10, 11, 12]). It represents an excellent example for the illustration of the CIF concepts since it exhibits both, complex nonlinear DAE dynamics and significant discrete dynamics.

Fig. 2 (a) shows a schematic representation of a supermarket refrigeration system. It consists of four major parts: several display cases (two in this case), a compressor rack, a suction manifold, and a condenser. After the liquid refrigerant has been supplied to the display cases through inlet valves, it evaporates and removes heat from the air around the evaporator. The resulting vapor accumulates in the suction manifold and is fed to the condenser via the compressors which increase the pressure of the refrigerant vapor. The thermal energy from the display cases can be removed in the condenser at room temperature since the evaporation temperature of the refrigerant increases with the pressure. Finally, the liquefied refrigerant is fed back to the display cases. The cross-section of an open refrigerated display case is shown in Fig. 2 (b). Cold air is circulated through the display case and forms an air curtain in front of the edible goods. Thermal energy is transferred from the goods to the air curtain ($\dot{Q}_{g-a}$) and, since the temperature of the surrounding air is larger than that of the air curtain, the curtain also absorbs heat from the surroundings ($\dot{Q}_{load}$). The absorbed thermal energy is transported to the evaporator ($\dot{Q}_{a-w}$) in which the refrigerant evaporates and thus takes on the thermal energy ($\dot{Q}_e$)[1].

A model of this system with hybrid dynamics was proposed in [9]. In this model, each display case i, $i \in (1, \ldots n_{dc})$, is described by four continuous state variables: the temperature of the goods ($T_{g,i}$), the temperature of the evaporator wall ($T_{w,i}$), the temperature of the air inside the case ($T_{a,i}$), and the mass of liquid refrigerant within the evaporator of the

[1] Note that in this paper, the time derivative of a variable x is denoted by $\frac{dx}{dt}$ while a dotted symbol $\dot{x}$ refers to a flow of mass or energy.



(a) (b)

Figure 2: A simplified scheme of a supermarket refrigeration system with two display cases (left, taken from [9]) and the cross section of a display case (right).

display case ($m_{r,i}$). Here, n_{dc} is the number of display cases. The dynamics of the condenser unit is not modeled. The pressure in the suction manifold is denoted by P_{suc}, $v_i \in \{v_1, \ldots, v_{n_{dc}}\}$ denotes the state of the inlet valve (closed/open) for the refrigerant of display case i, and $v_{cj} \in \{v_{c1}, \ldots, v_{cn_c}\}$ denotes the state of compressor j (off/on), where n_c is the number of compressors in the compressor rack.

The continuous dynamics is modeled by a lumped-parameter ODE system[2] under the assumption that all display cases are of equal design. The amount of refrigerant in a display case i and the current state of the corresponding inlet valve v_i influence the dynamics of the mass of refrigerant in the display case $m_{r,i}$ according to

$$\frac{dm_{r,i}}{dt} = \begin{cases} \frac{m_{rmax} - m_{r,i}}{\tau_{fill}} & \text{if } v_i = 1, \quad \text{(a)} \\ -\frac{\dot{Q}_{e,i}}{\Delta h_{lg}(P_{suc})} & \text{if } v_i = 0. \quad \text{(b)} \end{cases} \tag{1}$$

Here, the maximum amount of refrigerant each display case can accommodate is represented by m_{rmax}, $\dot{Q}_{e,i}$ is defined in Eq. 3, the specific enthaply of evaporation of the remaining liquefied refrigerant in the evaporator is given by $\Delta h_{lg}(P_{suc})$, and τ_{fill} is a time constant. The display case is filled with refrigerant as long as the inlet valve is open (Eq. 1.a), and after the inlet valve has been closed, the remaining refrigerant evaporates according to Eq. 1.b. The temperature dynamics within the i-th display case is given by:

$$\frac{dT_{g,i}}{dt} = -\frac{\dot{Q}_{g-a,i}}{m_g \cdot cp_g}, \; \frac{dT_{w,i}}{dt} = \frac{\dot{Q}_{a-w,i} - \dot{Q}_{e,i}}{m_w \cdot cp_w}, \; \frac{dT_{a,i}}{dt} = \frac{\dot{Q}_{g-a,i} + \dot{Q}_{load} - \dot{Q}_{a-w,i}}{m_a \cdot cp_a}, \tag{2}$$

with

$$\dot{Q}_{g-a,i} = UA_{g-a} \cdot (T_{g,i} - T_{a,i}), \; \dot{Q}_{a-w,i} = UA_{a-w} \cdot (T_{a,i} - T_{w,i}), \; \dot{Q}_{e,i} = UA_{w-r}(m_{r,i}) \cdot (T_{w,i} - T_e(P_{suc})). \tag{3}$$

Here, $UA_{w-r}(m_{r,i}) = UA_{w-rmax} \cdot m_{r,i}/m_{rmax}$, and m_g, m_w, m_a, cp_g, cp_w, cp_a, UA_{g-a}, UA_{a-w}, and UA_{w-rmax} are constant model parameters. $T_e(P_{suc})$ is the evaporation temperature of the refrigerant. The dynamics of the suction pressure is given by

$$\frac{dP_{suc}}{dt} = \frac{\dot{m}_{in-suc} + \dot{m}_{rconst} - \dot{V}_c \cdot \rho_{suc}(P_{suc})}{V_{suc} \cdot \frac{d\rho_{suc}}{dP_{suc}}(P_{suc})} \quad \text{with} \quad \dot{m}_{in-suc} = \sum_{i=1}^{n_{dc}} \frac{\dot{Q}_{e,i}}{\Delta h_{lg}(P_{suc})}, \quad \dot{V}_c = \eta_{vol} \cdot V_d \cdot \underbrace{\sum_{i=1}^{n_c} \frac{v_{ci}}{n_c}}_{c_{cap}}. \tag{4}$$

The total mass flow of refrigerant from all display cases into the suction manifold is given by $\dot{m}_{in-suc}$, and $\dot{m}_{rconst}$ is an external disturbance that represents an additional flow of refrigerant from other unmodeled cooling facilities into the suction manifold. $\rho_{suc}(P_{suc})$ is a nonlinear refrigerant-dependent function modeling the density of the vapor in the suction manifold, and the volume flow $\dot{V}_c$ from the suction manifold is computed using the constant model parameters η_{vol} and

[2] See [9] for a more detailed description of the model.

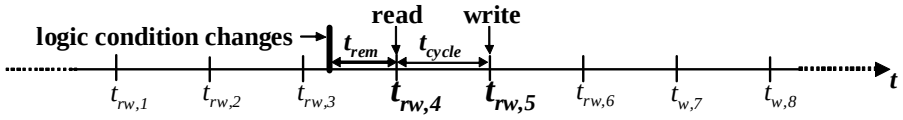


Figure 3: Schematic representation of the cycle-driven execution model for PLCs. It is assumed that reading and writing occurs simultaneously at the time instants $t_{rw,i}$, and that t_{cycle} is constant.

V_d and the displacement capacity c_{cap} of the running compressors in relation to the overall number of compressors n_c in the system. Here, it is assumed that all compressors are of equal capacity.

The controlled variables of the system are the pressure inside the suction manifold (P_{suc}) and the temperatures of the air inside the display cases ($T_{a,i}$). As the system never reaches a steady state, the control goal is not to track setpoints, but to maintain these variables within specified bounds. This is usually achieved by switching the discrete inputs depending on logic conditions that are defined over the continuous variables of the system. In industry, such *logic controllers* are typically described using dedicated programming languages, such as *Function Block Diagram* (*FBD*) or *Sequential Function Chart* (*SFC*) [13] which can then be implemented in a *Programmable Logic Controller* (*PLC*), a specialized hardware control system. These systems usually operate in a cyclic execution mode in which (1) the input variables are read, (2) the control program is executed, and (3) the output variables are assigned new values. This means that the inputs and the outputs are only updated at constant or varying time intervals. In the setting that is illustrated in Fig. 3, an update of the input and output variables is executed only at the time instants $t_{rw,i}$ with a constant cycle time t_{cycle}. The change of the (value of a) logic condition is only recognized by the control system after a delay of t_{rem} time units, and since the execution of the control program takes another cycle, the reaction to the input change only occurs after $t_{rem} + t_{cycle}$ time units. An algorithm has been developed that automatically translates a logic controller given as an SFC (including the cyclic PLC execution model) into a hierarchical CIF model in a procedure that resembles the approach described in [14, 15, 16]. However, since the description of the complete translation scheme is beyond the scope of this paper, here a simplified control system (including a cyclic execution model) is used. This control system was designed for a supermarket refrigeration system with two display cases and two compressors and is shown in Fig. 4.

3 Modeling the Supermarket Refrigeration System using the CIF

A CIF model can be specified using two different formats: 1) the *abstract* format that formally defines the syntax and semantics of a CIF model in terms of an interchange automaton in a mathematically sound way and thus enables correctness proofs of translations to/from other formalisms, and 2) the *concrete* format that provides a user-friendly syntax that adds the concepts of hierarchy and reusability, improves the readability of CIF models, and thus facilitates the manual modeling process. The semantics of the concrete format is defined by means of a mapping to the abstract format, see [17], whereas the formal semantics of the abstract format is defined in a structured operational semantics (SOS) style [18], see [5]. In this paper, we do not use the CIF as an interchange formalism, but we illustrate its usability as a generic modeling formalism. Therefore, we use the concrete format to model the supermarket refrigeration system.

In the remainder of this section, the model is described, where the syntax and semantics of language elements are explained informally at their first usage[3]. For clarification, a graphical representation of the same CIF model is shown in Fig. 4. In this figure, a model (or automaton instantiation) is represented as a solid box that is labeled with the name of the model (or automaton instantiation). Its internal declarations are listed in the upper left corner, and its external declarations are represented as ports on the borders of the box. The shape of a port depends on the type of declaration (see legend).

The model definition called `supermarket` is as follows:

```
model supermarket =
|[ extern var T_a1, T_a2, P_suc         : cont real = (277.5, 274.5, 1.4)
            ; v_1, v_2, v_c1, v_c2      : disc bool = (false, false, true, false)
   intern var T_g1, T_g2, T_w1, T_w2    : cont real = (275, 275, 273, 273)
            ; m_r1, m_r2                : cont real = (0.3, 0.5)
            ; Q_load, m_rconst          : disc real = (3000, 0.2)
   connect {T_g1, dc1.T_g}, {T_g2, dc2.T_g}
         , {T_w1, dc1.T_w}, {T_w2, dc2.T_w}
         , {T_a1, dc1.T_a, contr.T_a1}, {T_a2, dc2.T_a, contr.T_a2}
         , {P_suc, sm.P_suc, dc1.P_suc, dc2.P_suc, contr.P_suc}
         , {v_c1, sm.v_c1, contr.v_c1}, {v_c2, sm.v_c2, contr.v_c2}
```

[3]Note that the description of the concrete format in this paper is non-exhaustive. Please refer to [17] for a more detailed description of the complete specification.

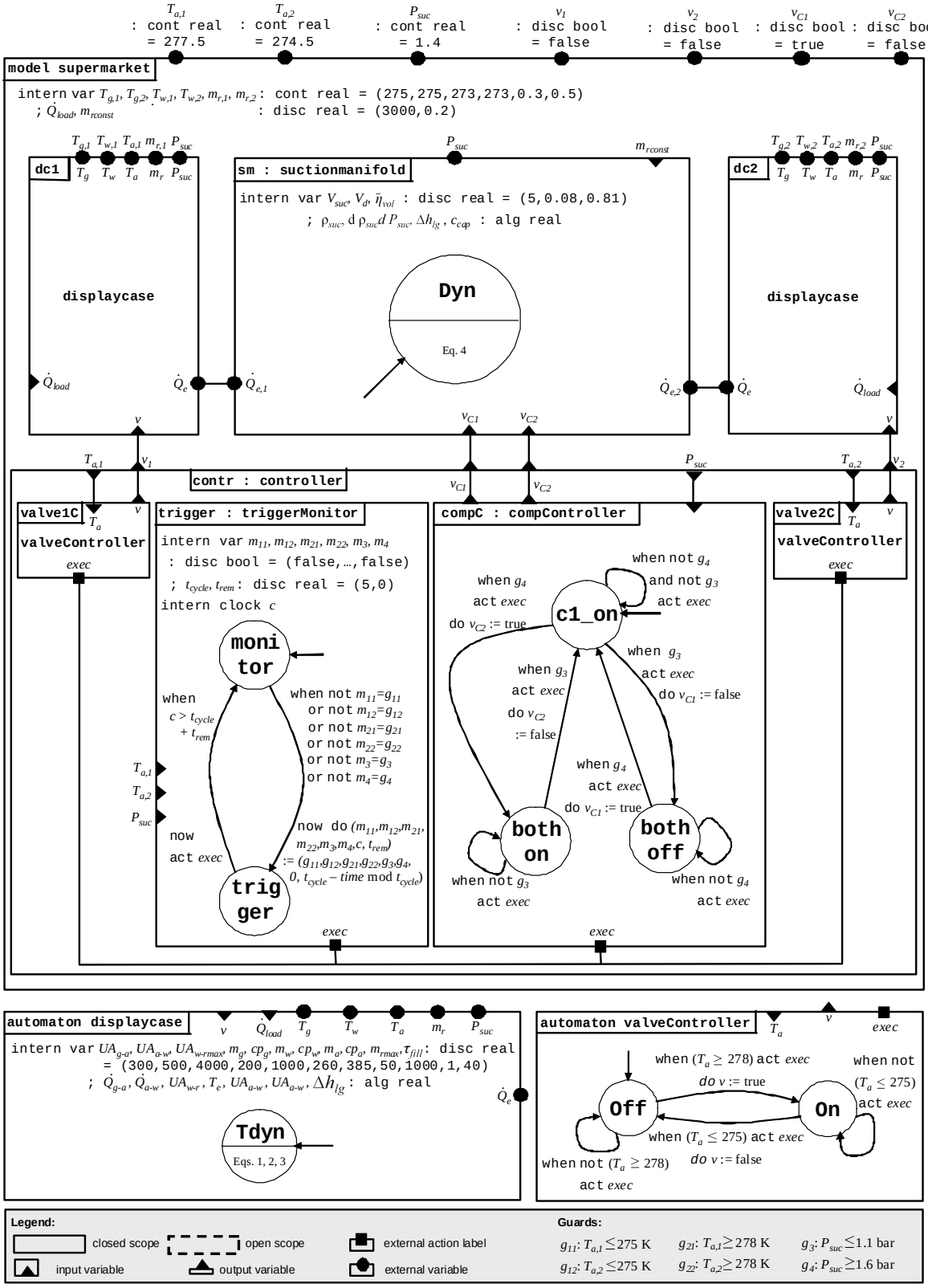


Figure 4: A graphical representation of the CIF model of the supermarket refrigeration system with two display cases and two compressors.

```
        , {v_1, dc1.v, contr.v_1}, {v_2, dc2.v, contr.v_2}
        , {Q_load, dc1.Q_load, dc2.Q_load}, {m_rconst, sm.m_rconst}
        , {m_r1, dc1.m_r}, {m_r2, dc2.m_r}, {sm.Q_e1, dc1.Q_e} , {sm.Q_e2, dc2.Q_e}
:: dc1   : displayCase
|| dc2   : displayCase
|| sm    : suctionManifold
|| contr : controller
]|
```

The model definition consists of external variable declarations, internal variable declarations, connect sets and (a parallel composition of) automaton instantiations modeling the suction manifold, the display cases and the controller, respectively. A variable declaration contains the name of the variable, its dynamic type that can be discrete (`disc`), continuous (`cont`), or algebraic (`alg`), and its static type, such as `real` denoting a real number, or `bool` denoting a boolean, and optionally its initial value.

The main differences between discrete, continuous, and algebraic variables are as follows: First, the values of discrete variables remain constant when model time progresses, the values of continuous variables may change according to a continuous function of time when model time progresses, and the values of algebraic variables may change according to a discontinuous function of time. Second, the values of the discrete and continuous variables do not change in action transitions unless such changes are explicitly specified, for example by assigning a new value. The values of algebraic variables can change arbitrarily in action transitions, unless such changes are explicitly restricted, for example by assigning a new value. Third, there is a difference between the different classes of variables with respect to how the resulting values of the variables in a transition relate to the starting values of the variables in the next transition. The resulting value of a discrete or continuous variable in a transition always equals its starting value in the next transition. For algebraic variables, there is no such relation. In most models, the values of discrete variables are defined by assignments, whereas the values of algebraic variables are defined by invariants ((in)equalities).

An automaton instantiation contains a label and the name of the automaton definition to be instantiated. The label is used as a prefix to refer to the variables that are declared as external in the automaton definition. E.g. `dc1.T_g` denotes the temperature of the goods in display case `dc1`.

A connect set connects the contained variables. As an example, the variable identifiers `T_g1` and `dc1.T_g` denote the same variable since they are within the connect set `{T_g1, dc1.T_g}`.

A display case is modeled by means of an instantiation of the automaton definition `displayCase`. The input variables `v` and `Q_load` model the state of the inlet valve of the case and the absorbed heat from the surroundings. These variables are modeled as input variables since they are considered as inputs for the display case. The body of the automaton definition consists of a single mode that is used to model the continuous-time dynamics of the system. Each mode can be labeled with flow predicates and invariant predicates. The flow predicates usually contain the ODE, and the invariant predicates usually contain the algebraic constraints that should hold during time-passing. Unlike other hybrid automata formalisms, there is no semantic distinction between these two types of predicates. The flow and invariant predicates should always hold (during time-passing as well as when entering the mode). In the model, mode `Tdyn` contains as flow the predicates from Eq. 2, and as invariant the predicates from Eqs. 1 and 3.

```
automaton displayCase =
|[ input var v      : bool
           ; Q_load : real
   extern var T_g, T_w, T_a, m_r, P_suc : cont real
            ; Q_e                       : alg real
   intern var UA_g_a, UA_a_w, UA_w_rmax : disc real = (300, 500, 4000)
            ; m_g, cp_g                 : disc real = (200, 1000)
            ; m_w, cp_w                 : disc real = (260, 385)
            ; m_a, cp_a                 : disc real = (50, 1000)
            ; m_rmax, tau_fill          : disc real = (1, 40)
            ; Q_g_a, Q_a_w, UA_w_r, T_e, delta_h_lg : alg real
:: |( mode Tdyn =
       flow dot T_g    = - Q_g_a / (m_g * cp_g)
          & dot T_w    = (Q_a_w - Q_e) / (m_w * cp_w)
          & dot T_a    = (Q_g_a + Q_load - Q_a_w) / (m_a * cp_a)
       inv  Q_g_a      = UA_g_a * (T_g - T_a)
          & Q_a_w      = UA_a_w * (T_a - T_w)
          & Q_e        = UA_w_r * (T_w - T_e)
          & UA_w_r     = UA_w_rmax * m_r / m_rmax
          & T_e        = -4.3544 * P_suc^2 + 29.2240 * P_suc - 51.2005 + 273.15
          & delta_h_lg = (0.0217 * P_suc^2 - 0.1704 * P_suc + 2.2988) * 100000
          & dot m_r    = (     v -> ( m_rmax - m_r) / tau_fill
```

```
                          | not v -> -Q_e / delta_h_lg
                          )
   :: Tdyn
   )|
]|
```

The switching dynamics for `dot m_r` that depend on the value of variable `v` is defined by means of a conditional expression.

The suction manifold is modeled by means of automaton definition `suctionManifold`.

```
automaton suctionManifold =
|[ input var v_c1, v_c2 : bool
           ; m_rconst   : real
   extern var P_suc      : cont real
            ; Q_e1, Q_e2 : alg real
   intern var rho_suc, del_rho_suc_P_suc, delta_h_lg, c_cap : alg real
            ; V_suc, V_d, n_vol : disc real = (5, 0.08, 0.81)
:: |( mode Dyn = inv dot P_suc         = ((Q_e1 + Q_e2) / delta_h_lg + m_rconst -
                                           (n_vol * V_d * c_cap) * rho_suc)
                                         / (V_suc * del_rho_suc_P_suc)
                 & rho_suc             = 4.6073 * P_suc + 0.3798
                 & del_rho_suc_P_suc = -0.0329 * P_suc^3 + 0.2161 * P_suc^2 -
                                         0.4742 * P_suc + 5.4817
                 & delta_h_lg          = (0.0217 * P_suc^2 - 0.1704 * P_suc + 2.2988)
                                          * 100000
                 & c_cap               = ( not v_c1 -> 0 | v_c1 -> 0.5 ) +
                                         ( not v_c2 -> 0 | v_c2 -> 0.5 )
   :: Dyn
   )|
]|
```

It consists of a single mode definition with Eq. 4 as invariant.

The controller consists of two valve controllers, a compressor controller, and a trigger-monitor automaton.

```
automaton controller =
|[ input var T_a1, T_a2, P_suc    : real
   output var v_1, v_2, v_c1, v_c2 : disc bool
   connect {T_a1, trigger.T_a1, valve1C.T_a}, {T_a2, trigger.T_a2, valve2C.T_a}
         , {P_suc, trigger.P_suc, compC.P_suc}
         , {v_1, valve1C.v}, {v_2, valve2C.v}
         , {v_c1, compC.v_c1}, {v_c2,compC.v_c2}
         , {trigger.exec, valve1C.exec, valve2C.exec, compC.exec}
:: trigger : triggerMonitor
|| valve1C : valveController
|| valve2C : valveController
|| compC   : compController
]|
```

The controller adjusts the state of the inlet valves of the refrigerant of the display cases (`v_1`, `v_2`) as well as the state of the compressors (`v_c1`, `v_c2`). Hence, these variables are declared as output variables.

The trigger-monitor automaton triggers the events (threshold crossings) of the system. It consists of two modes and two edges. Clock variable `c` is used to model time-passing until the next cycle time of the PLC. The initial value of a clock variable equals 0, and during time-passing, its value changes with rate 1. Clock variables can be reset by means of assignments labeled on edges (see below).

An edge `when` *guard* `now act` *action* `do` *assignment* `goto` *targetmode* starts with the keyword `when`, followed by a (optional) *guard* (boolean expression). If the value of the *guard* evaluates true, the edge is enabled. The (optional) keyword `now` denotes that the edge is urgent, which means that time-passing is not allowed when the *guard* evaluates true. When the edge is taken, it executes the action *act* (if the action is omitted, it executes a pre-defined action *tau*), while updating the value of the variables according to the (optional) *assignment*. Finally, the mode *targetmode* is enabled.[4]

[4]In Fig. 4, modes are visualized by means of circles labeled with the name of the mode, and edges are represented as arrows between modes and are labeled with their guard, action, and update (see below). If the edge is urgent, the arrow is also labeled with the the keyword *now*. A connect set is represented as one or more lines connecting the ports that belong to the elements of the connect set. If two ports have the same name, they are connected implicitly, and a line may be omitted.

```
automaton triggerMonitor =
|[ input var T_a1, T_a2, P_suc      : real
   intern var m11,m21,m12,m22,m3,m4 : disc bool = (false, false, false, false, false, false)
            ; t_cycle, t_rem : disc real = (5, 0)
   intern clock c
   extern act exec
:: |( mode monitor = when m11/=(T_a1 <= 275) or m21/=(T_a1 >= 278) or
                          m12/=(T_a2 <= 275) or m22/=(T_a2 >= 278) or
                          m3/=(P_suc <= 1.1) or m4/=(P_suc >= 1.6)
                     now do
                     (m11, m21, m12, m22, m3, m4, c, t_rem) :=
                     (T_a1 <= 275, T_a1 >= 278, T_a2 <= 275, T_a2 >= 278,
                      P_suc <= 1.1, P_suc >= 1.6, 0, 2*t_cycle - time mod t_cycle)
                     goto trigger
         , trigger = when c >= t_rem now act exec goto monitor
   :: monitor
   )|
]|
```

In the trigger automaton, time passes in mode `monitor` until a threshold is crossed. Then, the assignment is executed, and subsequently time passes until `c >= t_rem`. The action `exec` is executed synchronously with the valve and compressor controllers since they all share the action `exec`. Note that the controller automata are only triggered if a guard condition has changed and not in every cycle of the PLC. Since the cycle time of the logic controller is usually much smaller than the time scale of the plant, this *event-driven* simulation approach provides a considerable performance advantage over the cycle-driven approach in which the controller automata are executed after every time cycle of the PLC.

The six internal discrete boolean memory variables used in this automaton definition enable a concise description of this trigger automaton. Without these discrete variables, one would need an automaton for each OR-clause of the guard of the edge (`m11/=(T_a1 <= 275) or ... or m4/=(P_suc >= 1.6)`). This would result in 6 automata, each containing 4 locations and 6 edges. Alternatively, one could use a single automaton with 128 ($=2^{6+1}$) locations and 4096 ($=\frac{2^{6+1}}{2}\cdot 2^6$) edges.

The valve controllers switch the inlet valves of the display cases to keep the air temperatures within a tight region.

```
automaton valveController =
|[ input var T_a : real
   output var v  : disc bool
   extern act exec
:: |( mode Off   = when     T_a >= 278 act exec do v := true goto On
                   when not T_a >= 278 act exec goto Off
         , On    = when     T_a <= 275 act exec do v := false goto Off
                   when not T_a <= 275 act exec goto On
   :: Off
   )|
]|
```

The valves are closed initially. If the air temperature T_a in the corresponding display case exceeds an upper threshold, the valve controllers open the valves to reduce the temperature (`T_a >= 278`) when the `exec` action is initiated. As soon as the temperatures have dropped to 275 K (`T_a <= 275`), the valves must be closed again (triggered by `exec`) to avoid the freezing of the edible goods.

The compressor controller adjusts the state of both compressors.

```
automaton compController =
|[ input var P_suc : real
   output var v_c1, v_c2 : disc bool
   extern act exec
:: |( mode c1_On   = when     P_suc >= 1.6 act exec do v_c2 := true  goto bothOn
                    when     P_suc <= 1.1 act exec do v_c1 := false goto bothOff
                    when not P_suc >= 1.6 and not P_suc <= 1.1 act exec goto c1_On
         , bothOn  = when     P_suc <= 1.1 act exec do v_c2 := false goto c1_On
                    when not P_suc <= 1.1 act exec goto bothOn
         , bothOff = when     P_suc >= 1.6 act exec do v_c1 := true  goto c1_On
                    when not P_suc >= 1.6 act exec goto bothOff
   :: c1_On
   )|
]|
```

Initially, compressor 1 is switched on (mode `c1_On`). When the `exec` action is initiated and the suction pressure exceeds 1.6 (`P_suc >= 1.6`), compressor 2 is switched on (`v_c2 := true`). In case the suction pressure is lower than 1.1 bar, compressor 1 is shutdown (mode `bothOff`). If the suction pressure is between 1.1 bar and 1.6 bar, the states of both compressors remain unchanged. Similar behavior is modeled in modes `bothOn` and `bothOff`.

4 Tooling for the CIF

To support the design and analysis of CIF models, the following tools have been developed:

- a *compiler* that takes as input a CIF model specified in the concrete format and translates it to a CIF model in the abstract format. It implements the mapping as defined in [17].
- a model *visualizer* that takes as input a CIF model specified in the abstract format and visualizes it graphically.
- a *stepper* that takes as input a CIF model specified in the abstract syntax and calculates its dynamic behavior resulting in a hybrid transition system that consists of action and time transitions. It implements the SOS rules as defined in [5].
- a *simulator* that provides a front-end to the stepper. Several options exist to customize the output of the simulator, such as the visualization of the trajectories of the model variables during and/or after the simulation, or the visualization of the performed discrete actions.

For simulation purposes, a concrete CIF model is first mapped into the abstract format using the compiler. The action and delay transitions are calculated using symbolic and/or numerical solvers. The simulator can be run in different modes:

- *User-guided mode*: In this mode, the simulator shows all possible transitions, and the user may choose which transition to execute.
- *Automatic mode*: In this mode, the non-deterministic choices between transitions are resolved automatically by the simulator, or the simulator can simulate all possibilities (exhaustive simulation/state-space generation).

In both modes, the simulator can be parameterized with the solvers to be used, including solver-specific options, and the requested simulation output.

The CIF toolset is designed in such a way that it can be integrated with third-party tools. The main reason for this is to re-use existing applications and libraries. For the implementation of the tools, the PYTHON [19] programming language and several PYTHON packages have been used.

For visualization of a CIF model specified in the abstract format, the graph-filtering and graph-rendering tools of GRAPHVIZ [20] are used. GRAPHVIZ is an open toolkit for graph visualization. It is developed at AT&T Labs Research. The GRAPHVIZ tools use a common language to specify attributed graphs. This language is called *Libgraph*, but it is probably better known as the *dot* format, after its best-known application.

For the computation of solutions for a) discrete updates to model variables (e.g. assignments), b) the initialization problem (IP) and c) the initial-value problem (IVP), the symbolic solving capabilities of the mathematical software package MAPLE [21] can be used. More specifically, explicit equations and boolean expressions are evaluated using PYTHON, and implicit equations and algebraic loops are solved by MAPLE. For the IVP, numerical solutions can be obtained using the numerical solver DASSL [22]. For the visualization of simulation results, in particular the trajectories of the model variables, the portable command-line driven interactive data and function plotting utility GNUPLOT [23] is used.

The CIF simulator was used to simulate the model of the supermarket refrigeration system. The simulation results are shown in Fig. 5. Evidently, the switching of the valves and of the compressors meet the expected pattern, and the time trajectories are similar to those that were obtained with other simulation tools. In particular, Fig. 5 (b) demonstrates the accuracy of the simulator since it shows that the logic controller switches off the second compressor exactly after $t_{rem} + t_{cycle}$ time units have elapsed since the threshold crossing of P_{suc}. The simulation took about 3 minutes on an average desktop PC.

5 Conclusions and Outlook

In this paper, a recently developed interchange format for hybrid systems, the *Compositional Interchange Format* (CIF), is presented, and its concepts are described using a realistic and interesting modeling example, a supermarket refrigeration system under logic control. The CIF supports a very general class of hybrid systems and encompasses many sophisticated concepts, such as fully implicit DAE dynamics, different urgency concepts, parallel composition, re-use, and hierarchy. CIF models can be visualized and simulated using the CIF tool set that was implemented in PYTHON.

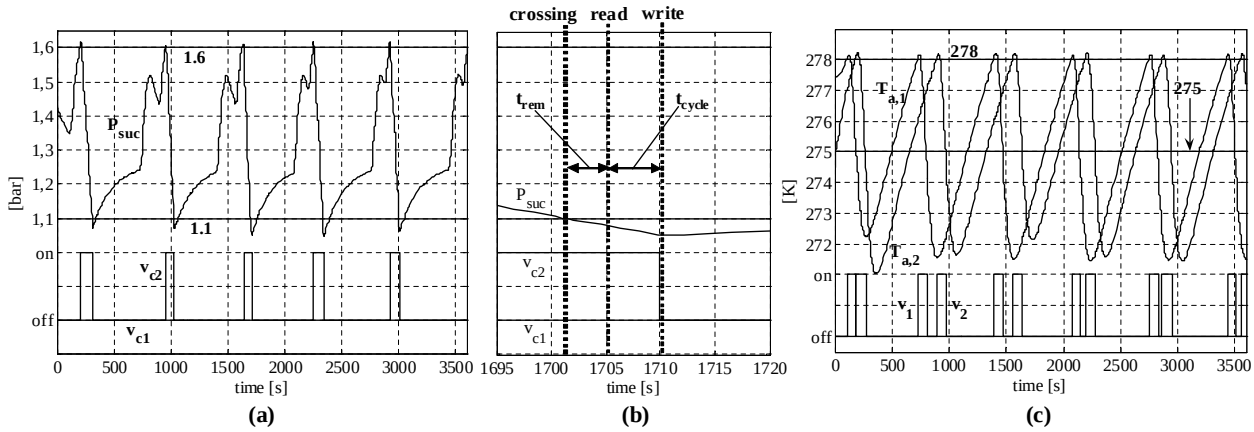


Figure 5: Simulation results: (a) suction pressure and compressor settings, (b) a zoomed version of (a), and (c) air temperatures and valve settings. The cycle time was set to $t_{cycle} = 5\ s$.

The CIF simulator supports both, numerical simulation (using *DASSL*) and symbolic simulation (using MAPLE), and the CIF visualizer is based on the open visualization toolkit GRAPHVIZ.

Within the new European research project MULTIFORM [7], the main objective of which is the integration and the support for inter-operability of tools and methods based on different modeling formalisms, translations will be defined for a large variety of modeling languages, including CHI, GPROMS, MATLAB/SIMULINK, MODELICA, MUSCOD-II, PHAVER, and UPPAAL. Using these (and future) translations, we expect that the CIF will help to significantly increase the applicability of hybrid systems techniques in industrial practice.

6 Acknowledgements

This work was partially done in the framework of the HYCON Network of Excellence, contract number FP6-IST-511368, 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, as part of the ITEA project Twins 05004, and as part of the Collaborative Project MULTIFORM, contract number FP7-ICT-224249.

7 References

[1] MoBIES team: HSIF semantics. Technical report, University of Pennsylvania (2002) internal document.

[2] Metropolis Project Team: The Metropolis meta model. Technical Report UCB/ERL M04/38, University of California, Berkeley (2004) internal document.

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

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

[5] Beek, D.A.v., Reniers, M.A., Rooda, J.E., Schiffelers, R.R.H.: Revised hybrid system interchange format. Technical Report D3.6.3 to the NoE HYCON (2007) http://www.bci.tu-dortmund.de/dyn/hycon3/D363.pdf.

[6] HYCON: Hybrid Control - Taming Heterogeneity and Complexity of Networked Embedded Systems: http://www.ist-hycon.org (2008)

[7] MULTIFORM: Integrated Multi-formalism Tool Support for the Design of networked Embedded Control Systems: http://www.ict-multiform.eu (2008)

[8] Systems Engineering Group TU/e: CIF toolset. http://se.wtb.tue.nl/sewiki/cif (2008)

[9] Larsen, L.F.S., Zamanabadi, R.I., Wisniewski, R., Sonntag, C.: Supermarket refrigeration systems - a benchmark for the optimal control of hybrid systems. Technical report for the Network of Excellence HYCON (2007) http://tinyurl.com/23nrkc.

[10] Larsen, L.F.S., Thybo, C., Izadi-Zamanabadi, R., Wisniewski, R.: Synchronization and desynchronizing control schemes for supermarket refrigeration systems. In: Proc. IEEE Multi-Conf. on Systems and Control (MSC / CCA). (2007) 1414–1419

[11] Sonntag, C., Devanathan, A., Engell, S.: Hybrid NMPC of a supermarket refrigeration system using sequential optimization. In: Proc. 17th IFAC World Congress. (2008) 13901–13906
[12] Sarabia, D., Capraro, F., Larsen, L.F.S., De Prada, C.: Hybrid NMPC of supermarket display cases. To appear in: Control Engineering Practice (2009)
[13] International Electrotechnical Commission (IEC): IEC 61131-3: Programmable Controllers - Programming Languages. (2003)
[14] Bauer, N., Engell, S., Huuck, R., Lohmann, S., Lukoschus, B., Remelhe, M., Stursberg, O.: Verification of PLC programs given as Sequential Function Charts. In: Integration of Software Specification Techniques for Applications in Engineering. Volume 3147 of LNCS. (2004) 517–540
[15] Stursberg, O., Lohmann, S.: Analysis of logic controllers by transformation of SFC into timed automata. In: Proc. 44th IEEE Conf. on Decision and Control / European Control Conference. (2005) 7720–7725
[16] Lohmann, S., Stursberg, O., Engell, S.: Comparison of event-triggered and cycle-driven models for verifying SFC programs. In: Proc. American Control Conference. (2007) 3606–3611
[17] Beek, D.A.v., Reniers, M., Rooda, J.E., Schiffelers, R.R.H.: Concrete syntax and semantics of the compositional interchange format for hybrid systems. In: 17th Triennial World Congress of the International Federation of Automatic Control, Seoul, Korea (2008)
[18] Plotkin, G.D.: A structural approach to operational semantics. Journal of Logic and Algebraic Programming **60-61** (2004) 17–139
[19] Python: http://www.python.org (2005)
[20] Emden, R.G., North, S.C.: An open graph visualization system and its applications to software engineering. Software – Practice and Experience **30** (2000) 1203–1233
[21] MapleSoft: http://www.maplesoft.com (2005)
[22] Petzold, L.R.: A description of DASSL: A differential/algebraic system solver. Scientific Computing (1983) 65–68
[23] Gnuplot: http://www.gnuplot.info (2005)