Hybrid Models of Physical Systems and Discrete Controllers

Martin Otter, Manuel Remelhe, Sebastian Engell, and Pieter Mosterman

A unifying modeling method is presented that (1) extends the declarative, equation based, object oriented modeling approach by discontinuous and variable structure components which arise from abstractions in physical system models and (2) combines this with imperative, reactive, discrete event controllers based on the statechart and the sequential function chart formalisms.

1 Introduction

Design, analysis, and testing of complex dynamic systems, such as robots, aircraft, automatic gearboxes, and multi-product batch plants, increasingly relies on modeling and simulation and requires the integration of different modeling and specification formalisms. For example, consider the primary attitude control surfaces of an airplane as shown in Fig. 1. In order to design, analyze and test the attitute control system under realistic conditions, a system model of the aircraft dynamics (mechanics, aerodynamics, gravity, wind), engines and actuators (electrical motors and hydraulic power systems) is needed. Such models often include nonlinearities which cause large behavior gradients. By abstracting these phenomena into discontinuities, simulation time and identification effort can be reduced significantly. As a result, the model of the physical system combines continuous and discrete behaviors.

Furthermore, systems such as an aircraft include embedded control functionality, e.g., low-level digital PID type control as well as high level supervisory logic and redundancy management. In case of the airplane, there is, e.g., redundancy in the actuators that position the elevators. In turn, these actuators may be controlled by redundant primary flight control units. Control logic is required to ensure that both of the elevators are always controlled by only one actuator with at least some minimal direct link functionality.

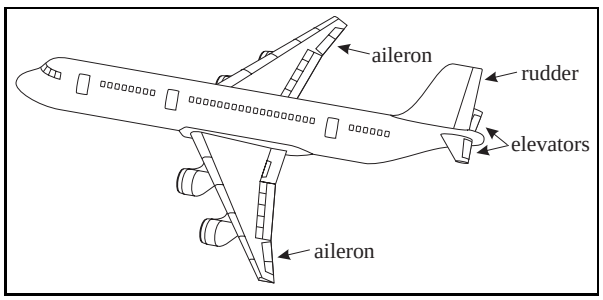

Fig. 1: Primary attitude control surfaces of an airplane.

This article discusses a method that supports appropriate *visual* and *computational* representations of all parts of complex hybrid system models by (1) extending the *declarative*, equation based, object oriented modeling approach [8, 1, 2, 15] by discontinuous and variable structure components which arise from abstractions in physical system models and by (2) combining this with *imperative*, reactive, discrete event formalisms, each accompanied by a dedicated graphical editor, such as the statechart [11] and the sequential function chart [5] formalisms. The approach is illustrated by modeling the dynamics of an aircraft with composition diagrams and part of the redundancy management by statecharts.

2 Object-Oriented Modeling of Physical Systems

In object-oriented modeling, physical systems are defined in a component-oriented way by *composition diagrams* that are close to the schematics used in many engineering disciplines. A typical example is given in Fig. 2 (a screen shot of a Modelica model [15] in the graphical editor of Dymola [7]). It consists of a direct current motor with gearbox and load inertia. The connections between the components describe the real physical connections, e.g., the line connecting the “resistor” and the ”inductor“ characterizes an electrical line whereas the connection line between the “idealGear” and the “loadInertia” characterizes a rigid mechanical connection of two flanges. Components in a composition diagram are hierarchically structured and may contain other composition diagrams. The primitive behavior of a component is specified by differential equations and by algebraic

equations such that the equations of the component are only a function of local variables defined in the component, and of the interface variables. Because of this structure, a component can usually be connected to all other components which have a compatible interface.

Modeling with composition diagrams is conveniently done using Modelica^{TM}, a uniform object-oriented language for modeling of physical systems, designed by the developers of the modeling languages Allan, Dymola, NMF, ObjectMath, Omola, SIDOPS+, and Smile as well as a number of modeling practioners. For details about the Modelica project, tools supporting Modelica, and the free Modelica component libraries, see http://www.Modelica.org/.

A composition diagram is transformed into a form that can be simulated by extracting the equations of all components and by adding the equations that describe the physical connections of the interfaces (e.g. equations of the form $v_1 = v_2$ for generalized potential and equations of the form $i_1 + i_2 + i_3 = 0$ for generalized flow interface variables). This procedure results in a set of differential-algebraic equations of the form:

$$\mathbf{0} = \mathbf{f}(\dot{\mathbf{x}}, \mathbf{x}, \mathbf{y}, t), \tag{1.1}$$

where **x**(t) are variables with derivatives $\dot{\mathbf{x}}(t)$ appearing in the equations and **y**(t) are algebraic variables. Because of the systematic component-oriented approach, Eq. (1.1) typically embodies a large set of equations, where each equation is a function of only a few variables. Effective symbolic transformation techniques can be applied to transform Eq. (1.1) into a reduced form that can be solved more efficiently. This is performed by

1. BLT-partitioning, i.e., by *permuting* variables and equations such that $\dot{\mathbf{x}}, \mathbf{y}$ can be computed in an explicit forward sequence as a function of **x** and t, which may require to solve local *algebraic loops*,
2. tearing, i.e., by reducing the dimensions of the algebraic loops and the number of operations by *variable substitution*,
3. hiding the explicitely solvable algebraic variables from the integrator, i.e., variables y_i which do not appear in an algebraic loop are not known to the integrator.

For details of the algorithms, see for example [8, 20].

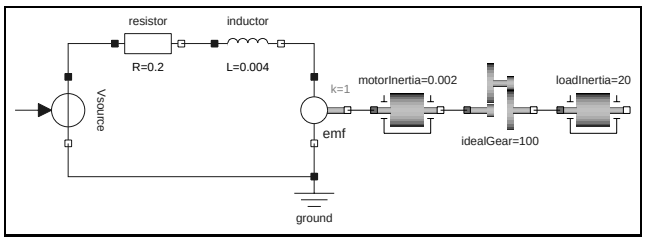

Fig. 2: Direct current motor with gearbox and load inertia.

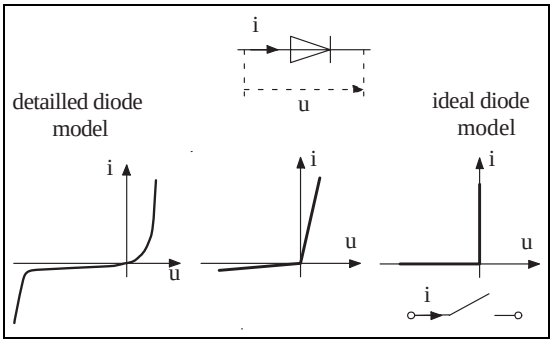

Fig. 3: Different model levels of diode characteristic.

3 Hybrid Systems Arising From Physical Abstractions

In a macroscopic view, physical components can be modeled by continuous behavior. However, these models may contain highly nonlinear behavior which cause large behavior gradients and may operate on widely different time scales. To achieve efficient simulation and to reduce the identification effort, relatively fast dynamic behavior can be abstracted into discontinuities. This is illustrated by different models of a diode shown in Fig. 3, where i is the current through the diode and u is the voltage drop between its pins. A detailed nonlinear characteristic of a diode is shown on the left in Fig. 3. If the detailed switching behavior around the origin is negligible compared to other model phenomena, it is often sufficient to approximate the characteristic by a piecewise linear model as shown in the middle of Fig. 3. This characteristic has a sharp discontinuity in the first derivative. A further abstraction removes the steep gradients and disallows the voltage drop to become positive and the current to become negative, resulting in the ideal diode model on the right in Fig. 3. This corresponds to an ideal switch. This abstraction typically gives a simulation speedup of 1 to 2 orders of magnitude compared to the detailed diode characteristic.

In this section hybrid models which arise from *physical abstractions* are analyzed in more detail. The goal is to include local, component-oriented models of ideal switching elements, such that a modeler can build-up complex models from basic components in a convenient way, where the components and the component connections mimic the structure of the modeled real world elements.

3.1 Example: Modeling an Ideal Diode

The detailed behavior of the diode in the left of Fig. 3 can be modeled by an analytic or tabulated function $i = f(u)$. This functional dependency is no longer valid for the ideal diode characteristic in the right of Fig. 3 because at the origin ($u = 0$) there are infinitely many values for the current i. In the following, different methods are discussed to mathematically describe this behavior.

State transition diagrams

One way to handle the ideal diode is by describing the diode as a switching system where the model switches between the two equations $i = 0$ and $u = 0$. This requires a discrete event switching structure such as, e.g., the state transition diagram in Fig. 4. For details, see for example [2, 9, 17].

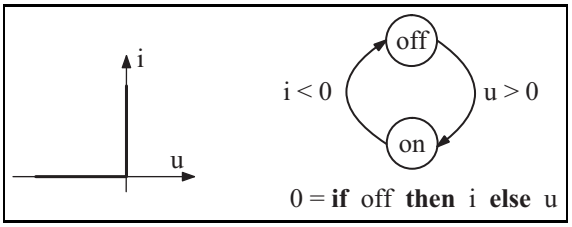

Fig. 4: Ideal diode model described with a state transition diagram.

This approach has the disadvantage that the description of the ideal diode characteristic is *imperative* and not *declarative*. If for example, several ideal diodes are dynamically coupled and one of the transitions will fire, a sequence of transitions in the state transition diagrams of all diodes are performed to determine the new switching structure, i.e., until no transition fires anymore. It turns out that this sequence may not converge, although the physical system has a unique consistent configuration.

Complementarity conditions

An alternative is to describe the ideal diode characteristic in a *declarative* way by two inequalities and one complementarity condition:

$$i \geq 0; \quad -u \geq 0; \quad i \cdot u = 0. \tag{1.2}$$

For some classes of electrical circuits containing ideal diodes it is then possible to transform the equations for the determination of the switching structure of the diodes into a linear complementarity problem:

$$\mathbf{y} = \mathbf{A}\,\mathbf{x} + \mathbf{b}; \quad y_i \geq 0; \quad x_i \geq 0; \quad \mathbf{y}^T\mathbf{x} = 0. \tag{1.3}$$

This approach is advantageous because a consistent switching state is defined by (1.3) and the solution of (1.3) by appropriate algorithms is a separate issue. In contrast, the approach using state transition diagrams can be seen as having the solution algorithm built into the model and therefore the simulation system cannot utilize algorithms with better convergence properties. The complementarity formulation for discontinuous systems was developed by Lötstedt [13] for mechanical systems with unilateral contacts. See also [22, 24] for further developments.

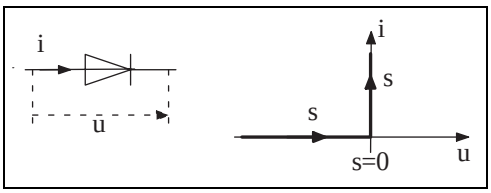

Fig. 5: Ideal diode model described as parameterized curve.

Parameterized curve descriptions

The difficulties with the state transition diagrams can also be avoided by recognizing that a 2-dimensional graph $y = f(x)$ can be described in a more general way by using either an implicit formulation, $0 = f(x,y)$ or by using a *parameterized curve description*

$$x = f_1(s)$$
$$y = f_2(s)$$

with curve parameter s. The latter description is more general and can be used to describe an ideal diode uniquely in a *declarative way*, see Fig. 5, by the equations

$$\mathit{off} = s < 0 \tag{1.4a}$$
$$u = \textbf{if } \mathit{off} \textbf{ then } s \textbf{ else } 0 \tag{1.4b}$$
$$i = \textbf{if } \mathit{off} \textbf{ then } 0 \textbf{ else } s \tag{1.4c}$$

This is a set of 3 equations which relate the 4 unknowns *off*, u, i, s. Equations (1.4) can be seen as a reformulation of the complementarity description (1.2) that allows a direct application of the standard algorithms of object-oriented modeling, such as BLT-partitioning and tearing, because the models are still described by equations and the datatypes of the unknowns are irrelevant for these algorithms. The parameterized curve description has the additional advantage that it allows the description of more general discontinuities than the complementarity formulation.

In order to understand the consequences of parameterized curve descriptions, the ideal diode model is used in the rectifier circuit of Fig. 6, where 4 diodes are connected together, such that a direct current is flowing through the load (R_2, C) driven by an AC voltage source. This circuit has essentially two operational modes: If the source current i_0 is

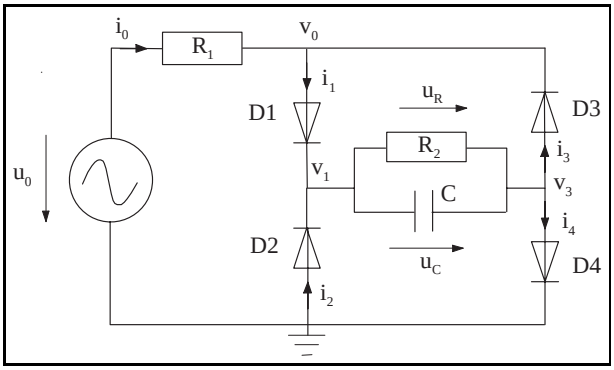

Fig. 6: Rectifier circuit with 4 diodes.

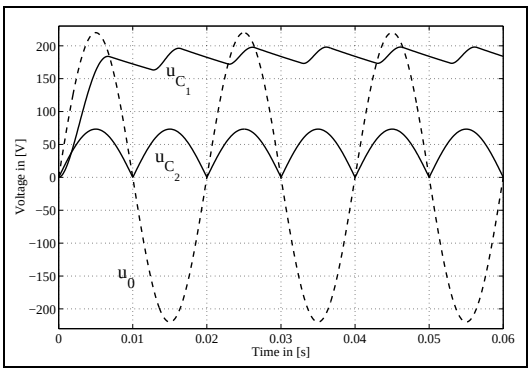

Fig. 7: Simulation results for the rectifier circuit.

positive, diodes 1 and 4 are on (= switches are closed) and diodes 2 and 3 are off. If the source current is negative, diodes 2 and 3 are on, and diodes 1 and 4 are off. In both cases the current flows from left to right, i.e., always in the same direction, through the load (R_2, C). Typical simulation result can be seen in Fig. 7, where the source voltage u_0 is shown together with the voltage drop over the capacitance u_C for two different values of the load[1].

Collecting the equations of all components and connections, and transforming this set of equations with the algorithms sketched in section 2, results in the state space form

$$\dot{u}_c \;=\; f(u_c, t) \tag{1.5}$$

where Table 1 displays the sequence of statements to compute the function $f(u_c(t), t)$. Note, that $s_1(t)$, $s_2(t)$, $s_3(t)$, $s_4(t)$ are the curve parameters of the corresponding ideal diodes and $m_1(t)$, $m_2(t)$, $m_3(t)$, $m_4(t)$ are the boolean variables that characterize the *off* structures of these diodes. It is assumed that the value **true** of a boolean variable is represented by 1 and the value **false** by 0 (e.g. $-m_2/R_2 = -1/R_2$ if $m_2 =$ **true**). As can be seen in Table 1, the sets 1 and 3 of the equations consist of an ordered set of assignment statements which can be evaluated in the given order. However, set 2 is a coupled set of 9 equations which have to be treated together in order to compute the 9 unknown variables $m_1, m_2, m_3, m_4, s_1, s_2, s_3, s_4, w$.[2]

Since integration methods require continuous model equations, all *relations* of a model[3] such as $s_1 < 0$, have to be *fixed* during integration, in order to guarantee that discontinuous changes of variables do not occur. All *relations* are *monitored* and when a relation changes its value, the time instant of the switching point is determined up to a certain precision and the integration is stopped, i.e., an event is localized. In the Modelica language [15] the modeler does not have to explicitly take care of this situation, because by default a change in the value of a relation automatically triggers an event.

[1] $R_1 = 20\,\Omega$; u_{C_1} for $R_2 = 500\,\Omega$, $C = 10^{-4}\,F$; u_{C_2} for $R_2 = 10\,\Omega$, $C = 10^{-6}\,F$

[2] The auxiliary variable w is only introduced in order to write the linear system of equations (2) in the table in one line.

[3] Relations are expressions of the form "$v_1\ op\ v_2$", where v_1 and v_2 are variables or values and op is one of $>, \geq, <, \leq$.

Table 1: Sorted equations of rectifier circuit

	input: $t,\ u_c(t)$ **output**: $\dot{u}_c(t)$
1	$u_0 := 220\sin(2\cdot\pi\cdot 50\cdot t)$
2	$m_1 = s_1 < 0$ $m_2 = s_2 < 0$ $m_3 = s_3 < 0$ $m_4 = s_4 < 0$ $w = m_1/R_1 + (1 - m_1)$ $\begin{bmatrix} 0 & m_2 & 0 & m_4 \\ m_1 & -m_2 & m_3 & -m_4 \\ w & -m_2/R_1 & m_3-1 & 0 \\ m_1-1 & m_2-1 & 1-m_3 & 1-m_4 \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \\ s_3 \\ s_4 \end{bmatrix} = \begin{bmatrix} -u_c \\ 0 \\ u_0/R_1 \\ 0 \end{bmatrix}$
3	$v_1 := -m_2 \cdot s_2$ $v_3 := m_4 \cdot s_4$ $u_R := u_C$ $i_R := u_R/R_2$ $i_4 := (1 - m_4)\cdot s_4$ $i_3 := (1 - m_3)\cdot s_3$ $i_c := i_3 + i_4 + i_R$ $\dot{u}_c := i_c/C$

Thus, the relations have a fixed value during integration (which are the values of the relations from the last event instant). Therefore, m_1, m_2, m_3, m_4, w can be computed directly. The remaining task to solve a system of linear equations in the unknowns s_1, s_2, s_3, s_4 can be performed by standard numerical methods.

If one of the relations changes its value, an event occurs. At the event instant, (2) is a mixed system of 9 equations in 5 real (s_1, s_2, s_3, s_4, w) and 4 boolean (m_1, m_2, m_3, m_4) unknowns, which *cannot* be solved by, e.g., Gaussian elimination, due to the boolean equations. The systems have the following structure:

$$\begin{aligned} \mathbf{m} &:= \mathbf{f}(\text{relation}(\mathbf{z})) \\ \mathbf{A}(\mathbf{m})\,\mathbf{z} &= \mathbf{b}(\mathbf{m}) \end{aligned} \tag{1.6}$$

where $\mathbf{m}$ are the *unknown* variables of type boolean or integer, $\mathbf{z}$ are the *unknown* real valued variables, relation($\mathbf{z}$) characterize relations of the form $z_2 > z_1$ and the functions $\mathbf{f}$ characterize boolean or integer expressions of the form m_2 **and not** $(z_2 > z_1)$. The first set of equations ($\mathbf{m} := \ldots$) states that the boolean and integer unknowns $\mathbf{m}$ can be computed, provided the real-valued variables $\mathbf{z}$ are known. This includes also equations of the following form:

$$\begin{aligned} m_1 &:= z_1 > 2 \\ m_2 &:= m_1 \textbf{ or } z_2 < 0 \end{aligned}$$

where variables m_i can be utilized on the right hand side of the assignment statements provided they are calculated beforehand. The mixed set of equations (1.6) can then be solved in the following way:

repeat
  r := <guess value for relation(**z**) >
  m := **f**(**r**)
  A := **A**(**m**)
  b := **b**(**m**)
  <solve **Az** = **b** for **z**>
until relation(**z**) == **r**

In other words: (1) make an *assumption* about the values of the *relations* in the system of equations. (2) Compute the non-real variables **m**. (3) Compute the real-valued variables **z** by Gaussian elemination (**m** fixed). (4) Compute the relations based on the solution of (2) and (3). If the relation values agree with the assumptions in (1), the iteration is finished and the mixed set of equations is solved. Otherwise, new assumptions on the relations are necessary, and the iteration continues. A useful assumption on the values of the relations is for example to utilize the values computed in the last iteration and start the iteration with the values from the last event instant. By additional algorithmic improvements, the convergence can be enhanced.

In the worst case, an exhaustive search must be performed, which may be time consuming if many relations are involved. For example, in the rectifier circuit an exhaustive search would require to try at every event instant, at most $2^4 = 16$ different combinations of the relations. An alternative is to formulate (1.6) as a mixed integer linear program [28] and to use solvers for such problems, e.g., CPLEX [6].

After a solution of the mixed set of equations is found, the integration is restarted and continues until a new event occurs or the final simulation time is reached.

The technique of parameterized curve descriptions was introduced in [4] and a series of related papers. In [21] it was shown that this approach leads to mixed real/boolean systems of equations and algorithms for their solutions were discussed.

3.2 Non-Unique Solutions

In order that a solution of the mixed set of equations can be computed, the matrix **A** in Eq. (1.6) has to be regular for all trial evaluations. When using ideal switch elements, it often happens that for certain switching structures this condition does not hold. For example, in Fig. 8 two cases are shown for the rectifier circuit of Fig. 6, where **A** is singular.

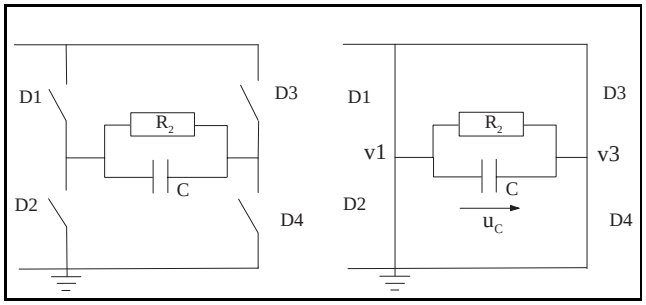

Fig. 8: Configurations with non-unique solutions.

In the left part of Fig. 8, all diodes are off. As a consequence, the load elements (R_2, C) no longer have a connection to ground and therefore one of the potentials of these elements can be selected arbitrarily. In the right part of Fig. 8, all diodes are on. This means that there are two short cuts over diodes D1, D2 and over D3, D4 and that no current flows through the load. It is not possible to tell from this model, in which way the source current is split over the two short cuts. Again, one of the currents (e.g. through D1, D2) is arbitrary and the other one (through D3, D4) can be computed. In reality, every electrical line has a resistance and these resistances determine the splitting of the current in this situation. Since in the model these resistances are neglected, such a non-physical situation occurs.

At the initial time, the rectifier circuit will be in one of the singular situations displayed in Fig. 8, if default initial values are used, because, e.g., all diodes will have the same default initial values when dragged from a library. For example in Modelica, boolean variables have a default value of **false** and therefore variable *off* will be initially **false**, i.e., the rectifier circuit will be in the configuration displayed in the right part of Fig. 8 at initial time.

During continuous simulation the rectifier circuit will be in one of the essential structural modes, explained previously. This means that the solution of the mixed systems of equations at an event instant may fail, because an intermediate switching structure leads to a singular linear system of equations, although the final solution will usually have a regular matrix.

This problem can be solved based on the following idea: (1) If the matrix **A** in Eq. (1.6) is singular, it is checked whether the equations are still consistent, i.e., that an infinite number of solutions exists. (2) From the infinitely many solutions, the one is picked which is "closest" to the solution in the "previous" step. This is a linear least squares problem with singular system matrix **A** that has a unique solution and that can be solved by standard methods, see e.g. [12].

In general it is possible that the simulation continues in a singular configuration, although this cannot occur for the rectifier circuit of Fig. 8. This is uncritical for the configuration in the left of Fig. 8, because at every integration step the linear least squares problem is solved and this gives a continuous solution over time, since **A** and **b** of Eq. 1.6 change continuously. However, for the configuration in the right of Fig. 8, the equation

$$u_C = v_1 - v_3$$

holds and since $v_1 = v_3 = 0$, it follows that $u_C = 0$, i.e., there is a constraint equation for the state u_C. This implies that u_C can no longer be a state. To handle such a situation automatically by a modeling system is a topic of future research.

3.3 Generalizations

From the example, some general conclusions can be drawn: Hybrid systems which arise from *physical abstractions*

should be described in such a way that

1. *equations* are defined for the *different configurations* of a component, and
2. *conditions* specify when a particular configuration is *valid*, i.e., the domain of applicability, and *not* when or how to switch between configurations. The latter is inherent in the definition and is deduced by the solution of mixed sets of equations.

Practically, this means that the configuration dependent behaviour of physical systems is described by if-clauses of the type:

if condition1 **then**
    <equations of domain 1>
else if condition2 **then**
    <equations of domain 2>
    ...
else
    <equations of domain n>
end if

where a domain *condition* is defined by boolean or integer expressions. By replacing every relation in an if-condition by a boolean variable and by adding an assignment statement for this boolean variable to the corresponding relation, a physical system model leads to a description which consists of sets of *differential-*, *algebraic* and *boolean/integer equations* of the form:

$$\mathbf{m} := \mathbf{f}_m(\text{relation}(\dot{\mathbf{x}}, \mathbf{x}, \mathbf{y}, t)) \tag{1.7a}$$

$$\mathbf{0} = \mathbf{f}_y(\dot{\mathbf{x}}, \mathbf{x}, \mathbf{y}, t, \mathbf{m}) \tag{1.7b}$$

where **x**(t) are variables with derivatives $\dot{\mathbf{x}}(t)$ appearing in the equations, **y**(t) are algebraic variables, and **m**(t) are variables of type boolean or integer. During the continuous integration, the relations of the DAE [4] (= relation$(\dot{\mathbf{x}}, \mathbf{x}, \mathbf{y}, t)$) are kept at the value which they had at the last event instant and Eq. (1.7a) is not evaluated[5], i.e., **m** is constant. As a consequence, Eq. (1.7b) is a standard DAE and can be solved by known techniques. Additionally, the relations of the DAE are used as *monitor functions*. When one of the relations changes its value, the switching time is determined and the integration is stopped, i.e., an event is located. At an event instant, the complete mixed set of equations (1.7) is solved by appropriate algorithms[6]. After a solution, i.e., a *consistent configuration*, is found, the integration is restarted.

An *efficient* solution of Eq. (1.7) requires the techniques described in section 2. Especially, BLT partitioning is used to identify algebraic loops. If an algebraic loop contains boolean or integer equations, appropriate solution techniques for mixed systems have to be applied only for this subset at an event instant. In many practical cases, these subsets have the linear system structure (1.6). BLT partitioning reduces the otherwise exponentially increasing number of different configurations: If different algebraic loops of mixed systems of equations are identified, these loops can be solved one after the other. This situation occurs if the physical elements are not directly coupled.

The technique of parametrized curve descriptions can also be applied for the description of, e.g., ideal valves, phase changes, Coulomb friction, and with some simple enhancements also for discontinuities which have a memory, such as an ideal thyristor model. Although, the discussed approach is quite general, there are cases which cannot yet be handled in a satisfactory manner, especially if a structural change introduces constraints between state variables (= variable index systems) or if state variables change discontinuously at an event instant, i.e., if impulses occur. These are topics of ongoing research.

4 Hybrid Systems With Complex Discrete Control Logic

As discussed in the introduction, technical systems comprise physical subsystems as well as computerized monitoring and control systems. While elementary control functions are usually implemented as sampled-data controllers which approximate standard continuous controllers and filters, higher-level functions such as actuator and sensor supervision, redundancy management or recipe-driven control comprise a large amount of logical functions and, at least in a slightly idealized view, operate in an asynchronous, event-driven manner. This means that the controllers react to external events and generate outputs that change continuous variables or switches at possibly any instance of time. In reality, the execution of such control programs also requires a certain amount of time which usually varies due to the actual load of the computer system. In abstract models of discrete systems the execution is assumed to be infinitely fast so that no delays occur.

4.1 Models of Discrete Controllers

In principle, discrete controllers are discrete state-transition systems where the transitions are triggered by external events and may depend on conditions, either that certain continuous variables of the physical system are within certain regions or that other discrete systems are in a specified discrete state, and events which are generated and sent to other systems when the transition is taken. A state transition system always is in one and only one discrete state.

If the control logic is modeled in a modular fashion by several state transition systems, they either perform transitions independently or are synchronized, e.g. by synchroni-

[4] DAE is an abbreviation for Differential Algebraic Equation system.

[5] It could be evaluated, but would always give the same values for **m**, since function relation$(\dot{\mathbf{x}}, \mathbf{x}, \mathbf{y}, t)$ returns the value of the last event instant during continuous integration.

[6] Similarly to the solution of (1.6), the solution of (1.7) can either be determined by trying different values of **m** and solving a sequence of nonlinear systems of equations or by the solution of a MINLP (mixed-integer nonlinear program).

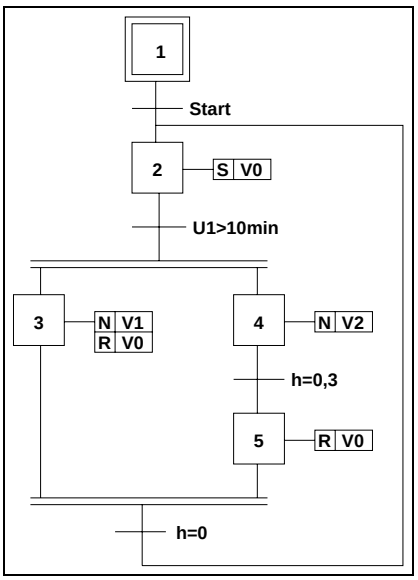

Fig. 9: A simple sequential function chart.

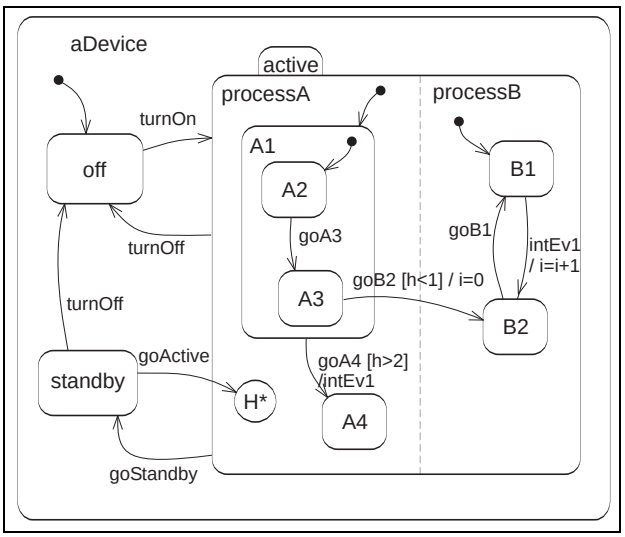

Fig. 10: A statechart.

zation labels. For more complex systems, a representation by simple transition systems very soon becomes unmanageable. Therefore, more powerful modeling formalisms have been developed. In technical applications, two higher-level graphical descriptions are most popular, sequential function charts (SFCs) and statecharts (SCs). SFCs are included in the standard IEC 1131-3 [5] as a tool to structure sequential logical programs. The basic mechanisms are similar to Petri nets, and SFCs are therefore very convenient to express sequential/parallel structures with alternatives and synchronization of branches, e.g. in recipe driven batch process operation [10].

Statecharts were introduced by Harel [11] and have become very popular in the specification of embedded systems, e.g., in cars and airplanes. Statecharts can be regarded as an extension of state transition systems by the introduction of hierarchy (a state can contain substates which either all are active together or are exclusive), concurrency, broadcast communication (events may trigger transitions everywhere in the system) and histories (e.g. after handling an exception state, the system returns to the previous state or resumes in a well-defined intermediate state). They provide a compact and intuitive graphical representation of even very complex state transition diagrams. Many variants of statecharts have been developed and are supported by tools for different purposes [27]. A well-known and rigorously defined variant is that of the commercial design tool Statemate [19].

There are three types of states in a statechart (see Fig. 10): OR-states (aDevice, processA, processB, and A1), AND-states (active) and basic states (all other states). Substates of an AND-state are concurrent states, so that if the statechart is in the state active, it is at the same time in the states processA and processB. Substates of an OR-state are exclusive, consequently if the statechart is in state A1, it is in exactly one of the substates, A2 or A3, as well. The default substate, in which the system goes when it enters an OR-state, is indicated by a default transition. Therefore the statechart assumes the state off when starting and then the states active, processA, processB, A1, A2, and B1 become active, when the event turnOn is registered. The optional parts of the transition labels are: ”triggering events [conditions] / actions“. By means of the history symbol H* the statechart is able to reenter the last active substates in the scope of processA , for example A1 and A3, when it has been in standBy mode and the event goActive occurs.

The main difficulties with statecharts arise from inter-level transitions, which cross the state hierarchy, e.g., the transition from A3 to B2, in combination with the possibility of conflicts with other transitions. The number of possible conflicts is reduced considerably by complex rules of priority, which take the structure of the statechart into account (the transition from A3 to B2 has a higher priority than the transition from A1 to A4, because the first one leaves the concurrent state processA), but eventually some conflict is not handled, so that a nondeterministic situation occurs and simulation has to stop. Alternatively it is possible to enable the user to choose a transition interactively. Such a transition conflict is regarded as a model error that has to be removed. In other approaches [27], inter-level transitions are not allowed because of the complex semantics and the conflicts. However, the expressiveness of statecharts is then reduced significantly. Another important feature is the option to execute a statechart step-by-step or by performing a full internal iteration until no more transitions take place.

4.2 Simulation of Hybrid Systems with Discrete Controllers and Physical Systems

In [18] it was shown for Petri nets how simple state transition diagrams can be described in a declarative way by boolean equations. The advantage is that a corresponding modeling formalism can be directly realized with a few model classes within the object-oriented modeling formalism. A realization was given for the Modelica language.

More complex discrete systems, e.g., systems modeled by statecharts are more conveniently described in an imperative than in a declarative object-oriented fashion. E.g., the priority relation between two interlevel transitions can not be determined with local equations in predefined objects, because it depends on the global structure of the statechart. Also the deep history connector, which can be placed into a state to store the last active sub-states in its scope and allows their reactivation when re-entering the ancestor-state, causes problems, because it requires global information about all sub-states. This approach is also not adequate if several discrete systems interact because it does not reflect the essentially sequential behavior of discrete transition sequences. Therefore, such discrete control formalisms have to be realized separately and have to be interfaced to the object-oriented modeling part of the physical system description.

Real logic controllers can operate in two basic modes, the "PLC-mode" and the sampled-data or synchronous mode. In the PLC-mode, a cyclic operation is realized where the inputs to the discrete system are read, then the control logic is evaluated, then the outputs to the process are updated and then the inputs are read again. Changes of inputs within one execution cycle only influence the controller if they persist until the next start of the cycle. The cycle time varies, depending on the calculations which have to be performed in the cycle under consideration. In the synchronous mode, all process variables are sampled at a fixed sampling rate and the outputs are either set as soon as the computation has terminated or at the next sampling instant.

Since the modeling of the synchronous mode is well-known, below only the PLC-mode will be analysed in more detail. Modeling the PLC-mode behaviour for the case where only one discrete controller is present, leads to the system shown in Fig. 11. In the figure, a physical system model is

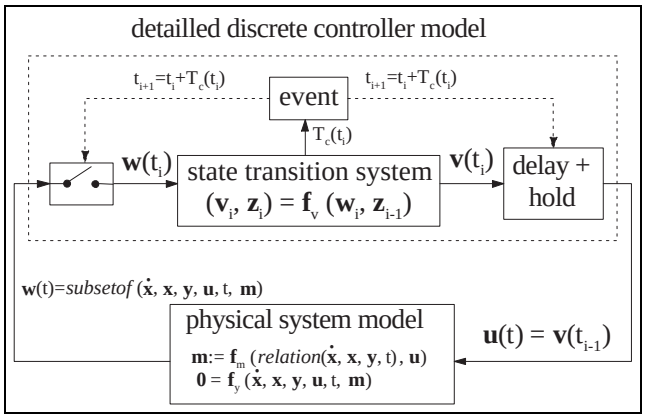

Fig. 11: Physical system and detailled model of discrete controller.

present, which is described essentially by equations of the type (1.7), but has additional (boolean, integer and real valued) input variables $\mathbf{u}$(t) to control the physical system by the discrete controller. At a sampling instant t_i, some or all of the physical system variables are sampled and are utilized in the state transition system of the controller, together with the internal state $\mathbf{z}(t_{i-1})$ of the controller of the previous sample instant t_{i-1} to compute the controller output $\mathbf{v}(t_i)$. The elements of vectors $\mathbf{z}$ and $\mathbf{v}$ maybe of boolean, integer and real type. The calculations within the controller are performed within the variable computation time $T_c(t_i)$ which depends, e.g., on the number of transitions carried out. Therefore, the actual output of the controller is delayed by this time period and the input to the physical system is the value of the controller output from the last event instant: $\mathbf{u}(t) = \mathbf{v}(t_{i-1}), \quad t_i \le t < t_{i+1}$. The computation time may be guessed by the number of performed operations and is used to define the next event instant t_{i+1} in the future.

The equations of the discrete controller (see Fig. 11)

$$(\mathbf{v}_i, \mathbf{z}_i) \;=\; \mathbf{f}_v(\mathbf{w}_i, \mathbf{z}_{i-1}) \tag{1.8}$$

have to be interpreted in such a way, that *several* transitions may be carried out. Depending on the actual implementation, this number may be bounded by an upper limit or transitions may be executed until no transition fires anymore. The return value of $\mathbf{f}_v$ is the value of the internal state $\mathbf{z}_i = \mathbf{z}(t_i)$ and of the controller output $\mathbf{v}_i = \mathbf{v}(t_i)$ *after* carrying out all these transitions.

This detailled model of a discrete controller is uncritical to simulate, because the delay in the calculation of the input to the physical system effectively decouples the discrete and the continuous part. Especially, the discrete controller can never be included in an algebraic loop together with the physical system, because the input $\mathbf{u}$(t) to the physical system is a known quantity (= value of $\mathbf{v}$ from the previous event instant).

In most practical cases, the computation time is small compared to the time constants of the physical processes and can be neglected. In such a case, the controller need only to be evaluated if one of its transitions actually fires, since only then the output of the controller can change. Usually, this leads to a considerable speed-up of the simulation, since the number of events to be processed is reduced. Furthermore, the modeling effort is reduced considerably. The necessary changes to the discrete controller model are shown in Fig. 12. Here, $\mathbf{z}^- = \mathbf{z}(t-\varepsilon)$, $\mathbf{v}^- = \mathbf{v}(t-\varepsilon)$ characterize variable values shortly before the current event is processed and $\mathbf{z}$, $\mathbf{v}$ are the values that are computed at this event instant and that are kept constant until the next event is processed. The new variable $\mathbf{e}$ characterizes a trigger signal, which indicates, e.g., whether a relation has changed its value, and may be utilized in the state transition system. Other changes compared to the detailed controller model of Fig. 11 are: (1) The events which trigger the evaluation of the discrete controller are no longer time events which are determined by the calculation time at the previous event instants, but are state events which are determined by monitoring the relations in the controller which are functions of the variables of the physical system and which potentially may lead to a

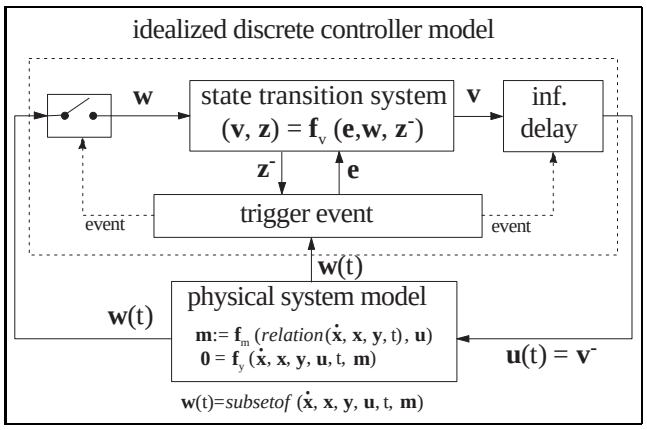

Fig. 12: Physical system and idealized model of discrete controller.

change of the discrete state, e.g., by firing of a transition. (2) The computation time is neglected. In order to still keep the "decoupling property" of discrete controllers and therefore to prevent algebraic loops between the discrete controller and the physical system, an infinitesimally small computation time is modelled by delaying the controller output to the physical system input by one event instant ($\mathbf{u} = \mathbf{v}^-$) and evaluating the physical system equations again whenever $\mathbf{v} \neq \mathbf{v}^-$, without advancing time. Under this premise the overall system evolution is computed in one event iteration loop, which can be described by the following conceptual algorithm:

// initialisation
$(t^-, \mathbf{x}^-, \mathbf{z}^-) := (t_{start}, \mathbf{x}_{start}, \mathbf{z}_{start})$
$\mathbf{v}^- := \mathbf{f}_{v0}(\mathbf{z}^-)$; $\mathbf{u}^- := \mathbf{v}^-$
$(\dot{\mathbf{x}}^-, \mathbf{y}^-, \mathbf{m}^-) :=$ < solution of physical system;
$\mathbf{x}^-, \mathbf{u}^-, t^-$ are known >
$\mathbf{w}^- := \mathit{subsetof}(\dot{\mathbf{x}}^-, \mathbf{x}^-, \mathbf{y}^-, \mathbf{u}^-, t^-, \mathbf{m}^-)$
$\mathbf{c}^- := \mathbf{f}_c(\mathit{relation}(\mathbf{w}^-))$
$\mathbf{u} := \mathbf{v}^-$; $\mathbf{x} := \mathbf{x}^-$; $t := t^-$
repeat // event iteration
$(\dot{\mathbf{x}}, \mathbf{y}, \mathbf{m}) :=$ < solution of physical system;
$\mathbf{x}, \mathbf{u}, t$ are known >
$\mathbf{w} := \mathit{subsetof}(\dot{\mathbf{x}}, \mathbf{x}, \mathbf{y}, \mathbf{u}, t, \mathbf{m})$ // inputs
$\mathbf{c} := \mathbf{f}_c(\mathit{relation}(\mathbf{w}))$ // conditions
$\mathbf{e} := \mathbf{f}_e(\mathbf{c}^-, \mathbf{c}, \mathbf{z}^-)$ // events
$(\mathbf{v}, \mathbf{z}) :=$ **if** $\mathbf{e} \neq \mathbf{0}$ **then** $\mathbf{f}_v(\mathbf{e}, \mathbf{w}, \mathbf{z}^-)$ **else** $(\mathbf{v}^-, \mathbf{z}^-)$
// continuous integration,
// when event iteration converged
if $\mathbf{v} == \mathbf{v}^-$ **then**
< integrate $\mathbf{0} = \mathbf{f}_y(\dot{\mathbf{x}}, \mathbf{x}, \mathbf{y}, \mathbf{u}, t, \mathbf{m})$ until next event >
// copy the values at the event instant
$(\dot{\mathbf{x}}, \mathbf{x}, \mathbf{y}, t) := (\dot{\mathbf{x}}(t_{event}), \mathbf{x}(t_{event}), \mathbf{y}(t_{event}), t_{event})$
end if
// initialisation of next iteration
$t^- := t$; $\mathbf{x}^- := \mathbf{x}$; $\mathbf{v}^- := \mathbf{v}$; ...
until $t == t_{end}$

When a relation has changed its value, the simulation stops and a discrete event iteration is started: First the physical system equations are evaluated, i.e., a mixed system of equations is solved, to compute the unknown variables of the physical system: $\dot{\mathbf{x}}$, $\mathbf{y}$ and $\mathbf{m}$. Thereafter the transition conditions $\mathbf{c}$ that define the thresholds of the controller are calculated depending on the actual inputs $\mathbf{w}$. In order to detect that a condition has become true the triggering events $\mathbf{e}$ are determined by comparing the actual values of the conditions $\mathbf{c}$ with the previous values. If any event has been detected the controller computes the new state $\mathbf{z}$ and the new outputs $\mathbf{v}$. Otherwise, the old values for $\mathbf{z}$ and $\mathbf{v}$ are used. Because the possible transitions depend on the actual state, superfluous evaluations of the state transition function $\mathbf{f}_v$ can be prevented by considering the state $\mathbf{z}^-$ in the computation of the triggering events $\mathbf{e}$. If $\mathbf{v}$ has changed during the last iteration step, the event iteration continues and it is checked if the controller performs another state transition. If not, the event iteration is stopped and the integration of the continuous subsystem is restarted.

The sequential evaluation of the discrete and the switched continuous system does not rule out that this evaluation runs into an infinite loop – switching between a set of different discrete states – while time does not advance. This can be detected by monitoring the sequence of discrete states (if the period is not too large), and the number of discrete iterations should be limited to a reasonable value. Then the simulation must be stopped and the model of the discrete controller must be corrected, e.g., by using a more detailed model taking into account the computing time, i.e., by introducing a finite delay at the output of the controller.

4.3 Simulation of Hybrid Systems with Interacting Logical Controllers

Complex technological systems such as aircraft or chemical plants often contain decentralized logical controllers, that jointly control the physical system part. This may require a coordination of the actions, so that the controllers have to interact with each other. How this interaction has to be considered in the simulation depends strongly on the communication details.

Real PLCs, for example, can communicate in the way that the data exchange is coupled to the execution cycle. This means that the PLC calls the main program at the beginning of a cycle with the actual input values and with the data that it has received during the last cycle from the other PLCs, and sends the data for the other PLCs after finishing the program when setting the new outputs to the physical system. In order to model such a behavior, the data has to be represented by variables that are handled exactly in the same way as the outputs to the physical system. The only difference is that they are provided to other PLCs. This applies to both the detailed and the idealized PLC model. Consequently a hybrid system with interacting PLCs can be modeled simply by inserting the PLC model blocks directly into the systems composition diagram (including the real or an infinitesimally small delay at the output) and connecting them with the physical system parts and the other PLCs. It is evident, that a more complex communication scheme

that has access to inner variables of other logical controllers at any time, increases the modeling effort considerably, because it would be necessary to know at which statement the program is at every point in time.

A different situation arises when several interacting discrete controllers are modeled in an abstract fashion, e.g. by simple automata, statecharts or SFCs. From the point of view of the continuous simulation, interacting discrete systems must be treated as one aggregated discrete block. This overall discrete block performs a sequence of (partially parallel) steps until it has settled into a final state **z** and an output **v** is generated, while the continuous simulation time is stopped. Then the (modular) continuous equations are evaluated. If this leads to a new triggering of the discrete controller the discrete system is evaluated again etc. The precise fashion in which the discrete steps in the communicating discrete controllers are taken in one event iteration is defined by the semantics of the discrete formalisms which are used. It seems not possible, and seems also not to be effective, to try to express this interaction in an equation-based manner, because the behavior is essentially sequential and not synchronous.

Thus the treatment of interacting discrete systems which are modeled by abstract formalisms which assume no execution time of the discrete steps in an object-oriented framework requires a preprocessing of the sets of directly interacting discrete blocks into aggregated discrete systems which can then, e.g., be executed as algorithm sections in Modelica. This is a topic of our ongoing research.

5 Aircraft Redundancy Control Management

Aircraft are safety critical systems, and, therefore, control systems incorporate several forms of redundancy. For example, the elevator control subsystem of the airplane controls its attitude and may consist of two mechanical elevators at the rear of the airplane (see Fig. 1) [14, 25]. Each of these elevators may be controlled by two redundant actuators. The four actuators may be powered by three independent hydraulic systems, see Fig. 13.

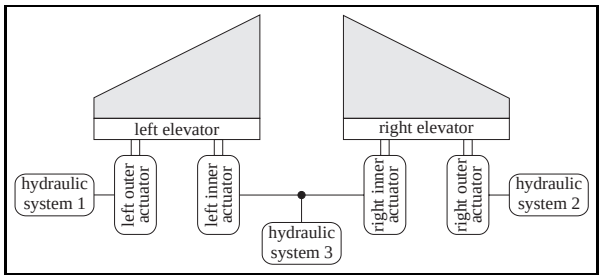

Fig. 13: The elevator control system.

At a given time, each elevator is controlled by one of its actuators only. This actuator is said to be in its *active* module actuator control mode (MACM). The redundant actuator may be in any one of a number of remaining MACMs defined in Table 2.

Table 2: Discrete modes of one actuator.

Mode	Description
Active	The module controls the servo valve in a closed loop. The corresponding actuator is active and controls the elevator movement.
Hot and *Standby*	The module controls the servo valve in a closed loop. The corresponding actuator is not active and operates as a load.
Passive	The module is waiting and does not generate actuator control signals.
Off	The module is turned off temporarily because of an intermittent failure and does not generate actuator control signals.
Isolated	The module is turned off indefinitely.

The behavioral redundancy requirements may be formalized by a set of rules to switch between MACMs and be extended with actuator behavior. This allows for testing system behavior in failure situations by simulation and requires the specifications to be translated in an executable format. To this end, an object oriented model is designed for the continuous dynamics of the system based on the schematic in Fig. 14. This model shows the supply and return of the hydraulics power system that is used to position the elevator by means of a positioning cylinder. The control variable in this system is the position of the servo valve, which controls the amount of oil flow into and out of the cylinder. A spool valve is included in the hydraulics path to switch the actuator on and off. When off, the spool valve allows a flow of oil between the chambers of the cylinders by means of a loading passage way.

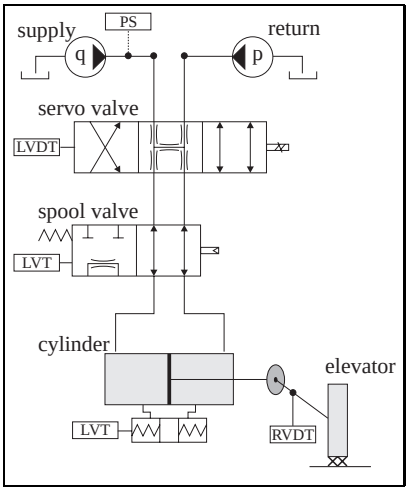

Fig. 14: The hydraulics of an actuator.

This system lends itself well to an object oriented modeling approach. Interaction between components is based on energy exchange and takes place by well definded interfaces. Complex components such as the cylinder can be decomposed into subcomponents at a more detailed level of the hierarchy. The discrete control of an actuator con-

trol module can be modeled by the statechart shown in Fig. 15 [14].

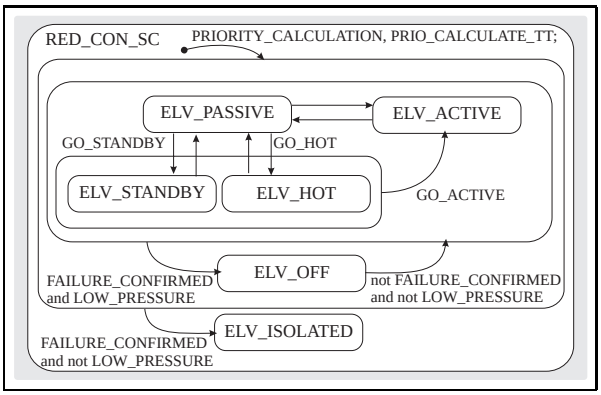

Fig. 15: A statechart to model the discrete control of one actuator on one module.

The statechart redundancy management of the elevator actuators was integrated in a detailed model of the flight characteristics of an aircraft [16]. The physical model of the actuator makes use of the hydraulics library for Dymola [3]. The model includes four statecharts for each of the two primary flight control units. These eight models are connected by logical equations.

6 Summary and Further Aspects

In the preceding sections a method was disussed to model and simulate complex hybrid systems based on appropriate *visual* and *computational* representations of all components of the system. It was shown that it is important to distinguish the sources for the discrete model parts: If discontinuities arise from physical abstractions, e.g., in ideal diode or Coulumb friction elements, a declarative description should be used, such as a complementarity formulation or a parameterized curve description. Using a discrete event formalism, such as a state transition diagram, may lead to difficulties because this introduces an infinitesimal delay that is not present in the physical model and may lead to non-convergent iterations at an event instant.

The continuous part of a hybrid system can be conveniently modeled in a declarative form by composition diagrams. On the other hand, discrete controllers with complex logic are more conveniently described in an imperative fashion and it is preferable to utilize graphical editors dedicated to the specific discrete formalism, e.g., SFC or statechart editors. Since the execution of a control program requires a certain amount of time, there is always a delay between the input and the output of a discrete controller which simplifies the simulation cycle because no algebraic loop can occur between the discrete controller and the physical system. If the computation time is neglected to speed-up the simulation, it is useful to still keep an infinitesimal delay to retain the decoupling property of the real system.

In contrast to other approaches, e.g., the Simulink/Stateflow simulation environment [26], an integrated simulation model has been presented that

- is strictly modular and object-oriented on the continuous level,
- includes a precise event handling and an effective representation of switched continuous systems (abstraction of physical systems),
- includes advanced discrete event modeling formalisms with precise semantics (Stateflow differs considerably from the Statemate semantics, e.g., by defining the execution priority by the vertical position of the blocks on the screen while other tools do not provide high-level modeling formalisms),
- treats the interaction of discrete and continuous systems in a rigorous fashion. The execution of the discrete steps is not coupled to the steps of the integrator of the continuous system as this is done, e.g., in Simulink/Stateflow.

It has been assumed so far that discrete systems can be most conveniently described graphically, since this enhances intuitiveness. However, there may be applications where other approaches are easier to use. For example, if a resource allocation system is to be modelled which has to allocate dozens of resources to dozens of control functions, it seems not to be a good idea to insert a graphical connection for each relation between resources and control functions. The clearest way to handle such systems is to specify the associations in tables and to use the corresponding mathematical relations (on the fundamental sets resources and control functions) in combination with *propositional logic* to specify the desired behaviour in a generic way [23]. This may seem to be a less intuitive representation of the behaviour, but the relations are declared in a clearer form and the behaviour is specified independently of the specific associations. Such a description form is still within the general framework discussed in this article.

In applications where the number of possible combinations of control functions and resources is large, such as recipe-driven batch processes, it makes sense to dynamically reconfigure the state vector during simulation as realized, e.g., in BaSiP [29].

Acknowledgement

We thank Hilding Elmqvist and Johann Bals for helpful discussions. The research reported here was performed in the context of the focused research program (Schwerpunktprogramm) "Continuous-Discrete Dynamic Systems" (KONDISK) and sponsored by the Deutsche Forschungsgemeinschaft under grants OT174/1-2 and EN152/22-2. This support is most gratefully acknowledged.

Literature

[1] M. Andersson. *Omola – An Object-Oriented Language for Model Representation*. Licenciate thesis tfrt-3208, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden, 1990.

[2] P.I. Barton. *The Modelling and Simulation of Combined Discrete/Continuous Processes*. PhD dissertation, University of London, Imperial College, London, 1992.

[3] P. Beater. *hylib — Library of Hydraulic Components for Use with Dymola*. Dynasim AB, Research Park Ideon, Lund, 1997.

[4] C. Clauß, J. Haase, G. Kurth, and P. Schwarz. Extended Amittance Description of Nonlinear n-Poles. *Archiv für Elektronik und Übertragungstechnik / International Journal of Electronics and Communications*, 40:91–97, 1995.

[5] International Electrotechnical Commission. *International Standard IEC 1131 Programmable Controllers, Part 3, Programming Languages*. IEC, Geneva, 1993.

[6] CPLEX. Homepage: http://www.cplex.com/.

[7] Dymola. Homepage: http://www.dynasim.se/.

[8] H. Elmqvist. *A Structured Model Language for Large Continuous Systems*. PhD dissertation, Depart. of Automatic Control, Lund Institute of Technology, Lund, Schweden, 1978.

[9] H. Elmqvist, F.E. Cellier, and M. Otter. Object–Oriented Modeling of Hybrid Systems. In *Proceedings ESS'93, European Simulation Symposium*, pages xxxi–xli, Delft, The Netherlands, Okt. 1993.

[10] S. Engell, M. Fritz, and T. Löhl. Referenzmodelle für Chargenprozesse. In S. Wenzel, editor, *Referenzmodelle für die Simulation in Produktion und Logistik*, pages 170–188. SCS International, Delft, Erlangen usw., 2000.

[11] D. Harel. Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming*, 8:231–274, 1987.

[12] C. Lawson and R. Hanson. *Solving Least Squares Problems*. Prentice Hall, 1995.

[13] P. Lötstedt. Mechanical Systems of Rigid Bodies Subject to Unilateral Constraints. *SIAM J. Appl. Math.*, 42, 1982.

[14] G. Mai and M. Schröder. Simulation of a Flight Control Systems' Redundancy Management System using Statemate. 7. User group meeting STATEMATE, April 1999.

[15] Modelica. Homepage: http://www.modelica.org/.

[16] D. Moormann, P.J. Mosterman, and G.-J. Looye. Object-Oriented Computational Model Building of Aircraft Flight Dynamics and Systems. *Aerospace Science and Technology*, 3:115–126, 1999.

[17] P.J. Mosterman. A Formal Hybrid Modeling Scheme for Handling Discontinuities. In *Proceedings of AAAI-96*, pages 905–990, Portland, USA, Aug. 2-4 1996.

[18] P.J. Mosterman, M. Otter, and H. Elmqvist. Modeling Petri Nets as Local Constraint Equations for Hybrid Systems Using Modelica. In *Proceedings of SCS Summer Simulation Conference*, pages 314–319, Reno, Nevada, July 1998.

[19] A. Naamad and D. Harel. The STATEMATE Semantics of Statecharts. *ACM Transactions on Software Engineering and Methodology*, 5, 1996.

[20] M. Otter. Objektorientierte Modellierung Physikalischer Systeme, Teil 4: Transformationsalgorithmen. *Automatisierungstechnik*, 48:A13–A16, 1999.

[21] M. Otter, H. Elmqvist, and S.-E. Mattsson. Hybrid Modeling in Modelica based on the Synchronous Data Flow Principle. In *CACSD'99*, Hawaii, USA, August 22.–26. 1999.

[22] F. Pfeiffer and C. Glocker. *Multibody Dynamics with Unilateral Contacts*. John Wiley, 1996.

[23] M.A.P. Remelhe and S. Engell. *Structuring Discrete Models in Modelica*. accepted for publication for ADPM 2000 in Dortmund, 2000.

[24] J.M. Schumacher and A.J. van der Schaft. Complementarity Modeling of Hybrid Systems. *IEEE Transactions on Automatic Control*, 43(4):483–490, April 1998.

[25] J. Seebeck. *Modellierung der Redundanzverwaltung von Flugzeugen am Beispiel des ATD durch Petrinetze und Umsetzung der Schaltlogik in C-Code zur Simulationssteuerung*. Diplomarbeit, Arbeitsbereich Flugzeugsystemtechnik, Technische Universität Hamburg-Harburg, 1998.

[26] SIMULINK. Homepage: http://www.mathworks.com/.

[27] M. von der Beeck. A Comparison of Statecharts Variants. In J. Vytopil, W.P.de Roever, and H.W.S Langmaack, editors, *Formal Techniques in Real-Time and Fault-Tolerant Systems*, pages 1–18. Springer Verlag, 1994.

[28] H.P. Williams. *Model Building in Mathematical Programming*. John Wiley & Sons, 4th ed., Berlin, Heidelberg, New York, 1999.

[29] K. Wöllhaf, M. Fritz, Chr. Schulz, and S. Engell. BaSiP - Batch Process Simulation with Dynamically Reconfigured Process Dynamics. *Computers & Chemical Engineering*, 20, 1996.

Manuskripteingang: 27. Juni 2000.

Dr.-Ing. Martin Otter is project manager at DLR, chairman of the Modelica Association and teaches at Tech. Univ. of Munich. Main areas of interest: Control of industrial robots, power train modeling for realtime applications, hybrid systems, Modelica.

Address: Deutsches Zentrum für Luft- und Raumfahrt, Institut für Robotik und Mechatronik, Postfach 1116, D-82230 Weßling.
E-Mail: Martin.Otter@dlr.de

Dipl.-Ing. Manuel A. Pereira Remelhe is a PhD. candidate in the Process Control Laboratory of the Department of Chemical Engineering at the University of Dortmund. Main areas of interest: hybrid systems, discrete control.

Address: Lehrstuhl Anlagensteuerungstechnik, Fachbereich Chemietechnik, Universität Dortmund, D-44221 Dortmund.
E-Mail: M.Remelhe@ct.uni-dortmund.de

Prof. Dr.-Ing. Sebastian Engell is Head of the Process Control Laboratory of the Department of Chemical Engineering at the University of Dortmund. Main areas of interest: controller design, hybrid systems, optimisation of batch processes.

Address: Lehrstuhl Anlagensteuerungstechnik, Fachbereich Chemietechnik, Universität Dortmund, D-44221 Dortmund.
E-Mail: s.engell@ct.uni-dortmund.de

Ir. Pieter J. Mosterman, Ph.D., is a research associate at DLR. Main areas of interest: modeling and simulation of physical systems and computer automated multi-paradigm modeling with applications to control system design, fault detection, isolation, reconfiguration, and computer automated instruction.

Address: Deutsches Zentrum für Luft- und Raumfahrt, Institut für Robotik und Mechatronik, Postfach 1116, D-82230 Weßling.
E-Mail: Pieter.J.Mosterman@dlr.de