Hybrid Modeling in Modelica based on the Synchronous Data Flow Principle

Martin Otter
DLR Oberpfaffenhofen
D-82230 Wessling, Germany
E-mail: Martin.Otter@DLR.de

Hilding Elmqvist
Dynasim AB
Research Park Ideon
SE-22370 Lund, Sweden
E-mail: Elmqvist@Dynasim.se

Sven Erik Mattsson
Dynasim AB
Research Park Ideon
SE-22370 Lund, Sweden
E-mail: SvenErik@Dynasim.se

Abstract

The unique features of the object-oriented modeling language Modelica to model combined continuous time and discrete event systems are discussed. A hybrid Modelica model is described by a set of synchronous differential, algebraic and discrete equations leading to deterministic behaviour and automatic synchronization of the continuous and discrete parts of a model. The consequences of this view are discussed and demonstrated at hand of a new method to model ideal switch elements such as ideal diodes ideal thyristors or friction. At event instants this leads to mixed continuous/discrete systems of equations that have to be solved by appropriate algorithms.

1 Introduction

Modelica^{TM} is 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 modelling practioners. It is a modern language built on *non-causal* modeling with *mathematical equations* and *object-oriented* constructs to facilitate reuse of modeling knowledge in order to support effective library development and model exchange. Modelica is primarily designed to model and to simulate systems consisting of components from different disciplines such as electrical circuits, drive trains, multibody systems, hydraulical and thermodynamical systems. For details about the Modelica project, see http://www.Modelica.org/.

The unique features of Modelica to model *continuous* systems described by differential-algebraic equations (DAEs for short) are discussed in [Elmq98, Elmq99]. Below, an overview of the hybrid features of Modelica are given to model discontinuous and variable structure systems, such as sampled data systems, limiters, ideal diodes, Coulomb friction, backlash and impact.

For modeling of continuous (time) systems, object-oriented modeling languages like Dymola, gPROMS, Modelica and Omola, are based on the same principle: using DAEs to mathematically describe model components. For discrete event systems this is different, because there does not exist a single widely accepted description form. Instead, many formalisms are available, e.g., finite automata, Petri nets, statecharts, sequential function charts, DEVS, logical circuits, difference equations, CSP, process-oriented languages that are all suited for particular application areas.

In Modelica the central property is the usage of *synchronous* differential, algebraic and discrete equations. The idea of using the well-known synchronous data flow principle in the context of hybrid systems was introduced in [Elmq93]. For pure *discrete event* systems, the same principle is utilized in synchronous languages [Halb93] such as SattLine [Elmq92], Lustre [Halb91] and Signal [Gaut94], in order to arrive at save implementations of realtime systems and for verification purposes.

2 Synchronous Equations

A hybrid Modelica model basically consists of differential, algebraic and discrete equations. A typical example is given in figure 1 where a continuous plant

$$\dot{\mathbf{x}}_p = \mathbf{f}(\mathbf{x}_p, \mathbf{u}) \qquad (2.1a)$$
$$\mathbf{y} = \mathbf{g}(\mathbf{x}_p) \qquad (2.1b)$$

is controlled by a digital linear controller

$$\mathbf{x}_c(t_i) = \mathbf{A}\mathbf{x}_c(t_i - T_s) + \mathbf{B}(\mathbf{r}(t_i) - \mathbf{y}(t_i)) \qquad (2.2a)$$
$$\mathbf{u}(t_i) = \mathbf{C}\mathbf{x}_c(t_i - T_s) + \mathbf{D}(\mathbf{r}(t_i) - \mathbf{y}(t_i)) \qquad (2.2b)$$

using a zero-order hold to hold the control variable $\mathbf{u}$ between sample instants (i.e., $\mathbf{u}(t) = \mathbf{u}(t_i)$ for $t_i \le t < t_i + T_s$), where T_s is the sample interval, $\mathbf{x}_p(t)$ is the state vector of the *continuous* plant, $\mathbf{y}(t)$ is the vector of measurement signals, $\mathbf{x}_c(t_i)$ is the state vector of the digital controller and $\mathbf{r}(t_i)$ is the reference input. In Modelica, the complete system can be easily described by connecting appropriate blocks. However, for simplicity of the following discussion, an overall description of the system in one model is used:

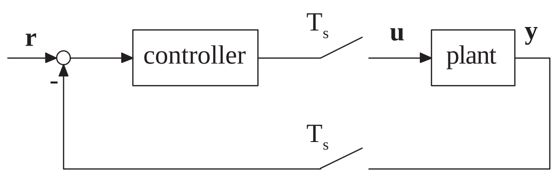


Figure 1: Sampled data system.

```
model SampledSystem
  parameter Real Ts=0.1 "sample interval";
  parameter Real A[:, size(A,1)],
                 B[size(A,1), :],
                 C[:, size(A,2)],
                 D[size(C,1), size(B,2)];
  constant  Integer nx = 5;
  input  Real r [size(B,2)]  "reference";
  output Real y [size(B,2)]  "measurement";
         Real u [size(C,1)]  "control";
         Real xc[size(A,1)]  "disc. state";
         Real xp[nx]         "plant state";
equation
  der(xp) = f(xp, u);     // plant
      y   = g(xp);
  when sample(0,Ts) then  // controller
     xc = A*pre(xc) + B*(r-y);
     u  = C*pre(xc) + D*(r-y);
  end when;
end SampledSystem;
```

This Modelica model consists of the continuous equations of the plant and of the discrete equations of the controller within the **when** clause. Note, that **der**(x) defines the time derivative of x. During continuous integration the equations within the **when** clause are de-activated. When the condition of the **when** clause *becomes* true an event is triggered, the integration is halted and the equations within the **when** clause are active at this event instant. The operator **sample**$(\ldots)$ triggers events at sample instants with sample time T_s and returns **true** at these event instants. At other time instants it returns **false**. Note, that the values of variables are kept until they are explicitly changed. For example, **u** is computed only at sample instants. Still, **u** is available at all time instants and consists of the value calculated at the last event instant.

Within the controller, the discrete states $\mathbf{x}_c$ are needed both at the actual sample instant $\mathbf{x}_c(t_i)$ and at the previous sample instant $\mathbf{x}_c(t_i - T_s)$. The latter value is determined by using the **pre**$(\ldots)$ operator. Formally, the *left limit* $x(t^-)$ of a variable x at a time instant t is characterized by **pre**(x), whereas x itself characterices the *right limit* $x(t^+)$. Since $\mathbf{x}_c$ is only discontinuous at sample instants, the left limit $\mathbf{x}_c(t_i^-)$ at sample instant t_i is identical to the right limit $\mathbf{x}_c(t_i^+ - T_s)$ at the previous sample instant and therefore **pre**$(\mathbf{x}_c)$ characterices this value.

The *synchronous principle* basically states that at every time instant, the *active* equations express *relations* between variables which have to be *fulfilled concurrently*. As a consequence, during continuous integration the equations of the plant have to be fulfilled, whereas at sample instants the equations of the plant and of the digital controller hold *concurrently*. In order to efficiently solve such types of models, all equations are *sorted* by block-lower-triangular partitioning, the standard algorithm of object-oriented modeling for continuous systems (now applied to a mixture of continuous and discrete equations), under the assumption that all equations are active. In other words, the order of the equations is determined by data flow analysis resulting in an automatic synchronization of continuous and discrete equations. For the example above, sorting results in an ordered set of assignment statements:

```
// "known" variables: r, xp, pre(xc)
y := g(xp);
when sample(0,Ts) then
   xc := A*pre(xc) + B*(r-y);
   u  := C*pre(xc) + D*(r-y);
end when;
der(xp) := f(xp, u);
```

Note, that the evaluation order of the equations is correct both when the controller equations are active (at sample instants) and when they are not active.

The synchronous principle has several consequences: First, the evaluation of the discrete equations is performed in zero (simulated) time. In other words, time is abstracted from the computations and communications, see also [Gaut94]. Second, in order that the unknown variables can be *uniquely* computed it is necessary that the number of active equations and the number of unknown variables in the active equations at every time instant are identical. This requirement is violated in the following example:

```
equation // incorrect model fragment!
   when h1 < 3 then
       close = true;
   end when;
   when h2 > 1 then
       close = false;
   end when;
```

If by accident or by purpose the relations h1 $<$ 3 and h2 $>$ 1 become **true** at the same event instant, we have two conflicting equations for `close` and it is not defined which equation should be used. In general, it is not possible to detect by source inspection whether conditions become **true** at the same event instant or not. Therefore, in Modelica the assumption is used that *all equations* in a model may potentially be active at the same time instant during simulation. Due to this assumption, the total number of (continuous and discrete) equations shall be identical to the number of unknown variables. It is possible to rewrite the model above by placing the when clauses in an **algorithm** section and changing the equations into assignment statements:

```
algorithm
   when h1 < 3 then
       close := true;
   end when;
   when h2 > 1 then
       close := false;
   end when;
```

In this case the two **when** clauses are evaluated in the order of appearance and the second one gets higher priority. All assignment statements within the *same* **algorithm** section are treated as a set of n equations, where n is the number of different left hand side variables (e.g., the model fragment above corresponds to one equation). An **algorithm** section is sorted as a whole together with the rest of the system.

Note, that another assignment to `close` somewhere else in the model would still yield an error.

Handling hybrid systems in this way has the advantage that the *synchronization* between the continuous time and discrete event parts is *automatic* and leads to a deterministic behaviour *without conflicts.* Furthermore, some difficult to detect errors of other approaches, such as deadlock, can often be determined during translation already. Note, that some discrete event formalisms, such as finite automata or prioritized Petri nets, can be formulated in Modelica in a component-oriented way, see [Most98].

The disadvantage is that the types of systems which can be modeled is restricted. For example, general Petri nets cannot be described because such systems have non-deterministic behaviour. For some applications another type of view, such as a process oriented type of view or CSP, may be more appropriate or more convenient.

3 Relation triggered events

During continuous integration it is required that the model equations remain continuous and differentiable, since the numerical integration methods are based on this assumption. This requirement is often violated by **if** clauses. For example the simple block of figure 2 with input *u* and output *y* may be described by the following model:

```
model TwoPoint
   parameter Real y0=1;
   input     Real u;
   output    Real y;
equation
   y = if u > 0 then y0 else -y0;
end TwoPoint
```

At point `u=0` this equation is discontinuous, if the if-expression would be taken *literally*. A discontinuity or a non-differentiable point can occur if a relation, such as $x_1 > x_2$ changes its value, because the branch of an if statement may be changed. Such a situation can be handeled in a numerical sound way by detecting the switching point within a prescribed bound, halting the integration, selecting the corresponding new branch, and restarting the integration, i.e., by triggering a *state event*. This technique was developed by Cellier [Cell79]. For details see also [Eich98].

In general, it is not possible to determine by source inspection whether a specific relation will lead to a discontinuity or not. Therefore, by default it is assumed that every relation potentially will introduce a discontinuity or a non-differentiable point in the model. Consequently, relations in Modelica *automatically* trigger state events (or time events for relations depending only on time) at the time instants where their value is changed. This means, e.g., that model `TwoPoint` is treated in a numerical sound way (the **if**-expression $u > 0$ is *not* taken literally but triggers a state event).

In some situations, relations do not introduce discontinuities or non-differentiable points. Even if such points are present, their effect may be small, and it may not affect the integration by just integrating over these points. Finally, there may be situations where a literal evaluation of a relation is required, since otherwise an ”outside domain“ error occurs, such as in the following example, where the argument of function `sqrt` to compute the square root of its argument is not allowed to be negative:

```
y = if u >= 0 then sqrt(u) else 0;
```

This equation will lead to a run time error, because *u* has to become small and negative before the **then**-branch can be changed to the **else**-branch and the square root of a negative real number has no real result value. In such situations, the modeler may explicitly require a *literal* evaluation of a relation by using the operator **noEvent**():

```
y = if noEvent(u>=0) then sqrt(u) else 0;
```

Modelica has a set of additional operators, such as **initial**() and **terminal**() to detect the initial and final call of the model equations, and **reinit**(...) to reinitialize a continuous state with a new value at an event instant. For space reasons, these language elements are not discussed. Instead, in the next section some non-trivial applications of the discussed language elements are explained.

4 Variable structure systems

4.1 Parametrized curve descriptions

If a physical component is modelled detailed enough, there are usually no discontinuities in the system. When neglecting some ”fast“ dynamics, in order to reduce simulation time and identification effort, discontinuities appear in a physical model. As a typical example, in figure 3 a diode is shown, where *i* is the current through the diode and *u* is the voltage drop between the pins of the diode. The diode characteristic is shown in the left part of figure 3. If the detailed switching behaviour is neglectable with regards

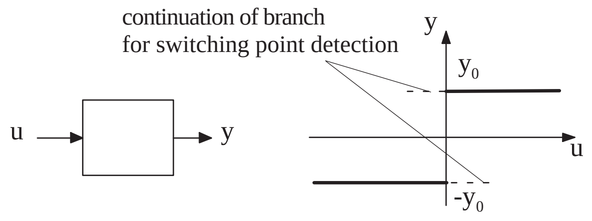


Figure 2: Discontinuous component.

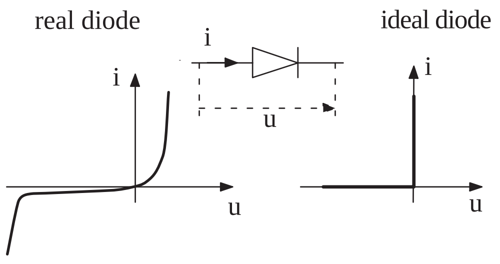


Figure 3: Real and ideal diode characteristic.

to other modeling effects, it is often sufficient to use the ideal diode characteristic shown in the right part of figure 3, which typically give a simulation speedup of 1 to 2 order of magnitudes.

It is straightforward to model the real diode characteristic in the left part of figure 3, because the current i has just to be given as (analytic or tabulated) function of the voltage drop u. It is more difficult to model the ideal diode characteristic in the right part of figure 3, because the current at $u = 0$ is no longer a function of u, i.e., a mathematical description in the form $i = i(u)$ is no longer possible. This problem can be solved by recognizing that a curve can also be described in a parameterized form $i = i(s),\ u = u(s)$ by introducing a curve parameter s. This description form is more general and allows us to describe an ideal diode *uniquely* in a *declarative* way, see first row in table 1.

In order to understand the consequences of parameterized curve descriptions, the ideal diode is used in the simple rectifier circuit of figure 4. Collecting the equations of all components and connections, as well as sorting and simplifying the set of equations under the assumption that the input voltage $v_0(t)$ of the voltage source is a known time function and that the states (here: v_2) are assumed to be known, leads to

$$\begin{array}{rcl}
\text{off} & = & s < 0 \\
u & = & v_1 - v_2 \\
u & = & \textbf{if } \text{off } \textbf{then } s \textbf{ else } 0 \\
i_0 & = & \textbf{if } \text{off } \textbf{then } 0 \textbf{ else } s \\
R_1 \cdot i_0 & = & v_0(t) - v_1 \\
 & & \\
i_2 & := & v_2/R_2 \\
i_1 & := & i_0 - i_2 \\
\frac{dv_2}{dt} & := & i_1/C
\end{array} \qquad (4.3)$$

The first 5 equations are coupled and build a system of equations in the 5 unknowns off$, s, u, v_1, i_0$. The remaining assignment statements are used to compute the state derivative $\dot{v}_2$. During continuous integration the Boolean variables, i.e., off, are fixed and the Boolean equations are not evaluated. In this situation, the first equation is not touched and the next 4 equations form a *linear* system of equations in the 4 unknowns s, u, v_1, i_0 which can be solved by Gaussian elemination. An event occurs if one of the relations (here: $s < 0$) changes its value.

At an *event instant*, the first 5 equations are a mixed system of discrete and continuous equations which cannot be solved by, say, Gaussian elemination, since there are Real and *Boolean* unknowns. However, appropriate algorithms can be constructed: (1) Make an *assumption* about the values of the *relations* in the system of equations. (2) Compute the discrete variables. (3) Compute the continuous variables by Gaussian elemination (discrete variables are 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. Useful assumptions on relation

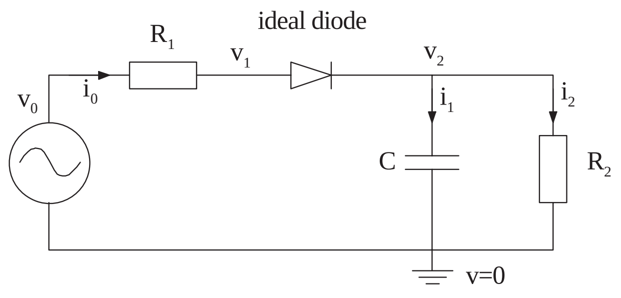
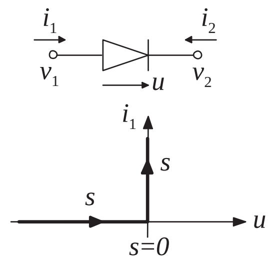
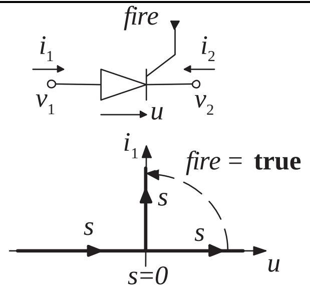
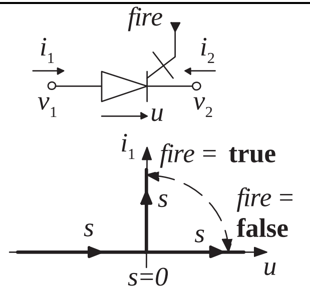


Figure 4: Simple rectifier circuit.

values are for example: (a) Use the relation values computed in the last iteration. (b) Try all possible combinations of the values of the relations systematically (= exhaustive search). In the above example, both approaches can be simply applied, because there are only two possible values ($s < 0$ is **false** or **true**). However, if n switches are coupled, there are n relations and therefore 2^n possible combinations which have to be checked in the worst case.

In table 1 parameterized curve descriptions of the ideal thyristor and the ideal GTO thyristor are shown for further demonstration. Especially note that also non-unique curve parameters s can be used by introducing additional discrete variables (here: *fire*) to distinguish the branches with the

Table 1: Ideal electrical switches

	*ideal diode* $0 = i_1 + i_2$ $u = v_1 - v_2$ $\textit{off} = s < 0$ $u = \textbf{if } \textit{off} \textbf{ then } s \textbf{ else } 0$ $i_1 = \textbf{if } \textit{off} \textbf{ then } 0 \textbf{ else } s$
 fire i1 i2 v1 v2 u fire = true s s s u s=0	*ideal thyristor* $0 = i_1 + i_2$ $u = v_1 - v_2$ $\textit{off} = s < 0$ **or** **pre** $(\textit{off})$ **and not** *fire* $u = \textbf{if } \textit{off} \textbf{ then } s \textbf{ else } 0$ $i_1 = \textbf{if } \textit{off} \textbf{ then } 0 \textbf{ else } s$
 fire i1 i2 v1 v2 u fire = true fire = false s s s u s=0	*ideal GTO-thyristor* $0 = i_1 + i_2$ $u = v_1 - v_2$ $\textit{off} = s < 0$ **or not** *fire* $u = \textbf{if } \textit{off} \textbf{ then } s \textbf{ else } 0$ $i_1 = \textbf{if } \textit{off} \textbf{ then } 0 \textbf{ else } s$

same parameterization values.

The technique of parameterized curve descriptions was introduced in [Clau95] and a series of related papers. However, no proposal was yet given how to actually implement such models in a numerically sound way. In Modelica the (new) solution method follows logically because the equation based system naturally leads to a system of mixed continuous/discrete equations which have to be solved at event instants.

In the past, ideal switching elements have been handeled by (a) using variable structure equations which are controlled by *finite automata* to describe the switching behaviour, see e.g. [Bart92, Elmq93, Most96], or by (b) using a complementarity formulation, see e.g. [Loet82, Pfei96]. (a) has the disadvantage that the continuous part is described in a declarative way but not the part describing the switching behaviour. As a result, e.g., algorithms with better convergence proporties for the determination of a consistent switching structure cannot be used. Furthermore, this involves a global iteration over *all* model equations whereas parameterized curve descriptions lead to local iterations over the equations of the involved elements. (b) seems to be difficult to use in an object-oriented modeling language and seems to be applicable only in special cases (e.g. it seems not possible to describe ideal thyristors).

4.2 Friction

The simulation of components with ideal switch elements becomes difficult, if switching results in an index change of the DAE, i.e., if the number of states is changing. A typical example is Coulomb friction where this situation is present even in the most simple case. To concentrate on the essentials, first the simplified friction element in figure 5 is discussed:

The friction force f acts between two surfaces, see right part of figure 5, and is a linear function of the relative velocity v between the friction surfaces when the surfaces are sliding relative to each other. When the relative velocity becomes zero, the two surfaces are stuck to each other and the friction force is no longer a function of v. The element starts sliding again if the friction force becomes larger than the maximum static friction force f_0. This element can also be described as a parameterized curve, as indicated in figure 5, leading to the following equations:

```
forward  = s >  1;
backward = s < -1;
v=if forward  then  s - 1       else
  if backward then  s + 1       else 0;
f=if forward  then  f0+f1*(s-1) else
  if backward then -f0+f1*(s+1) else f0*s;
```

This model completely describes the simplified friction element in a *declarative* way. Unfortunately, currently we do not know, how to transform such an element description *automatically* in a form which can be simulated. Let us analyse the difficulties by applying this model to the simple block on a rough surface shown in the right part of figure 5

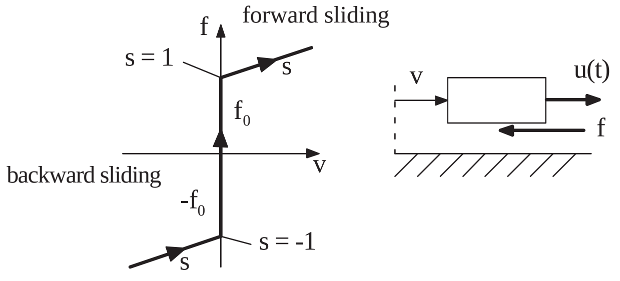


Figure 5: Simplified Coulomb friction element.

which is described by the following equation:

$$m \cdot \dot{v} = u - f \tag{4.4}$$

Note, that m is the mass of the block and $u(t)$ is the given driving force. If the element is in its *forward sliding* mode, i.e., $s \geq 1$, this model is described by

$$\begin{aligned} m \cdot \dot{v} &= u - f \\ v &= s - 1 \\ f &= f_0 + f_1 \cdot (s-1) \end{aligned}$$

which can be easily transformed into state space form with v as the state. If the block becomes stuck, i.e., $-1 \leq s \leq 1$, the equation $v = 0$ becomes active and therefore v can no longer be a state, i.e., an index change takes place. Besides the difficulty to handle the variable state change, there is a more serious problem: Assume that the block is stuck and that s becomes greater than one. Before the event occurs, $s \leq 1$ and $v = 0$; at the event instant $s > 1$ because this relation is the event triggering condition. The element switches into the forward sliding mode where v is a state which is initialized with its last value $v = 0$. Since v is a state, s is computed from v via $s := v + 1$, resulting in $s = 1$, i.e., the relation $s > 1$ becomes **false** and the element switches back into the stuck mode. In other words, it is never possible to switch into the forward sliding mode. Taking numerical errors into account, the situation is even worse.

The key to the solution is the observation that $v = 0$ in the stuck mode and when forward sliding starts, but $\dot{v} > 0$ when sliding starts and $\dot{v} = 0$ in the stuck mode, see figure 6. Since the friction characteristic in figure 6 at zero velocity is no functional relationship, again a parameterized curve description with a new curve parameter s_a has to be used leading to the following equations (note: at zero velocity):

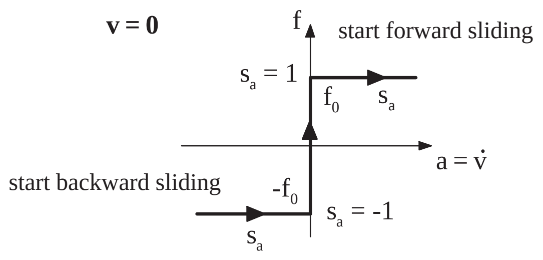


Figure 6: Friction characteristic at $v = 0$.

```
   startFor  = sa >  1;
   startBack = sa < -1;
   a = der(v);
   a = if startFor  then  sa-1 else
       if startBack then  sa+1 else 0;
   f = if startFor  then  f0   else
       if startBack then -f0   else f0*sa;
```

At zero velocity, these equations and the equation of the block (4.4) form again a mixed continuous/discrete set of equations which has to be solved at event instants, similarily as in the simple rectifier circuit discussed above. When switching from sliding to stuck mode, the velocity is small or zero. Since the derivative of the constraint equation $\dot{v} = 0$ is fulfilled in the stuck mode, the velocity remains small even if $v = 0$ is not explicitly taken into account. By this well-know procedure, the velocity v remains a state in all switching configurations.

Consequently, v is small but may have any sign when switching from stuck to sliding mode; if the friction element starts to slide, say in the forward direction, one has to wait until the velocity is really positive, before switching to forward mode (note, that even for exact calculation without numerical errors a ”waiting“ phase is necessary, because $v = 0$ when sliding starts). Since $\dot{v} > 0$, this will occur after a small time period. This ”waiting“ procedure is most easily described by the state machine of figure 7. Collecting all the pieces together, finally results in the following equations of a simple friction element:

```
// part of mixed system of equations
startFor  = pre(mode)==Stuck and sa >  1;
startBack = pre(mode)==Stuck and sa < -1;
a=der(v);
a=if pre(mode)==Forward or startFor then
      sa - 1 else
  if pre(mode)==Backward or startBack then
      sa + 1 else 0;
f=if pre(mode)==Forward or startFor then
      f0 + f1*v else
  if pre(mode)==Backward or startBack then
     -f0 + f1*v else f0*sa;

// state machine to determine configuration
mode=if (pre(mode)==Forward or startFor)
         and v>0 then Forward else
     if (pre(mode)==Backward or startBack)
         and v<0 then Backward else Stuck;
```

Note, that the equations within the mixed system are evaluated based on the value of ”mode“ when the event occured, i.e., on **pre**(mode). After the new sliding or stuck mode is determined by the solution of a mixed set of continuous/discrete equations, the new value of mode is computed by the last equation which is just a direct mapping of the state machine of figure 7.

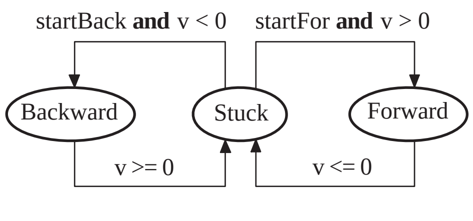


Figure 7: Switching structure of friction element.

The described procedure can be easily applied also for the more general friction element in figure 8 where the sliding friction force has a nonlinear characteristic and there is a jump in the friction force from f_{max} to f_0 when sliding starts. The element equations of the simple friction element

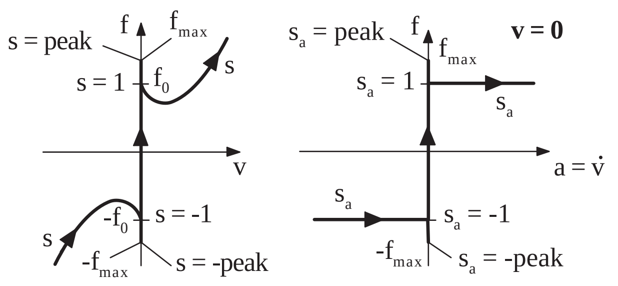


Figure 8: Coulomb friction characteristic.

need only two changes: (1) the linear equation of the sliding friction force has to be replaced by an appropriate nonlinear relationship (usually realized by interpolation in a table) and (2) the sliding conditions have to be modified to:

```
startFor  = pre(mode)==Stuck and (sa > peak
            or pre(startFor) and sa > 1);
startBack = pre(mode)==Stuck and (sa< -peak
            or pre(startBack) and sa < -1);
```

where `peak` = $f_{max}/f_0 \geq 1$. All other equations are identical to the simple friction element. It is straightforward to adapt this general friction element, e.g., to model clutches or brakes.

A simple example of dynamic coupling of friction elements is shown in figure 9 where two blocks are sliding on each other and on every surface friction is present which is described according to the discussed general friction element. By applying appropriate time varying external forces $u_1(t), u_2(t)$ on the two blocks, a stick-slip like behaviour occurs. For this situation, a simulation was carried out with Dymola [Dymo99] leading to the simulation results of figure 10. The discontinuities in the friction forces when switching from stuck to sliding mode are due to a `peak` value of 1.25. The iteration to find a consistent set

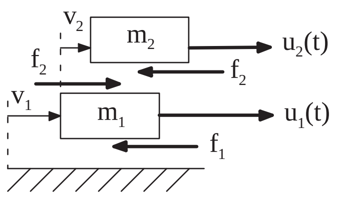


Figure 9: Two blocks with friction.

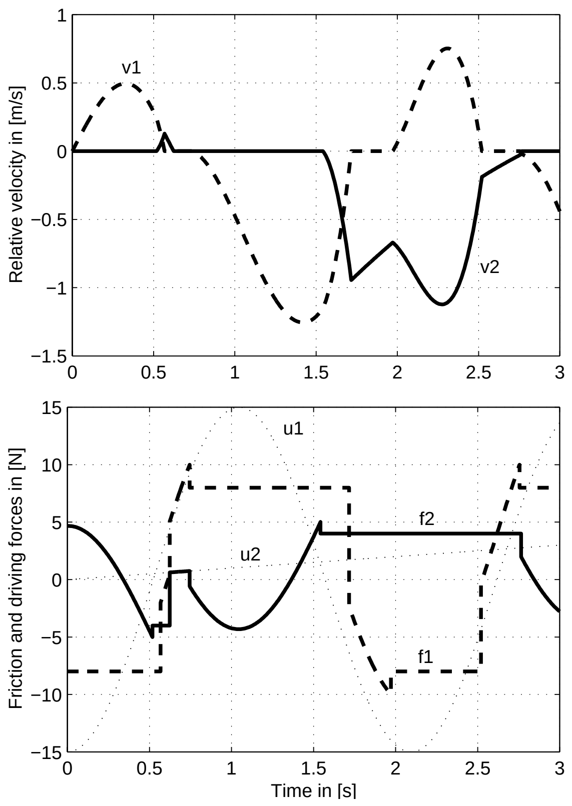


Figure 10: Simulation results of two block system.

of relations and continuous variables can be manually easily checked for demonstration purposes with the following settings ($m_1 = m_2 = 1$; `peak1` = `peak2` = 1):

```
u1 = 0.9 * f0;
u2 = if time<0.1 then 0 else 1.1*f0;
```

At the beginning of the simulation the two blocks are stuck. At time=0.1 s an event occurs and force f_2 jumps from 0 to $1.1 \cdot f_0$. After evaluating all equations under the assumption that both blocks are stuck, it turns out that both friction forces become larger than f_0. Therefore, it is natural to have the assumption that both elements start to slide in forward direction. Re-evaluating leads to an acceleration of block 2 which is negative, i.e., block 2 cannot slide in forward direction. The assumption that block 1 slides and block 2 is stuck, finally leads to a consistent configuration.

5 Conclusion

Modelica is based on synchronous differential, algebraic and discrete equations, leading to a unified mathematical description form of continuous time and discrete event parts of a model. This gives great potential for model analysis and verification of hybrid elements. A typical example is the treatment of ideal switch elements, such as ideal diodes or Coulomb friction, where the Modelica approach together with the technique of parameterized curve descriptions leads to a very promising new method to handle such systems in an efficient and reliable way.

Acknowledgements

The authors would like to thank the other members of the Modelica Design Group for inspiring discussions and their contributions to the Modelica Design.

References

[Bart92] Barton P.I.: ***The Modelling and Simulation of Combined Discrete/Continuous Processes***. Ph.D. Thesis, University of London, Imperial College, 1992

[Cell79] Cellier F.E.: ***Combined Continuous/Discrete System Simulation by Use of Digital Computers: Techniques and Tools.*** Diss ETH No 6483, ETH Zürich, Switzerland, 1979.

[Clau95] Clauß C., 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, pp. 91-97, 1995.

[Dymo99] Dymola. *Homepage: http://www.dynasim.se/.*

[Eich98] Eich-Soellner E., and C. Führer. ***Numerical Methods in Multibody Dynamics***. Teubner, 1998.

[Elmq92] Elmqvist H.: ***An Object and Data-Flow based Visual Language for Process Control.*** ISA/92-Canada Conference & Exhibit, Instrument Society of America, Toronto, April 1992.

[Elmq93] Elmqvist H., F. E. Cellier, and M. Otter: ***Object–Oriented Modeling of Hybrid Systems***. Proceedings ESS'93, European Simulation Symposium, pp. xxxi-xli, Delft, The Netherlands, Oct. 1993.

[Elmq98] Elmqvist H., B. Bachmann, F. Boudaud, J. Broenink, D. Brück, T. Ernst, R. Franke, P. Fritzson, A. Jeandel, P. Grozman, K. Juslin, D. Kagedahl, M. Klose, N. Loubere, S.E. Mattsson, P. Mosterman, H. Nilsson, M. Otter, P. Sahlin, A. Schneider, H. Tummescheit, and H. Vangheluwe: ***Modelica^{TM} – A Unified Object-Oriented Language for Physical Systems Modeling, Version 1.1, 1998***. Modelica homepage: http://www.modelica.org/.

[Elmq99] Elmqvist H., S.E. Mattsson, and M. Otter: ***Modelica – A language for Physical System Modeling, Visualization and Interaction***. Plenary talk, CACSD'99, Hawaii, 1999.

[Gaut94] Gautier T., P. Le Guernic, and O. Maffeis. ***For a New Real-Time Methodology***. Publication Interne No. 870, Institut de Recherche en Informatique et Systemes Aleatoires, Campus de Beaulieu, 35042 Rennes Cedex, France, 1994.

[Halb91] Halbwachs N., P. Caspi, P. Raymond, and D. Pilaud. ***The synchronous data flow programming language LUSTRE***. Proc. of the IEEE, 79(9), pp. 1305–1321, Sept. 1991.

[Halb93] Halbwachs N. ***Synchronous Programming of Reactive Systems***. Kluwer, 1993.

[Loet82] Lötstedt P.: ***Mechanical systems of rigid bodies subject to unilateral constraints***. SIAM J. Appl. Math., Vol. 42, No. 2, pp. 281-296, 1982.

[Most96] Mosterman P. J., and G. Biswas: ***A Formal Hybrid Modeling Scheme for Handling Discontinuities in Physical System Models***. Proceedings of AAAI-96, pp. 905-990, Aug. 2.-4., Portland, OR, 1996.

[Most98] Mosterman P. J., M. Otter, and H. Elmqvist: ***Modeling Petri Nets as Local Constraint Equations for Hybrid Systems using Modelica.*** SCSC'98, Reno, Nevada, 1998.

[Pfei96] Pfeiffer F., and C. Glocker: ***Multibody Dynamics with Unilateral Contacts***. John Wiley, 1996.