Modelica extensions for Multi-Mode DAE Systems

Hilding Elmqvist[1], Sven Erik Mattsson[1], Martin Otter[2]
[1]Dassault Systèmes AB, Ideon Science Park, SE-223 70 Lund, Sweden
[2]German Aerospace Center (DLR), Institute of System Dynamics and Control,
D-82234 Wessling, Germany,
Hilding.Elmqvist@3ds.com, SvenErik.Mattsson@3ds.com, Martin.Otter@dlr.de,

Abstract

This paper describes a proposal for modeling systems with multiple operating modes, such as changing a controller from nominal operation to startup or shutdown or describing failure situations where the model structure is changing (e.g. an electrical line or a mechanical shaft breaks). This is achieved by extending the Modelica 3.3 synchronous state machines to continuous-time state machines having continuous-time models as "states". Every model can be a "state" of a state machine, and in particular certain acausal models. Currently, no new language element is needed for Modelica, but a generalized semantics for state machines has to be introduced, such as "merge semantics for differential equations". Symbolic transformations are still handled during translation, so the generated code is efficient and there is no run-time interpreter. On the other, this feature restricts the class of multi-mode systems that can be handled.

Keywords: Multi-mode, failure simulation, dynamically changing states, continuous-time state machine, hybrid state machine.

1 Introduction

The intention is to extend the scope of Modelica and model and simulate systems with multiple operating modes[1]. Examples:

- Changing a controller from nominal operation to startup, shutdown or manual operation.
- Structural changes of a physical model (e.g. modeling the separation mechanism of a two or three stage rocket).
- Describing failure situations where the model structure is changing (e.g. an electrical line or a mechanical shaft breaks).
- Describing failure situations where the model is completely changing (e.g. the normal behavior of a pump is a 0D model. In case of cavitation, a 1D model is needed to describe physics, requiring to switch dynamically from a 0D to a 1D model when cavitation occurs).

In general this means that the number of continuous-time states of the model might change dynamically during the simulation. Such models cannot be described with current Modelica, version 3.3 (*Modelica Association, 2012*), since in this case the basic requirement is that the number of continuous-time states of a model is fixed during a simulation.

The basic idea for multi-mode modeling is to extend the Modelica 3.3 synchronous state machines (*Elmqvist et. al., 2012*) to continuous-time state machines having continuous-time models as "states". Every model can be a "state" of a state machine, including acausal models, such as a capacitor, or a rotational inertia. The number of continuous-time state variables can change at a transition of a state machine. No new Modelica language element is needed, but a generalized semantics for state machines has to be introduced, such as "merge semantics for differential equations". The concepts have been evaluated with a Dymola prototype (*Dassault Systèmes, 2014*).

A related paper (*Bouissou et. al., 2014*) discusses the use of continuous-time state machines in Modelica for the modeling of stochastic hybrid systems by means of Piecewise Deterministic Markov Processes (PDMP). It focuses on how to handle the case when the transitions on continuous-time state machines depend on stochastic transitions.

Modeling state machines with differential equations in the states is a well-known approach in automata theory, called hybrid automata, see e.g. (*Henzinger 1996*). Such an approach is only of limited use when modeling physical systems, even if the Ordinary Differential Equations (ODEs) on a state are extended to Differential-Algebraic Equations (DAEs): The user has to enumerate all different configurations of a system and has to provide the equa-

[1] Section 1 and 2 of this article are an updated version of the internal report D4.1.2-M12 from the ITEA2 project MODRIO. The rest of this article is new material.

tions for every configuration and also provide all the details to switch between these configurations, see also next section.

MOSILAB (*Bastian et. al., 2010*) from Fraunhofer is an extension to Modelica by introducing continuous-time statecharts in Modelica and supporting DAEs on the states. This approach has similar drawbacks as hybrid automata, but is more powerful as hybrid automata due to the usage of Modelica.

Sol (*Zimmer, 2010*) is an experimental language to model variable structure systems with index changes. The large flexibility requires an interpreter for the simulation: Whenever the DAE index of a system is changed during simulation, the relevant equations of the model are newly symbolically processed and especially also newly index-reduced. The advantage is that a very large class of physical systems with variable structure can be described. The drawback is that an interpreter is needed at run-time, considerably reducing the simulation efficiency.

The approach presented in this paper introduces certain restrictions on what kind of changes can be made to the model at a mode change. This allows symbolic transformations still to be handled during translation, so the generated code is efficient and there is no run-time interpreter.

2 Continuous-time state machines with causal blocks

2.1 Synchronous state machines

In Modelica 3.*3 (Modelica Association, 2012*) synchronous state machines are defined. These state machines are only executed as sampled data systems. From the specification:

Any Modelica block instance without continuous-time equations or algorithms can potentially be a state of a state machine. A cluster of instances which are coupled by transition statements makes a state machine. All parts of a state machine must have the same clock. All transitions leaving one state must have different priorities. One and only one instance in each state machine must be marked as initial by appearing in an initialState statement.

2.2 Continuous-time state machines

In order to define multi-mode systems, the basic idea is to generalize the clocked state-machines from Modelica 3.3 to continuous-time. In a first step, two cases are distinguished:

1. All states and transitions of one state machine are clocked and belong to the same clock (= semantics in Modelica 3.3).
2. All states and transitions of one state machine are continuous-time (= new, additional semantics; discussed below).

In this section we only consider "causal" continuous-time systems, that is, the "states" must be "blocks" with defined "input" and "output" variables. As a result, every "state" block can be separately symbolically processed (such as BLT partitioning) assuming that all its inputs, `pre(..)` and arguments of `der(..)` are known and all other variables, especially outputs, `der(..)`, and arguments of `pre(..)` are unknown. Therefore, from a conceptual point of view a "state" is a set of ordinary differential equations (ODE) with known inputs $\mathbf{u}$ and states $\mathbf{x}$ and of explicit algebraic equations with unknown outputs $\mathbf{y}$ computed in the block:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}, t)$$
$$\mathbf{y} = \mathbf{g}(\mathbf{x}, \mathbf{u}, t)$$

The outgoing transitions of the active state of a state machine are Boolean conditions that are continuously monitored and are transformed to event indicator signals (also called zero-crossing functions) that signal an event when the Boolean condition changes its value. At this time instant the numerical integration is stopped, the state machine switches to the new "state" and the ODE of this new state is either restarted or continues from the previous value of its continuous-time state variables when the "state" was active the last time.

As a simple example consider the continuous-time state machine in *Figure 1* with two "states". This state machine consists of "states" that consist of completely unrelated blocks (in the upper "state" it is a drive train with clutches, and in the lower "state" it is a controlled electrical motor with load). At the start of the simulation the upper state is active and the drive train is simulated. At time = 1.5 s, the simulation of the drive train is stopped and the controlled electrical motor block starts simulating.

As can be seen, the number of continuous-time states changes dynamically during simulation and the state machine switches between unrelated models. There is the restriction that every "state" block needs to have a full initialization definition (e.g., starting from given start values of the continuous-time states, or starting from a steady-state condition which requires to solve a nonlinear algebraic equation system when the block becomes active the first time). When a "state" block is activated the first time, the initialization equations are first evaluated before the simu-

DOI 10.3384/ECP14096183

lation of the block is started. When a "state" block is re-entered, the block is re-initialized if the reset flag is set in the transition. Otherwise, the block continues from the values of the continuous-time variables state of the previous activation.

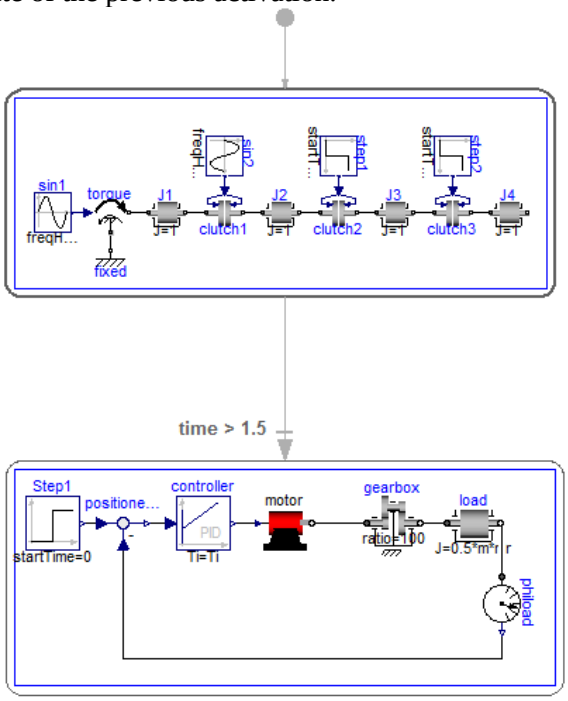


Figure 1: Continuous-time state machine

2.3 Hybrid automata

The system in *Figure 1* is practically not very useful, since signal values are not exchanged between the different "state" blocks. More useful state machines are the already mentioned hybrid automata, see for example (*Henzinger 1996*). A simple example is shown in *Figure 2* and *Figure 3*:

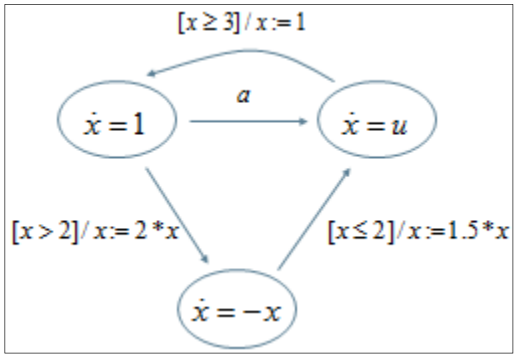


Figure 2: Hybrid automata with "a" as input signal.

On every "state" an ODE is present. The transition conditions consist of trigger conditions when to switch the state (e.g. "$[x \geq 3]$"). Furthermore, at the transition it is defined in which way to reset the state of the ODE (e.g. "`/x:=1`" means that continuous-time state x is reset to 1, before entering the target "state"). In the Dymola prototype, this state

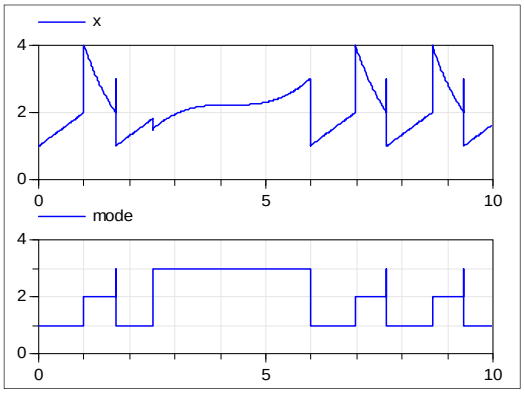


Figure 3: Solution of the hybrid automata from Figure 1 with a = time > 2.5".

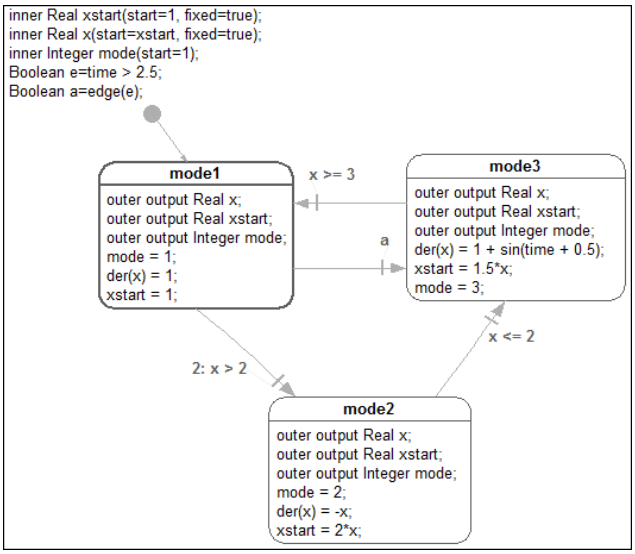


Figure 4: Hybrid automata from Figure 1 modeled in Modelica with indirect reset.

machine can be modeled with the proposed Modelica extension according to *Figure 4*. Here, every "state" is a block that contains the differential equation. Note, the same continuous-time state variable "x" is used in all "states" (called "`mode1`", "`mode2`", "`mode3`" here), because this variable is defined with "`outer output`" in the mode blocks. Additionally, the start value `xstart` of the state is reported via `inner/outer` to all mode blocks. The expected semantics is that whenever entering a "state" and the reset flag in the transition is set, then the differential equation is newly initialized with the actual start value of "x".

The current semantics of the prototype implementation in Dymola is that when an immediate transition is used, the equations are activated once before the reset is made. This explains why `xstart` is modified in the destination state. The use of the construct:

```
inner Real xstart(start=1, fixed=true);
inner Real x(start=xstart, fixed=true);
```

is not standard Modelica since the variability of the start attribute is parametric. To allow any variability of the start expression would be one possible extension to Modelica to solve the re-initialization problem. However, the solution of using a "global" variable xstart is not elegant and is error prone. It would be better to allow such information for re-initialization to be associated with the transition. Further investigations are needed for designing the "right" extension.

Similarly as for clocked synchronous state machines, also for continuous-time state machines the "single assignment rule" holds for every variable. This means that variables in the different "states" must be merged. The merge-semantics for algebraic variables is identical to the merge-semantics of clocked synchronous state machines (see section 17.3.5 of the Modelica 3.3 specification). Variables that appear in the derivative operator, `der(..)`, are merged in the following, equivalent way:

```
der(x) := if activeState(state1) then
            expr1
          elseif activeState(state2) then
            expr2
          elseif
             ...
          else last(der(x))
```

This means that in the example above, the equations for der(x) in the different "state" blocks are merged into the following single statement:

```
der(x) := if     activeState(mode1) then
            1
          elseif activeState(mode2) then
            -x
          elseif activeState(mode3) then
            1 + sin(time+0.5)
          else last(der(x))
```

3 Continuous-time state machines with acausal models

Hybrid automata are of limited use for physical system modeling, because the equations have to be first manually transformed into an input-output block and this is inconvenient and might be non-trivial. Furthermore, the graphical representation in an object diagram might be "not nice" due to input and output connections in a diagram that uses physical, that is acausal, connectors otherwise. Therefore, from a user point of view, it is important to support acausal models as "states". In general this is non-trivial, because different "states" may require different symbolic handling of the equations in the environment since causality and the DAE index might change between "states" of a state machine. However, a quite large class of acausal model "states" have been identified that can be reasonably handled.

As an example, consider the electrical circuit with two state machines in *Figure 6*. In the upper part of the electrical circuit a diode model is present as a "state". At "`time > 0.9`" this state is left and state "`brokenDiode`" is activated consisting of a resistor with a large resistance. Note, that the "states" have "electrical pins" from which they are connected with the rest of the circuit. In the lower right part of the circuit diagram, another state machine is present. It consists of a capacitor "state" (modeled as a resistor in series with an ideal capacitor model).

At time = 0.3s, the hardware configuration is changed and the state switches to state "`R2`" consisting of a small resistance. At time `0.7s`, the configuration is switched back. Note, that in this second case the number of continuous-time states is changing when switching.

In the rest of this section, the new method is described that allows us to handle such systems in an efficient way.

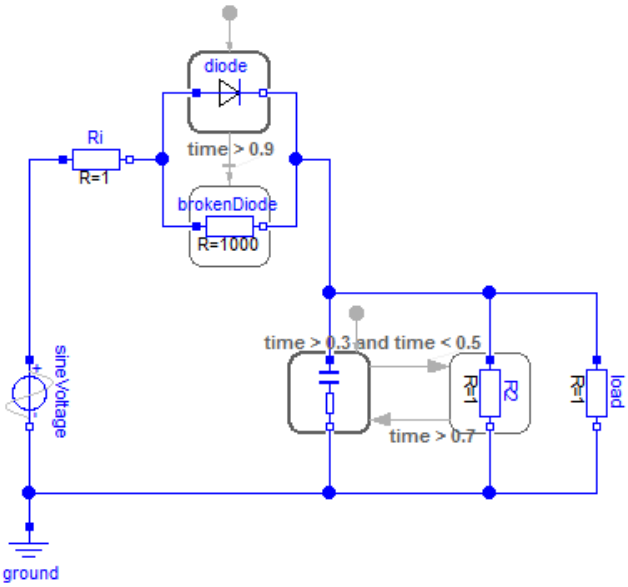


Figure 5: Circuit with two acausal state machines.

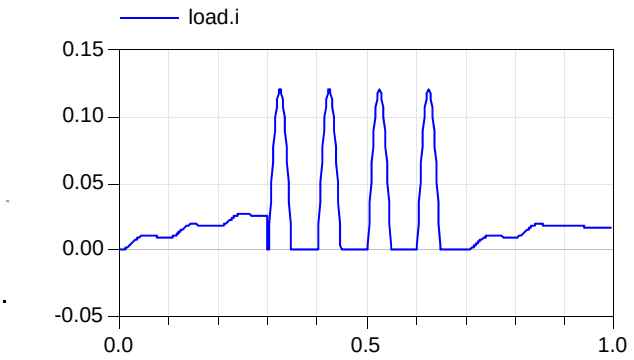


Figure 6: Simulation result of the circuit of Figure 5.

DOI 10.3384/ECP14096183

3.1 Basic idea for symbolic processing

The basic idea to symbolically process continuous-time state machines with physical connectors is explained at hand of the simple example in *Figure 7*. This circuits contains a state machine with two states, capacitor `C` (state 1) and resistor `R2` (state 2). The simulation starts with capacitor `C` and after 0.5 s the state machine is switched to the resistor `R2`. Conceptually, the same behavior can be achieved with the Modelica model shown in *Figure 8* where the models of the two previous two states are connected together with a special switch explained below.

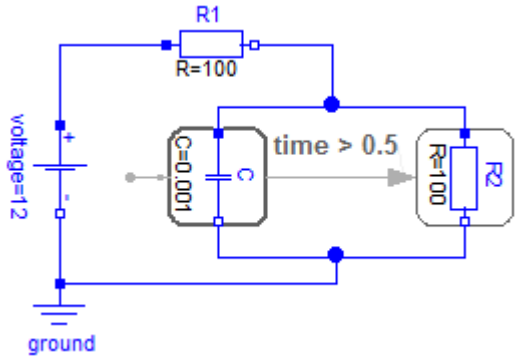


Figure 7: Simple circuit with an acausal state machine.

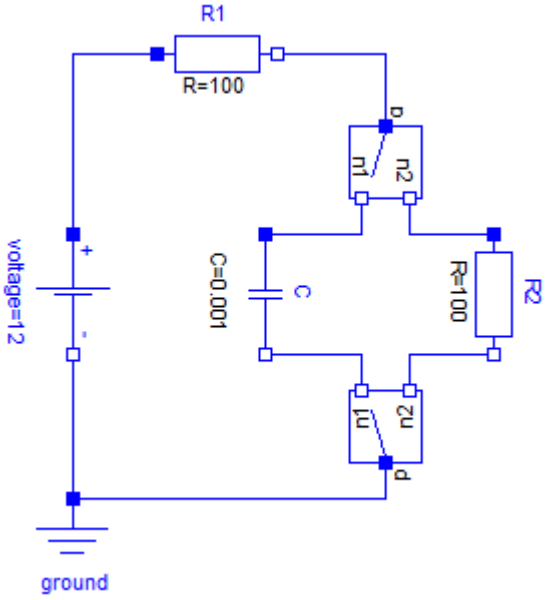


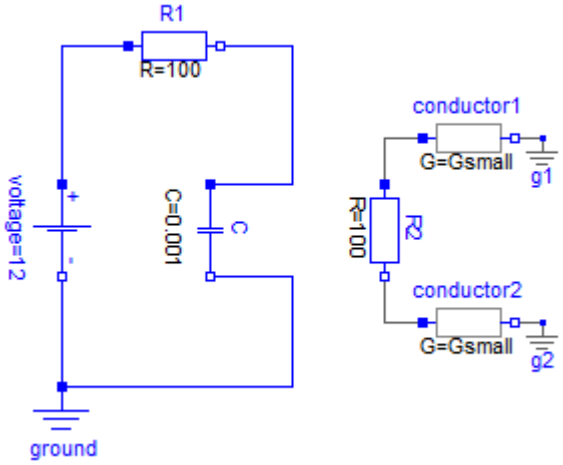
Figure 8: Simple circuit from Figure 7 implemented with special switches using standard Modelica.

Via the Integer input `state` to the switches it can be defined whether pin `n1` or pin `n2` is connected to pin `p`. In the circuit from above, it is defined as:

```
Integer state = if time <= 0.5 then 1
                               else 2;
```

The effect of the two switch positions is demonstrated in *Figure 9*. If in state 1, the capacitor `C` is connected with the rest of the circuit. The resistor `R2` is connected with small (dummy) conductances to ground. This is necessary, because otherwise `R2` would be "floating" and there would be missing equations for voltages. The currents into the pins would be zero since no environment would be attached. However, it is already stated in the resistor that the currents sum to zero. So there would be one equation too much for currents.

Configuration if state = 1



Configuration if state = 2

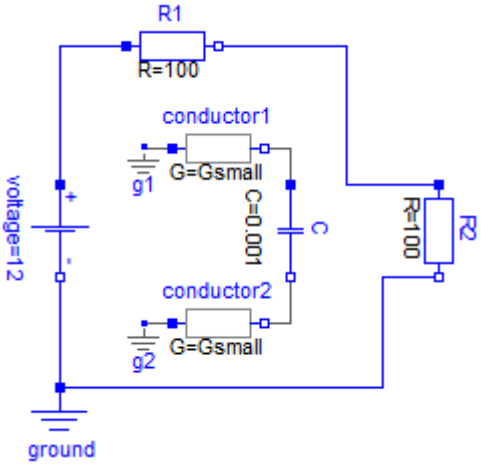


Figure 9: The two configurations of the simple circuit from Figure 8.

If in state 2, the resistor `R2` is connected with the rest of the circuit. The capacitor `C` is connected with small (dummy) conductances to ground, similarly as the resistor in state 1.

It is clear that both configurations can be simulated with a standard Modelica simulator and that the variables of the "active" state will have identical results to the "active" state of the circuit in *Figure 7*. In order to achieve this behavior, the special switch models of *Figure 8* need to be defined as:

```
// Equation for potential variables
p.v = if state==1 then n1.v else n2.v;

// Equation for flow variables
0 = p.i + (if state==1 then n1.i
                       else n2.i);

// Equation for not connected state
0 = if state==1 then Gsmall*n2.v-n2.i
                else Gsmall*n1.v-n1.i;
```

There are different variants to formulate the equations of this switch. The variant above has the advantage that one equation depends only on potential variables, but not on flow variables, and one equation depends only on flow variables and not on potential variables. This is the same principal structure as for every single configuration. For example, assume that in both configurations the potential variable equations need to be differentiated (due to a constraint between states). If in this case the combined potential variable equation of the switch is differentiated, then no flow variables are differentiated because the equation contains only potential variables.

The basic idea to handle acausal state machines can now be sketched:

(1) Connect statements from the outside of a state machine to connectors on the states of a state machine are replaced by the equations of the special switch statement above.

(2) All state machine states are removed and the equations of all states and all transitions are added to the rest of the model. The result is a standard DAE according to current Modelica.

(3) The standard symbolic transformation algorithms are applied on this resulting DAE, such as Pandelides algorithm (*Pantelides 1988*), BLT, dummy derivative method (*Mattsson and Söderlind 1993*).

(4) When generating code, all equations originally belonging to a state are de-activated (are not evaluated), when this state is not active. Furthermore, all continuous-time state variables from all non-active states are removed from the integrator[2].

Note, applying rules (1) and (2) on the example from *Figure 7* results in the circuit of *Figure 8*. Due to rule (4), the code for the dummy conductors will not be executed and therefore the values of the conductances do not matter. The conductors are only important during the symbolic transformation phase where structural properties of the equations are utilized.

Since the symbolic processing might differentiate equations and/or might lead to linear or non-linear systems of equations, this means, that also equations on states might be differentiated or algebraic systems of equations over states and non-states might be present.

In the rest of this paper, the above sketch is more formally defined, consequences and limitations are discussed and several examples are presented.

3.2 Mapping physical connections to equations

In this section it is defined how the `connect(..)` statement of Modelica is mapped to equations if one of the arguments is a connector on a state of a state machine.

3.2.1 Connections between states of one state machine

Requirement 1:
It is not allowed to have a connection set where all connectors of the set belong to the same state machine and at least two connectors are on different states of this state machine.

Note, the states of a state machine are mutually exclusive. If there are connections between mutually exclusive states of a state machine, then the connection will never have an effect (because always only one connector will be active), and there will be missing equations.

3.2.2 Connections between single potential-flow variable pairs

Assume a connector c_i present on a state i is defined by one potential variable p_i and one flow variable f_i and that n of these connectors from the same state machine are connected to connectors outside of this state machine.

A virtual connection node c_0 is (conceptually) introduced outside of the state machine. All connections from c_i to connectors outside of the state machine are replaced by connections to c_0 and connections from c_0 to the original targets. As a result, the following connect statements will be present (the connect statements to the targets are handled according to current Modelica):

$$\texttt{connect}(c_0, c_1)$$
$$\texttt{connect}(c_0, c_2)$$
$$\ldots$$
$$\texttt{connect}(c_0, c_n)$$

[2] A simple way to not integrate over deactivated states is to just report to the integrator that the derivatives of these states are zero, and otherwise keep the dimension of the state vector.

DOI 10.3384/ECP14096183

These connect statements are replaced by the following equations, where i characterizes the active state:

Mapping Equations 1:

$$\begin{aligned}
\boldsymbol{p} &= [p_1, p_2, \cdots, p_n]^T \\
\boldsymbol{f} &= [f_1, f_2, \cdots, f_n]^T \\
i &= \text{activeState}() \\
\\
p_0 &= \boldsymbol{p}[i] \\
0 &= f_0 + \boldsymbol{f}[i] \\
&\textbf{for } j = 1{:}n-1 \\
&\quad k = \text{mod}(i + j - 1, n) + 1 \\
&\quad 0 = G_{small} \cdot \boldsymbol{p}[k] - \boldsymbol{f}[k]
\end{aligned}$$

Note, the for-loop generates $n-1$ dummy equations describing a linear relationship between the potential and flow variables of the connectors on the states that are not active. Due to this mapping, the connect equations have the following structural dependency:

$$\begin{aligned}
0 &= g_1(p_0, \boldsymbol{p}, i) \\
0 &= g_2(f_0, \boldsymbol{f}, i) \\
0 &= \boldsymbol{g}_{3:n+1}(\boldsymbol{p}, \boldsymbol{f}, i)
\end{aligned}$$

These are $n+1$ equations for $n+1$ connectors, where

- one equation depends on all potential variables of the connection set and on the active state,
- one equations depends on all flow variables of the connection set and on the active state,
- $n-1$ equations depend on all potential and all flow variables of the connectors that are on the state machine and on the active state.

The equations $\boldsymbol{g}_{3:n+1}$ are in principal only relevant for the symbolic analysis. During simulation, these equations are never evaluated, because they are deactivated for the active state.

If these equations appear in an algebraic loop, a simple implementation might just compute the solution without taking into account that the equations are deactivated and then ignore the computed value for the deactivated variables. In this case the value of G_{small} matters because the value is used during the solution of the equation system, and if the value becomes too small, the Jacobian of the equation system may become badly conditioned. In this case G_{small} should be in the order of the other elements of the equation Jacobian (or if this information is not known, $G_{small} = 1$ might be used).

An efficient implementation requires to rewrite algebraic equation systems when a new state becomes active. For example, assume a linear system of equations is present during simulation and for state $i = 1$ it has the form:

$$\begin{bmatrix} \boldsymbol{A}_{11} & \boldsymbol{A}_{12} & \boldsymbol{A}_{13} & \boldsymbol{A}_{14} \\ \boldsymbol{0}^T & G_{small} & -1 & \boldsymbol{0}^T \\ \boldsymbol{A}_{31} & \boldsymbol{A}_{32} & \boldsymbol{A}_{33} & \boldsymbol{A}_{34} \end{bmatrix} \cdot \begin{bmatrix} \boldsymbol{z}_1 \\ p_2 \\ f_2 \\ \boldsymbol{z}_3 \end{bmatrix} = \begin{bmatrix} \cdots \\ \cdots \\ \cdots \end{bmatrix}$$

Since the variables and equations of state $i = 2$, especially p_2, f_2, are deactivated (and the variables hold their values) if this state is not active, this linear system of equation can be simplified for state $i = 1$ to:

$$\begin{bmatrix} \boldsymbol{A}_{11} & \boldsymbol{A}_{14} \\ \boldsymbol{A}_{31} & \boldsymbol{A}_{34} \end{bmatrix} \cdot \begin{bmatrix} \boldsymbol{z}_1 \\ \boldsymbol{z}_3 \end{bmatrix} = \begin{bmatrix} \cdots \\ \cdots \end{bmatrix}$$

With this type of equation handling, $\boldsymbol{g}_{3:n+1}$ are never evaluated during simulation and the value for G_{small} does not matter.
Note, when implementing a generic switch for n connectors, the "Mapping Equations 1" can be compactly defined in Modelica:

```
  input Integer state;
  parameter Integer nStates;
  Pin p, n[nStates]
equation
  p.v = n[state].v;
    0 = p.i + n[state].i;
  zeros(nStates-1) = {Gsmall*
     n[mod(state+i-1, nStates)+1].v -
     n[mod(state+i-1, nStates)+1].i
     for i in 1:nStates-1};
```

3.2.3 Connections between input-output connectors

Assume state i of a state machine is an input-output block with an input u_i and an output y_i and that n of these connectors from the same state machine are connected to connectors outside of this state machine – from a node u_0 to the inputs u_i and from the outputs y_i to a node y_0. This means that the following connect statements are present:

$\texttt{connect}(u_0, u_1)$	$\texttt{connect}(y_1, y_0)$
$\texttt{connect}(u_0, u_2)$	$\texttt{connect}(y_2, y_0)$
...	...
$\texttt{connect}(u_0, u_n)$	$\texttt{connect}(y_n, y_0)$

These connect statements are replaced by the following equations, where i characterizes the active state:

Mapping Equations 2:

$$\begin{aligned}
\boldsymbol{u} &= [u_1, u_2, \cdots, u_n]^T \\
\boldsymbol{y} &= [y_1, y_2, \cdots, y_n]^T \\
\\
\boldsymbol{u} &= u_0 \cdot \text{ones}(n) \\
y_0 &= \boldsymbol{y}[\text{activeState}()]
\end{aligned}$$

These are $n+1$ equations for $n+1$ connections. When a state j is not active, an arbitrary value can be provided for its input u_j. For simplicity, just the

overall input u_0 is provided, so $u_j = u_0$. Again, because equations on not active states are deactivated during simulation, these equations are only present during the symbolic transformation, but not during run-time. Therefore, the actually provided value for the input does not matter. Additionally, to the mapping rules above, the standard input-output semantic restrictions (e.g. an output can be only connected to one input), must be relaxed, so that this rule holds only for the active state, but not for deactivated states.

In *Figure 10* an example with two input-output blocks is given: The simulation starts with block `firstOrder`, a first order block. When time > 0.5, the state machine switches to block `secondOrder`. Entering `seondOrder` re-initializes the block to the output of state `firstOrder` (more details on re-initialization are given in section 3.3).

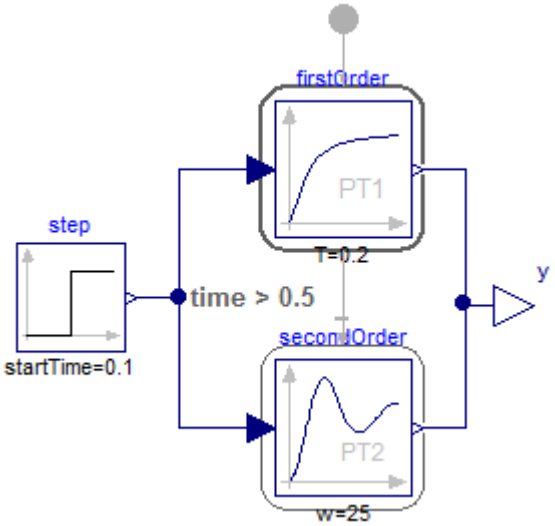


Figure 10: State machine switching between two input-output blocks.

Simulation results are shown in *Figure 11*. Here a first order behavior is seen for the first 0.5 s. Afterwards a second order behavior occurs.

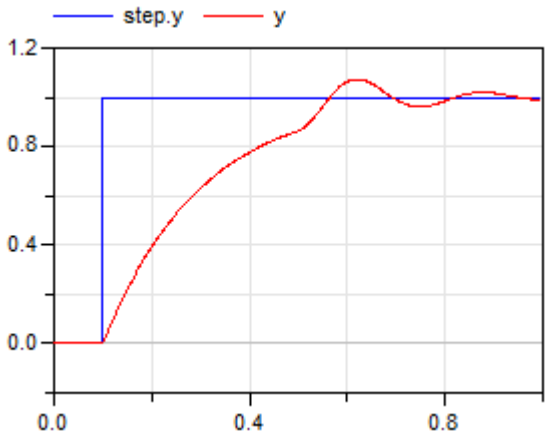


Figure 11: Simulation results of the state machine of Figure 10.

3.2.4 Equation rewriting to enhance efficiency

The approach from section 3.2.2 might lead to algebraic equation systems where the size of the systems is unnecessarily large. In some cases these sizes might be reduced with the following rewriting rules for connect equations:

First, run the Pantelides algorithm (*Pantelides 1988*) and perform the BLT (Block Lower Triangular) transformation on the differentiated problem.

Second, if the following equations of one connection set (see "Mapping Equations 1"),

$$\begin{aligned} p_0 &= \boldsymbol{p}[i] \\ 0 &= G_{small} \cdot \boldsymbol{p}[k] - \boldsymbol{f}[k] \end{aligned}$$

or their differentiated form, appear in an algebraic equation system having all potential variables $\boldsymbol{p}[k]$ as unknowns, but not the flow variables $\boldsymbol{f}[k]$[3], then the equations above can be reformulated to:

$$\begin{aligned} &\textbf{for } i = 1{:}n \\ &\quad \boldsymbol{p}[i] = \textbf{if } \text{activeState}(i) \textbf{ then } p_0 \textbf{ else last}(\boldsymbol{p}[i]) \end{aligned}$$

where $\textbf{last}(\boldsymbol{p}[i])$ is the value of $\boldsymbol{p}[i]$ when state i was active the last time (so it is a known value).

The proof for this rewriting is given for $n = 2$. For $n > 2$, similar arguments can be given. So, assume $n = 2$. Then, the equations from the connect-statements have the following form for the different states, according to "Mapping Equations 1":

$$\begin{aligned} &\textbf{if } \text{activeState}(1) \textbf{ then} \\ &\quad p_0 = p_1 \\ &\quad 0 = f_0 + f_1 \\ &\quad 0 = G_{small} \cdot p_2 - f_2 \\ &\textbf{else} \\ &\quad p_0 = p_2 \\ &\quad 0 = f_0 + f_2 \\ &\quad 0 = G_{small} \cdot p_1 - f_1 \\ &\textbf{end if} \end{aligned}$$

Since it is assumed that the flow variables f_1 and f_2 are known (are computed somewhere else), the dummy conduction equations can be solved for the potential variables (e.g. $p_2 := f_2/G_{small}$). This means that for the non-active states, the potential variables are known. Instead of computing them by a dummy conduction equation, alternatively another known value can be used (because all equations of non-active states are anyway de-activated) and especially $\textbf{last}(\boldsymbol{p}[i])$. As a result, the equations above are equivalent to:

[3] The flow variables do not appear in the algebraic loop if all are continuous-time states or are computed from continuous-time states.

DOI 10.3384/ECP14096183

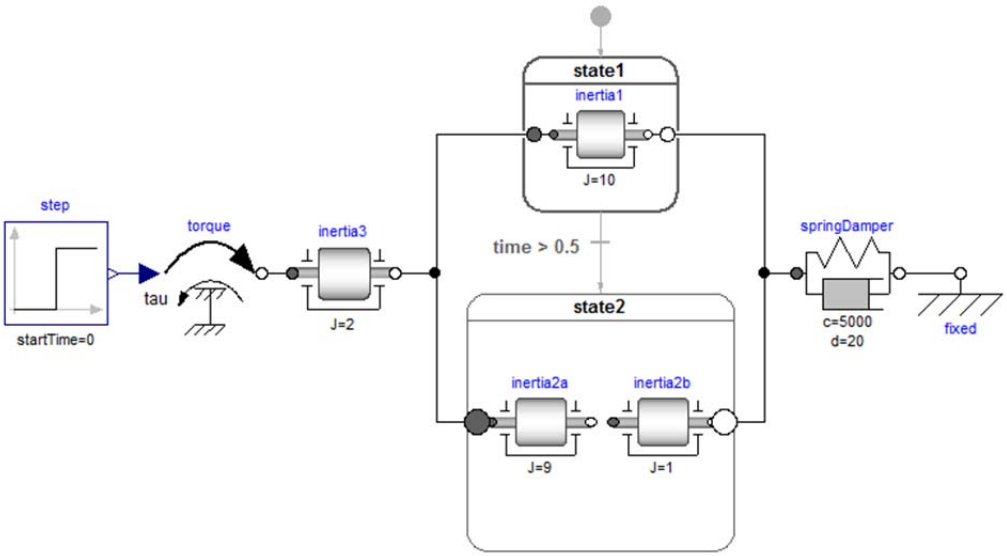


Figure 12: Breaking inertia that requires index reduction and algebraic loop handling over a state machine, as well as re-initialization when entering state2.

$$
\begin{aligned}
&\textbf{if } \text{activeState}(1) \textbf{ then}\\
&\quad p_0 = p_1\\
&\quad 0 = f_0 + f_1\\
&\quad p_2 = \textbf{last}(p_2)\\
&\textbf{else}\\
&\quad p_0 = p_2\\
&\quad 0 = f_0 + f_2\\
&\quad p_1 = \textbf{last}(p_1)\\
&\textbf{end if}
\end{aligned}
$$

This equation set is in turn equivalent to:

$$
\begin{aligned}
p_1 &= \textbf{if } \text{activeState}(1) \textbf{ then } p_0 \textbf{ else last}(p_1)\\
p_2 &= \textbf{if } \text{activeState}(2) \textbf{ then } p_0 \textbf{ else last}(p_2)\\
0 &= f_0 + \textbf{if } \text{activeState}(1) \textbf{ then } f_1 \textbf{ else } f_2
\end{aligned}
$$

and the proof is complete.

After re-writing the equations, all symbolic algorithms are re-run. Usually, the equation systems will become smaller, since the equations depend now only on two variables, and not on many variables. Intuitively this rewriting means that p_0 is computed outside of the state machine and propagated to the states of the state machine. Therefore, the previous algebraic loop over all the states is broken. A prerequisite for this rewriting is that for all non-active states it is sufficient to treat the potential variables as inputs. A sufficient condition for this to be possible is that the flow variables have been already computed somewhere else.

In a similar way, the equations can be rewritten, if the flow variable equation and the dummy conductor equations are in an algebraic loop.

3.3 Re-Initialization

When changing from one state to another one, the DAE of the target state must usually be initialized. This is basically performed with the methods from section 2.3 as demonstrated at hand of the example in *Figure 12*. This example models a drive train, where a rotational inertia breaks during the operation. In particular, this models consists of

- a `state1` with `inertia1` and J=10,
- a `state2` with `inertia2a` (J=9) and `inertia2b` (J=1) that are not connected,
- an `inertia3` outside of the state machine that is connected to `inertia1` and `inertia2a` and is driven by a step torque[4], and
- a spring-damper that is connected to `inertia1` and `inertia2b`.

At time = 0.5, the state machine switches from `state1` to `state2` and therefore `inertia1` is replaced by two unconnected inertias that have together the same moment of inertia as `inertia1`. In other words, the "breaking" of `inertia1` is modelled. Note, that the number of continuous-time states is changing (there are 2 continuous-time state when in `state1` and 4 when in `state2`).

With the generation of connect equations in section 3.2 and the sketched symbolic processing of the overall DAE, this system gives rise to index reduction between `inertia3`, `inertia1` and `inertia2a`

[4] `inertia3` is only present to demonstrate index reduction and algebraic loops over a state machine

and due to this index reduction an algebraic loop between equations of these components are present. The Dymola prototype handles this correctly.

When switching from `state1` to `state2`, `inertia1` has some angle and angular velocity and the inertias on `state2` need to be appropriately initialized. Since `inertia2a` is rigidly attached to `inertia3`, no initialization of `inertia2a` is needed. However, the continuous-time states of `inertia2b` need to be initialized to the states of `inertia3`. This is performed in the following way:

```
inner Real phi(start=0, fixed=true) =
                          inertia3.phi;
inner Real w(start=0,fixed=true) =
                          inertia3.w;
...
// in state2
import R=Modelica.Mechanics.Rotational;
outer Real phi;
outer Real w;
R.Components.Inertia inertia2b(J=1,
          phi(start=phi, fixed=true),
            w(start=w  , fixed=true));
```

On the top level the inner variables `phi` and `w` are associated with the corresponding variables of `inertia3`. In `state2` these variables are declared as `outer` and used as `start` values for `inertia2b`. The semantics is that when `state2` is entered, then the variables of `state2` are re-initialized to their start values. A result plot of the angular velocities of the inertias on the two states is shown in *Figure 13*.

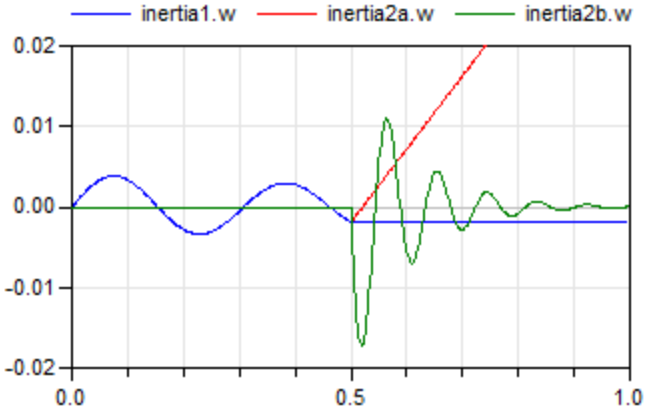


Figure 13: Simulation result for the breaking inertia of Figure 12.

As can be seen, the amplitude and frequency of `inertia2b` increases with respect to `inertia1`, because its moment of inertia is much smaller.

3.4 Limitations

The question arises which types of models cannot be handled with the proposed approach? First, independently of the symbolic algorithms used, the mapping of connect statements to equations as defined in section 3.2 is always correct. It is clear that there must be limitations when applying the standard symbolic algorithms on the resulting set of equations, because the structure of the equations depends on the active state and this dependency is not taken into account by the standard algorithms. For example assume that the following equation is present

```
p.v = if state==1 then n1.v else n2.v
```

when mapping some connect statements to equations. Assume that `p.v` and `n1.v` are states, but `n2.v` is not. Then this state constraint is not detected and `n2.v` is always computed from `p.v` and from `n1`. Of course, this will fail (will give a division by zero) in state 1. The correct handling would be that in state 1 the equation `p.v = n1.v` is present and if both variables are states, this equation must be differentiated. However, the standard algorithms do not take this into account and it seems also non-trivial to generalize.

In *Figure 14* there is a state machine with a capacitor `C1` and a resistor `R2` . These two states are connected in parallel to a capacitor `C2`. This model cannot be handled with the proposed method (and will give a run-time error that a matrix is singular), because in the capacitor state there is a state variable constraint between `C1` and `C2` and in the resistor state there is no such state constraint.

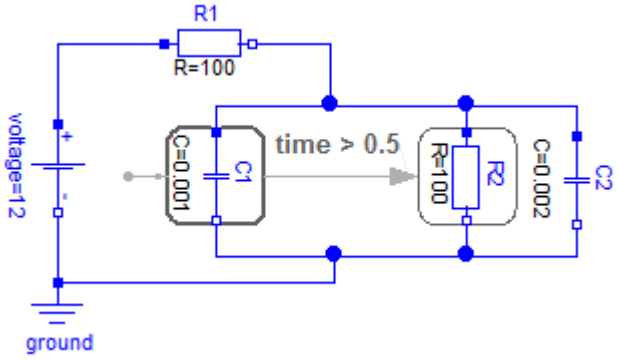


Figure 14: Parallel capacitors that cannot be handled due to different state constraints in the different states.

In *Figure 15* there is a state machine with a connection of two flanges in `state1` and no connection in `state2` (this is defined by connecting zero-torques to the two flanges). This state machine is placed between two inertias. The model describes a breaking inertia in a more natural formulation as in *Figure 12*. This model cannot be handled with the proposed method (and will give a run-time error that a matrix is singular), because in `state1` there is a state constraint between `inertia1` and `inertia2` and in `state2` there is no such state constraint.

DOI 10.3384/ECP14096183

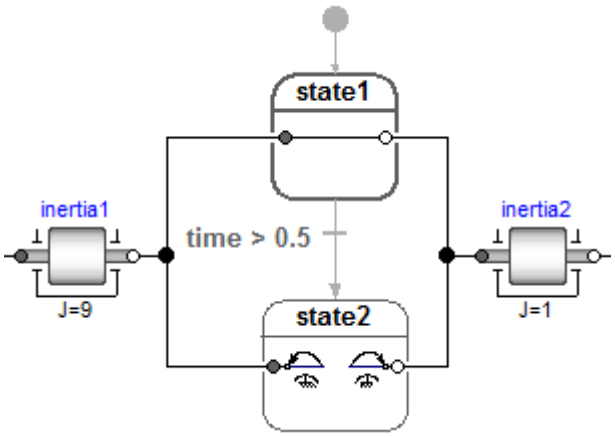


Figure 15: Breaking inertia that cannot be handled due to different state constraints in the different states.

It is not nice that such models lead to an error only when simulating the model. It is also difficult to deduce the source of the problem from an error message stating that a matrix is singular. In principal, the diagnostics can be improved, so that such errors occur during translation with an understandable error message:

The sorted equations are inspected and every algebraic equation system that depends on a state of a state machine is processed again: The equations of an algebraic equation system are partitioned (so the assignment algorithm is applied) for every state machine state taking into account the equation structure of the particular state only. If one of the assignments fails, the equation system is structurally singular for the selected state and therefore the model cannot be handled.

4 Conclusions

A proposal is presented for modeling variable structure systems with dynamically changing number of states in Modelica by extending the synchronous clocked state machines to continuous-time state machines. With this extension it is straightforward to model hybrid automata. However, hybrid automata are not practical to use for physical system modeling. A novel extension is proposed to use acausal models as states of a state machine. By mapping connections to connectors on a state machine in a particular way on equations, the standard symbolic processing for Modelica models can be applied. This approach allows already handling a large class of useful variable structure systems with dynamically changing sizes of continuous-time states.

Models cannot be handled with this new method, if connections between state and non-state components lead to constraints on continuous-time state variables that vary for the different state machine states.

The proposal is not yet complete. Especially, mappings for all connector types of Modelica need to be still defined, especially for multi-body and for fluid systems. Additionally, the switching between DAEs may lead to Dirac impulses, if not properly re-initialized (or it must be modelled in a way that impulses occur, due to the underlying approximation of the reality). Furthermore, algebraic equation systems over states need to be analyzed in more detail, especially in combination with the dummy derivative method. It is planned to work on these topics in the near future.

Acknowledgements

This paper is based on research performed within the ITEA2 project MODRIO. Partial financial support of the Swedish VINNOVA and the German BMBF for this development are highly appreciated.

References

Bastian J., Clauß C., Enge-Rosenblatt O., and Schneider P. (2010): **MOSILAB – a Modelica solver for multi-physics problems with structural variability**. 1st Conference on Multiphysics Simulation - Advanced Methods for Industrial Engineering 2010. Download: http://publica.fraunhofer.de/starweb/servlet.starweb?path=urn.web&search=urn:nbn:de:0011-n-1355711

Dassault Systèmes (2014): **Dymola 2015 Alpha**. http://www.Dymola.com

Elmqvist H., Gaucher F., Mattsson S.E., Dupont F. (2012): **State Machines in Modelica**. Modelica'2012 Conference, Munich, Germany, Sept. 3-5, 2012. Download: http://www.ep.liu.se/ecp/076/003/ecp12076003.pdf

Bouissou M., Elmqvist H., Otter M., and Benveniste A. (2014): **Efficient Monte Carlo simulation of stochastic hybrid systems**. Modelica'2014 Conference, Lund, Sweden, March 10-12.

Henzinger T.A. (1996): **The Theory of Hybrid Automata**. Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science (LICS 96), pp. 278-292.

Mattsson, S.E. and G. Söderlind (1993): **Index reduction in differential-algebraic equations using dummy derivatives**. SIAM Journal of Scientific and Statistical Computing, Vol. 14 pp. 677-692.

Modelica Association (2012): **The Modelica Language Specification, Version 3.3**. Download: https://www.modelica.org/documents/ModelicaSpec33.pdf.

Pantelides C. (1988): **The consistent initialization of differential-algebraic systems**. SIAM Journal of Scientific and Statistical Computing, 9(2), pp. 213–231.

Zimmer D. (2010): **Equation-Based Modeling of Variable-Structure Systems**. Dissertation, ETH Zürich, No. 18924. Download: http://www.inf.ethz.ch/personal/fcellier/PhD/zimmer_phd.pdf