Synchronous Events in the OpenModelica Compiler with a Petri Net Library Application

Willi Braun[1] Bernhard Bachmann[1] Sabrina Proß[1]

[1]Department of Applied Mathematics, University of Applied Sciences Bielefeld, Germany,
{wbraun,bernhard.bachmann,spross}@fh-bielefeld.de

Abstract

In this work an approach is presented that extends the OpenModelica Compiler (OMC) with an event handling module and controls events separately from the integrator. The aim of this extension is to improve the event handling controller of the OMC to handle all equations synchronously, resulting in an efficient simulation of hybrid dynamical systems. This improvements of the event handling allows to formulate the Petri Net library in optimal Modelica code.

Keywords Modelica, Hybrid Models, Petri Nets, OpenModelica, Synchronous

1. Introduction

In general, Modelica models are represented mathematically through differential-algebraic equations (DAEs). A special feature of the Modelica language is the ability to describe continuous and discrete processes in a so-called hybrid model. Hybrid models consist of continuous differential and algebraic as well as discrete equations. The latter introduces events during simulation.

Typical applications for hybrid models are electronic circuits or models with collisions of bodies. These models generate events which can change the behaviour of the system. Furthermore, there are also approaches within event-based modelling, e.g. with hybrid Petri Nets, which involve discrete and continuous places and transitions as well as stochastic transitions.

For the numerical simulation of hybrid systems, a special treatment of events is therefore needed. In addition to a robust numerical integration of the DAEs, the instant of time in which events modify the system should be approximately determined and events should be treated in the correct chronological sequence as they appear. Modelica requires that all equations are handled synchronously at all points in time, even more at events.

In OpenModelica a version of the DASSL algorithm with associated root finding (DASRT) is used, so that the event handling can not be considered independently from the solver [5]. The Petri Net Library [8] could not been expressed in optimal Modelica code since at that time the OMC did not treat discrete events synchronously. This paper describes the correct treatment of the event handling which has been implemented in the OMC, leading to an enhanced formulation of the Petri Net library.

2. Modelica Synchronous Data-flow Principle

A hybrid Modelica model consists of differential, algebraic and discrete equations. The discrete equations are not permanently active, they are only activated when an event occurs. It is important to keep all variables synchronously at all time points.

In order to solve Modelica models efficiently, all equations are sorted into the block-lower-triangular form. The Modelica synchronous principle states that at every time instant, the active equations express relations between variables which have to be fulfilled concurrently (cf. [7]). Based on this synchronous data-flow principle, all equations are considered active during sorting to ensure a correct order at all points of time. The idea of using the synchronous data flow principle in the context of hybrid systems was introduced in [4].

The following Modelica example illustrates how all equations are kept synchronously.

```
when y1 > 2 then
  y2 = f1(y3);
end when;
y3 = f3(y4);
when y0 > 0 then
  y4 = f2(u);
  y1 = f4(y3);
end when;
y0 = f5(u);
```

The example consists of five equations. The three `when`-equations are only active at the points of time at which

3rd International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools. October, 2010, Oslo, Norway.

 Proceedings available at:
http://www.ep.liu.se/ecp/047/
EOOLT 2010 website:
http://www.eoolt.org/2010/

the conditions are even fulfilled. The other two equations describe the continuous behavior. The order of evaluation plays an important role to ensure the synchronous principle.

The individual equations are sorted according to the contained variables. So it must be assumed that all equations are activated simultaneously during transformation into a block-lower-triangular form. Then the right evaluation order can be determined automatically. To sort the `when`-equations correctly, it has to be noted that they also depend on variables that occur in the condition of a `when`-expression.

If the block-lower-triangular transformation is performed on these principles, the evaluation results in the following specified order.

```
//known Variable: u
  y0 = f5(u);
  when y0 > 0 then
    y4 = f2(u);
  end when;
  y3 = f3(y4);
  when y0 > 0 then
    y1 = f4(y3);
  end when;
  when y1 > 2 then
    y2 = f1(y3);
  end when;
```

During continuous integration and also at events the sorting order is always correct because the discrete variables are kept constant during the continuous integration [6].

3. Hybrid Modelica Model represented as DAEs

Flat hybrid DAEs could represent continuous-time behavior and discrete-time behavior. This is done mathematically by the equation (1).

$$F(\underline{\dot{x}}(t), \underline{x}(t), \underline{u}(t), \underline{y}(t), \underline{q}(t_e), \underline{q_{pre}}(t_e), \underline{c}(t_e), \underline{p}, t) = 0 \tag{1}$$

This implicit equation (1) is transformed to the explicit representation of equation (2) by block-lower-triangular transformation.

$$\begin{pmatrix} \underline{\dot{x}}(t) \\ \underline{y}(t) \\ \underline{q}(t_e) \end{pmatrix} = \begin{pmatrix} \underline{f_s}(\underline{x}(t), \underline{u}(t), \underline{p}, \underline{q_{pre}}(t_e), \underline{c}(t_e), t) \\ \underline{f_a}(\underline{x}(t), \underline{u}(t), \underline{p}, \underline{q_{pre}}(t_e), \underline{c}(t_e), t) \\ \underline{f_q}(\underline{x}(t), \underline{u}(t), \underline{p}, \underline{q_{pre}}(t_e), \underline{c}(t_e), t) \end{pmatrix} \tag{2}$$

From this explicit form all necessary calculations can be educed for the simulation of hybrid models. This is done by formulating the continuous-time part, followed by the discrete-time part. Below are summarized the notation used in the following equations:

- $\underline{\dot{x}}(t)$, the differentiated vector of state variables of the model.
- $\underline{x}(t)$, the vector of state variables of the model, i.e., variables of type `Real` that also appear differentiated, meaning that `der()` is applied to them somewhere in the model.
- $\underline{u}(t)$, a vector of input variables, i.e., not dependent on other variables, of type `Real`. They also belong to the set of algebraic variables since they do not appear differentiated.
- $\underline{y}(t)$, a vector of Modelica variables of type `Real` which do not fall into any other category.
- $q(t_e)$, a vector of discrete-time Modelica variables of type discrete `Real`, `Boolean`, `Integer` or `String`. These variables change their value only at event instants, i.e., at points t_e in time.
- $q_{pre}(t_e)$, the values of q immediately before the current event occurred, i.e., at time t_e.
- $c(t_e)$, a vector containing all `Boolean` condition expressions evaluated at the most recent event at time t_e. This includes conditions from all `if`-equations and `if`-statements and `if`-expressions from the original model as well as those generated during the conversion of `when`-equations and `when`-statements.
- $p = p1, p2, \ldots$, a vector containing the Modelica variables declared as parameter or constant i.e., variables without any time dependency.
- t, the Modelica variable time, the independent variable of type `Real` implicitly occurring in all Modelica models.

3.1 Continuous Behavior

The continuous behavior of hybrid DAEs can be formulated with the following equations (3).

$$\begin{pmatrix} \underline{\dot{x}}(t) \\ \underline{y}(t) \end{pmatrix} = \begin{pmatrix} \underline{f_s}(\underline{x}(t), \underline{u}(t), \underline{q_{pre}}(t_e), \underline{c}(t_e), \underline{p}, t) \\ \underline{f_a}(\underline{x}(t), \underline{u}(t), \underline{q_{pre}}(t_e), \underline{c}(t_e), \underline{p}, t) \end{pmatrix} \tag{3}$$

The states $\underline{x}(t)$ are determined by an integration method, so that they are assumed to be known as the vectors $\underline{u}(t)$ and $\underline{p}$. For discrete variables and the condition expressions t_e is used instead of t to indicate that such variables may only change values at event points of time and are kept constant in the continuous parts of the simulation. Additional the values of $\underline{q}(t_e)$ and $\underline{q_{pre}}(t_e)$ are equivalent in the continuous parts of the simulation. Since every dependents of the functions $\underline{f_s}$ and $\underline{f_a}$ are known, the continuous behavior is fully described by (3).

3.2 Discrete Behavior

The discrete behavior is controlled by events. Events are triggered by the event conditions $\underline{c}(t_e)$ and can appear at any time as well as influence the system several times.

An event occurs when a condition of $\underline{c}(t_e)$ change its value at time t_e from `false` to `true` or the other way around. This occurs if and only if for a sufficient small ϵ, one condition in $\underline{c}(t_e)$ is changed, for e.g. $\underline{c}(t_e - \epsilon)$ is `false` and for $\underline{c}(t_e + \epsilon)$ is `true`. When an event occurs all caused changes in the system can be carried out. In addition, the entire system must be determined by the function (2) to guarantee the synchronism of all equations. However,

it is not enough to determine only the discrete variables by the function $\underline{f_q}$ at this point.

The problem to be solved here is the most accurate determination of the event time t_e. For this conditions $\underline{c}(t_e)$ can be divided into three groups.

1. Conditions $\underline{c_k}(t_e)$, which also depend on continuous variables.
2. Conditions $\underline{c_d}(t_e)$, that only depend on discrete variables.
3. Conditions $\underline{c_{\text{noEvent}}}(t)$, where the `noEvent()` operator is present.

If the `smooth` operator applies to a condition in $\underline{c}(t_e)$ this condition can be categorized depending on the order of the integration method by 1. or 3., respectively.

The second and third group of conditions are easy to handle, because if a condition in $\underline{c}(t_e)$ only depends on discrete variables, then they could only change at events and the conditions $\underline{c_d}(t_e)$ must be tested only at events. The conditions $\underline{c_{\text{noEvent}}}(t)$ result logically in no events. Thus, the equations which depend on conditions $\underline{c_{\text{noEvent}}}(t)$, will be determined during the continuous integration at the output points. Hence the variables that are determined by the function (4) should be treated appropriately, like algebraic variables.

$$\underline{q_{\text{noEvent}}(t)} := g(\underline{x}(t_e), \underline{u}(t_e), \underline{q_{pre}}(t_e), \underline{c_{\text{noEvent}}}(t_e), \underline{p}, t) \tag{4}$$

The event conditions $\underline{c_k}(t_e)$, that depend only on time, can be considered separately as they lead to time events. The presence of time events is known at the start of the simulation, thus they can be treated efficiently. This separation is not yet implemented in OpenModelica, but can easily be realized.

What remains is the group of conditions, that lead to state events. For this group of conditions a time-consuming search has to be performed. These conditions have to be checked during the continuous solution as described in the next section.

Additional, discontinuous changes can caused by the `reinit()` operator to the continuous states $\underline{x}(t)$. As for purely discrete conditions $\underline{c_d}(t_e)$ the `reinit()` operator can only be activated at event times t_e . This new allocation to the states could use the function (5).

$$\underline{x(t_e)} := \underline{f_x}(\underline{x}(t), \underline{\dot{x}}(t), \underline{u}(t), \underline{q}(t_e), \underline{q_{pre}}(t_e), \underline{c}(t_e), \underline{p}, t) \tag{5}$$

3.3 Crossing Functions

In order to evaluate and check conditions $\underline{c_k}(t_e)$ during the continuous solution, they are converted into continuous functions, so-called ZeroCrossing functions. Such functions have a root if the `Boolean` condition jumps from `false` to `true`. The relation $x > 2$, for example, changes its value from `false` to `true` if $x - 2$ changes their value from less zero to greater than zero, as shown in figure 1.

The values of ZeroCrossing functions are evaluated at the points t_i as this points are provided by the continuous solution. Through the continuous monitoring of ZeroCrossing functions the interval $[t_i, t_{i+1}]$ is obtained, where at least one of the ZeroCrossing function has a zero-crossing. To determine the zero-crossing more precisely, the state values are necessary for the entire interval. A continuous solution for that can be provided by interpolating methods within the interval limits. The order of the used interpolating method should match the order of the integration process, to get an error that only depends on the used step size (cf. [9, p. 197-215]).

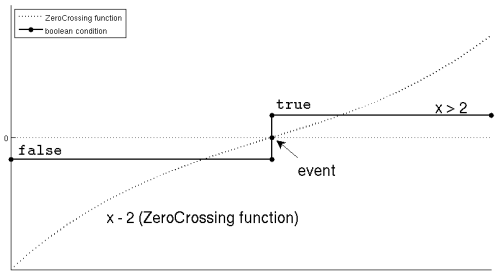

Figure 1. A boolean condition and its ZeroCrossing-function.

4. Hybrid DAE Solution Algorithm

A general approach for the simulation of hybrid systems has been developed by Cellier (cf. [2]). In the following a schematic Flowchart (see fig. 2) for the simulation is shown and each step is described.

First of all the simulation must be initialized consistently. For that the initial values are found with a simplex optimization method in OpenModelica (cf. [1]). By use of the initial conditions the initial values for the entire system can be determined with the function (2). This will also execute all initial events at time t_0.

After the initialization the main simulation loop starts with the continuous integration step that calculates the states $\underline{x}(t_{i+1})$. With the new values of $\underline{x}(t_{i+1})$, the functions $\underline{f_s}$ and $\underline{f_a}$ can be evaluated. Thus, the entire continuous system is determined.

The continuous integration step is accepted if none of the ZeroCrossing functions has a zero-crossing, i.e in $\underline{c}(t_{i+1})$ no value has changed compared to $\underline{c}(t_i)$. If no event has occurred the values can be saved and the next step can be performed.

However, if a value of $\underline{c}(t_{i+1})$ changes, an event occurred within the interval t_i and t_{i+1}. Then the exact time t_e has to be detected. Therefore a root finding method is performed on the ZeroCrossing functions of the corresponding conditions. If several ZeroCrossing functions apply the first occurring root is chosen as the next event time t_e.

The next step is to prepare the treatment of an event by evaluating the system just before an event at time $t_e - \epsilon$, and shortly after the event at $t_e + \epsilon$. Current derivative-free root finding methods work under the principle that the root is approximated through limits at the two sides,

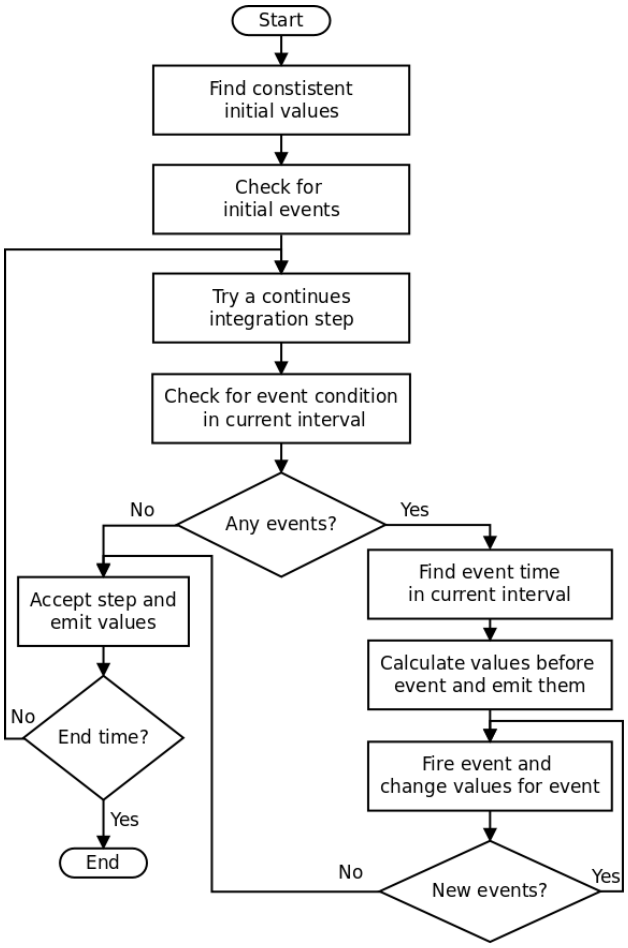

Figure 2. Schematic Flowchart for simulation hybrid models.

so that the delivered root lies somewhere in the interval $[t_e - \epsilon; t_e + \epsilon]$. Here ϵ is the tolerance level of the root finding method. Thus the necessary information to treat the event are available after the root is found.

The treatment of an event looks like that: The continuous part is evaluated at the time just before the event $t_e - \epsilon$ and all values are saved to provide them to the `pre()` operator. Then the entire system is evaluated by the function (2) at time $t_e + \epsilon$. At this point the causing event is handled and now further caused events are processed with the so-called EventIteration. Therefore the entire system constantly is re-evaluated, as long as there exist discrete variables q_j that satisfy $\texttt{pre}(q_j) \neq q_j$. Only if for all discrete variables $\texttt{pre}(q_j) = q_j$ is fulfilled, the EventIteration has reached a stable state and the next integration step can be performed.

5. Test and Evaluation in OpenModelica

In the following two test cases are presented to verify the implementation. The first model will show that `when`-equations are sorted properly and events are processed correctly. The next model is from [5] and an example for EventIteration, that works with only one re-evaluation through the correct order of all equations. Finally a hybrid Petri-Net model with its results is presented.

In order to check the correct sorting for `when`-equations the following test model is considered.

```
model when_sorting
  Real x;
  Real y;
  Boolean w(start=true);
  Boolean v(start=true);
  Boolean z(start=true);
equation
  when y > 2 and pre(z) then
    w = false;
  end when;
  when y > 2 and z then
    v = false;
  end when;
  when x > 2 then
    z = false;
  end when;
  when sample(0,1) then
    x = pre(x) + 1;
    y = pre(y) + 1;
  end when;
end when_sorting;
```

All variables in the model are discrete variables because all are contained within the `when`-equations. The variables `w` and `v` are taken to determine whether the system behaves correctly. The condition `sample(0,1)` creates events at points of time $0.0, 1.0, 2.0, \ldots$, and the variables `x` and `y` are incremented by one at this points. At time point 2.0 the variables `x` and `y` are set to 3, so the other `when`-equations are activated at point of time 2.0 firstly. The evaluation order of the equations is significant for this example. The order of evaluation has to be read on the basis of the following sorted adjacency matrix of the model.

$$\begin{array}{l} \\ y = pre(y) + 1 \\ w = false \\ x = pre(x) + 1 \\ z = false \\ v = false \end{array} \begin{array}{c} \begin{array}{ccccc} y & w & x & z & v \end{array} \\ \begin{pmatrix} \mathbf{1} & 0 & 0 & 0 & 0 \\ 1 & \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & 1 & \mathbf{1} & 0 \\ 1 & 0 & 0 & 1 & \mathbf{1} \end{pmatrix} \end{array}$$

The variable `w` is turned to `false` because the variables `y` and `pre(z)` are both `true` and are not dependent on the equation of variable `z`. The variable `v` is not turned because it depends on variable `z` and is always evaluated afterwards. Thus the condition for this equation can not be fulfilled. As a result, the equation that describes the variable `v` is not activated. However, the order of evaluation in this example is not unique, but it would not change the described behavior of the model. The results of the model are illustrated in figure 3.

The following example is developed in [5] for demonstration of the event iteration mechanism.

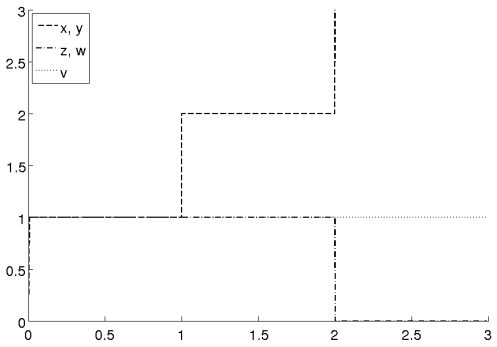

Figure 3. Results of the above example.

```
model EventIteration
  Real x(start = 1);
  discrete Real a(start = 1.0);
  Boolean z(start = false);
  Boolean h1,h2;
equation
  der(x) = a * x;
  h1 = x >= 2;
  h2 = der(x) >= 4;
  when h1 then
    y = true;
  end when;
  when y then
    a = 2;
  end when;
  when h2 then
    z = true;
  end when;
end EventIteration;
```

In the original version of the OMC which did not fulfill the synchronous data flow principle this example produced up to three EventIterations. However, if all equations are sorted in correct order and an event is handled like described above, the chain of events is treated at once. Therefore we consider again the sorted adjacency matrix to the model:

$$\begin{array}{l} \\ h1 = x >= 2 \\ y = true \\ a = 2 \\ der(x) = a * x \\ h2 = der(x) >= 4 \\ z = true \end{array} \begin{array}{c} \begin{array}{cccccc} h1 & y & a & dx & h2 & z \end{array} \\ \begin{pmatrix} \mathbf{1} & 0 & 0 & 0 & 0 & 0 \\ 1 & \mathbf{1} & 0 & 0 & 0 & 0 \\ 0 & 1 & \mathbf{1} & 0 & 0 & 0 \\ 0 & 0 & 1 & \mathbf{1} & 0 & 0 \\ 0 & 0 & 0 & 1 & \mathbf{1} & 0 \\ 0 & 0 & 0 & 0 & 1 & \mathbf{1} \end{pmatrix} \end{array} \tag{6}$$

The first event appears if `x` reaches 2 and the condition `x>=2` is fulfilled. Thus the function (2) is executed and all further caused events are treated directly. So that the variable `z` is turned to `true` after the first evaluation of (2).

6. Petri Net Example in OpenModelica

The Petri Net library developed at the University of Applied Sciences Bielefeld consists of Modelica Models for the basic components of a Petri Net (see [8]). In the library models for discrete and continuous places and transitions as well as stochastic transitions are included. With these basic components Petri Net models can be created intuitively and quickly with a graphical representation. With hierarchical capabilities of Modelica large Petri Net models can be constructed easily.

In the following we present an example of a production process modeled by the Petri Net Library. Figure 4 shows the correnspond Petri Net of the production process of crude steel, compare [3]. At first, the iron ore is transported per ship from Brasilia to a stock at the port of Rotterdam. This trip takes generally 14 days. Every 24 days a ship arrives at the port of Rotterdam. But the exact time of arrival is uncertain. The trip can take a little bit longer or shorter because of nature or other conditions. This is modeled with the aid of a stochastic Transition (Transition ship). The time of arrival is a normal distributed random variable with the expectation value $m = 24$ and the standard deviation $s = 1$. A shipload contains $360.000t$ iron ore. For this reason `add1` of Transition ship is equal to 360.000. The stock at the port can contain at most $720.000t$ iron ore. Therefore, the maximal value of the Place `stock` is fixed to 720.000. The start value of this Place is 360.000.

At the next level the iron ore is loaded from the stock to several trains. A train can contain $5000t$ iron ore and the drive to the steel production in Duisburg takes 8 hours. The iron ore is delivered "just in time" to the production process. Hence, no other stock is needed. The discrete Transition train represents the transport from Rotterdam to Duisburg. The delay is $1/3$ day ($= 8$ hours) and `sub1 = add1 = 5000`. The iron ore (Place pro) and the coke (Place coke) are mixed in the sintering plant. It accrues the intermediate product sinter (Place `I1`). For one ton employed iron ore $0.2t$ coke is needed and $0.73t$ sinter is produced. This production step is modeled continuously by means of the Transition `Si`. The edge weightings are the following:

```
sub1 = 0.2 pro.t
sub2 = pro.t
add1 = 0.73 pro.t
```

The sinter is further processed in the blast furnace to hot metal (Place `I2`). In addition, the by-products slag (Place `slag`) and blast furnace dust (Place `dust1`) are produced. For one ton employed sinter $0.2t$ coke is needed and $0.1t$ slag, $0.65t$ hot metal and $0.01t$ blast furnace dust are produced. The Transition `Fu` displays this. The edges weightings are:

```
sub1 = 0.2 I1.t
sub2 = I1.t
add1 = 0.1 I1.t
add2 = 0.65 I1.t
add3 = 0.01 I1.t
```

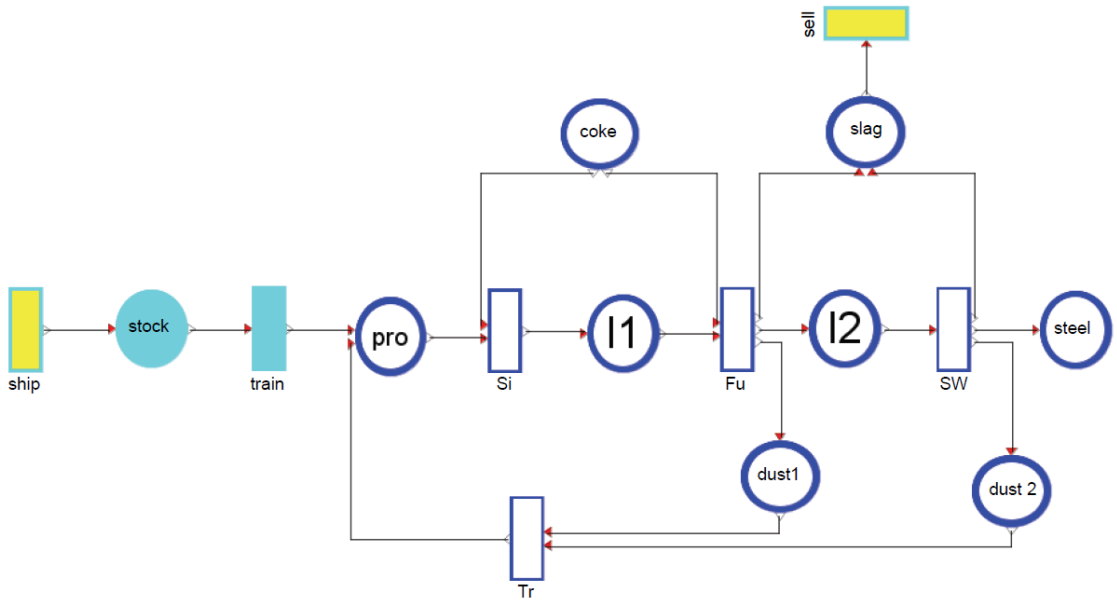

Figure 4. Steel production process.

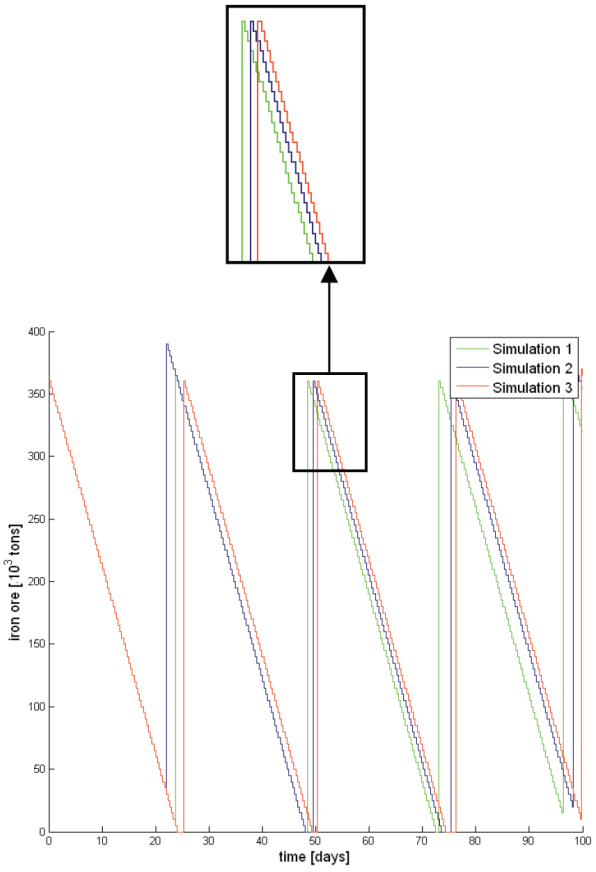

Figure 5. Three simulation results of the iron ore stock at the port Rotterdam.

The by-product slag is sold to building industry. When $50.000t$ slag are produced the company is informed but it is uncertain when the company arrives to pick up the slag and how long this procedure takes. This is modeled with a stochastic Transition with a normal distributed delay ($m = 1/2$ and $s = 1/8$) and `sub1 = 50.000`. In the last production step the hot metal is processed to crude steel (Place `steel`) in the steel works. Slag (Place `slag`) and converter dust (Place `dust2`) are the by-products here. For one ton employed hot metal $0.13t$ slag, $0.8t$ crude steel and $0.05t$ converter dust are produced. The Transition `SW` represents the steel works. The edge weightings are:

```
sub1 = I2.t
add1 = 0.13 I2.t
add2 = 0.8 I2.t
add3 = 0.05 I2.t
```

Iron ore can be substituted by blast furnace dust (Place `dust1`) and converter dust (Place `dust2`). This is modeled with the Transition `Tr` and the edges weightings are:

```
sub1 = dust1.t
sub2 = dust2.t
add1 = 0.1 (dust1.t +dust2.t)
```

Following, some simulation results are shown. Figure 5 displays three possible progressions of the stock of iron ore at the port of Rotterdam. Every progression is different because of the stochastic modeling. The stock is limited to $720.000t$ iron ore. Hence, this border is not exceed. The iron ore is loaded to trains. Every 8 hours a train drives with $5000t$ iron ore to Duisburg. These are the discrete stages in the magnification. The iron ore is exhausted in all simulations at specific time points:

Simulation1	Simulation2	Simulation3
48 - 48.5	48 - 49.5	24 - 25.25
72.5 - 73	73.5 - 75.4	49.25 - 50.31
		74.31 - 76.28

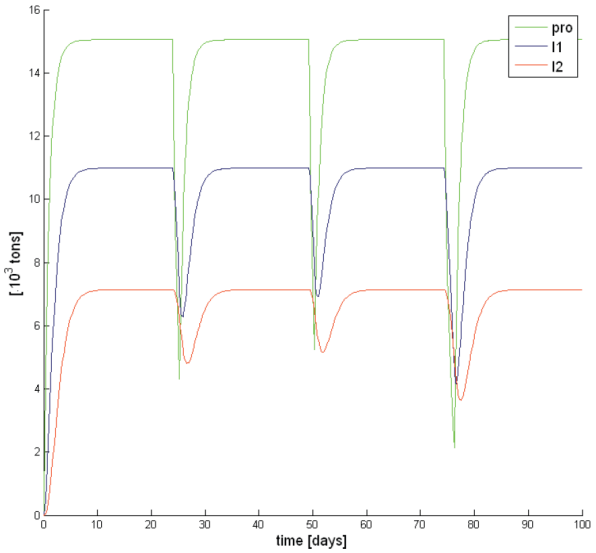

Figure 6. The progressions of iron ore (pro), sinter (I1) and hot metal (I2) (simulation 3).

This causes bottlenecks in the production process. The next figure 6 shows the progression of iron ore, sinter and hot metal of simulation 3. The decrease after day 24.3, 49.6 and 74.4 is caused by the exhaused stocks. Figure 7 illustrates the bottleneck in the production process of simulation 3, too. The exhausted stocks are reflected in the amount of crude steel. The production is decreased after every empty stock period.

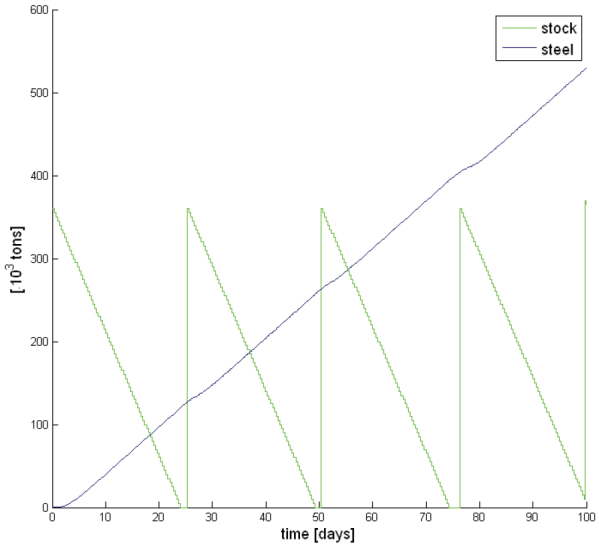

Figure 7. Stock of iron ore (stock) and produced crude steel (steel) by comparison (simulation 3).

The conclusion of these simulations is that the delivery period of iron ore has to be reduced. The new period has to be big enough that only small bottlenecks appear and small enough that no high stocks accumulate. If for example a period of 22.5 days is chosen, the probability of a bottleneck is 6.7% and the probability that this bottleneck takes longer than one day is 0.62%. Now is the task to find the "optimal" solution between bottlenecks and stock costs. Figure 8 shows three simulation results of the progression of the iron ore stock if the delivery period is 22.5 days.

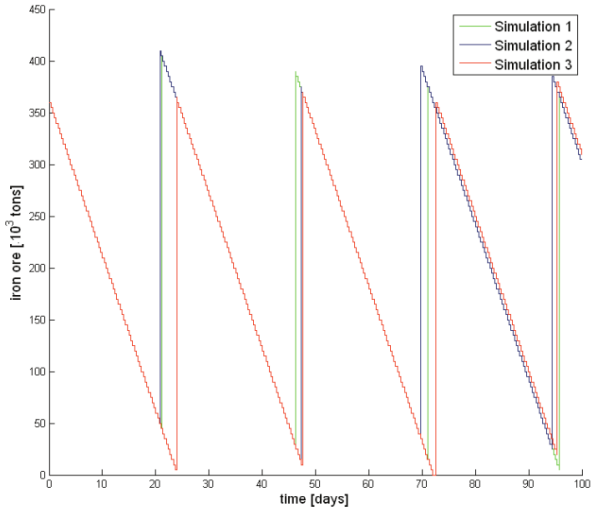

Figure 8. Three simulation results of the iron ore stock with a delivery period of 22.5.

7. Conclusion and Future Work

In this paper the algorithmic approach with respect to the synchronous event handling in the OpenModelica Compiler is stated. The advantage of this procedural method has been clearly demonstrated on some application examples.

Previously, the use of when-equations had to be avoided in the OpenModelica version of the Petri Net library. In order to get correct simulation results the equivalent formulation `if edge(b) then ... end if` had to be employed. This also yields for all other event driven Modelica libraries. Since this work this reformulation is unnecessary. The description of the Petri Net Library for OpenModelica has been optimized, so that now events can be specified with the aid of when-equations.

The current implementation represents the concept, but efficiency can be further improved in the near future. For example, an increase in performance can be achieved by more advanced root finding methods. Furthermore, the efficiency can be enhanced by handling time events separately with a suitable step-size control.

Acknowledgments

The German Ministry BMBF has partially funded this work (BMBF Förderkennzeichen: 01IS09029C) within the ITEA2 project OPENPROD (http://www.openprod.org).

References

[1] Bernhard Bachmann, Peter Aronsson, and Peter Fritzson. Robust initialization of differential algebraic equations. In *Proceedings 5th Modelica Conference, Vienna, Austria*, 2006.

[2] François E. Cellier. *Combined Continuous/Descrete System Simulation by use of digital computers: Techniques and Tools*. PhD thesis, ETH Zürich, 1979.

[3] Harald Dyckhoff and Thomas Stefan Spengler. *Produktionswirtschaft*. Springer-Verlag Berlin Heidelberg, 2005.

[4] Hilding Elmqvist, François E. Cellier, and Martin Otter. Object-oriented modeling of hybrid systems. In *European Simulation Symp.*, pages 31–41, 1993.

[5] Håkan Lundvall, Peter Fritzson, and Bernhard Bachmann. Event handling in the openmodelica compiler and runtime system. Technical report, PELAB,The Institute of Technology, Linköpings universitet, 2008.

[6] Modelica Association. *Modelica - A unified Object-Oriented Language for Physical Systems Modeling Language Specification - Version 3.2*, 03 2010.

[7] Martin Otter, Hilding Elmqvist, and Sven Erik Mattsson. Hybrid modeling in modelica based on the synchronous data flow principle. In *Proceedings of CACSD, Symposium on Computer Aided Control System Desig*, 1999.

[8] Sabrina Pross and Bernhard Bachmann. A petri net library for modeling hybrid systems in openmodelica. In *Proceedings 7th Modelica Conference, Como, Italy*, pages 454–463. The Modelica Association, 2009.

[9] Lena Wunderlich. *Analysis and Numerical Solution of Structured and Switched Differential-Algebraic Systems*. PhD thesis, TU Berlin, 2008.