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

Simulation hybrid systems in Modelica

Willi Braun, Bernhard Bachmann and Sabrina Pross

Department of Applied Mathematics
University of Applied Sciences Bielefeld
33609 Bielefeld, Germany

2010-10-03

Outline

Hybrid models

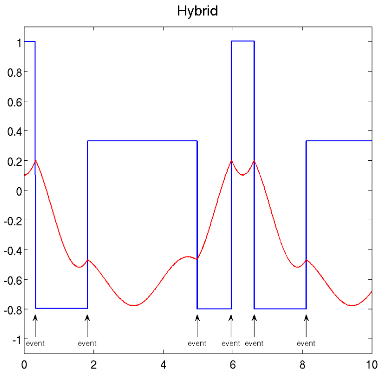
Hybrid modeling

Hybrid

- Mixed systems with continuous and discrete components
- Simulation needs handling with events and discontinuities

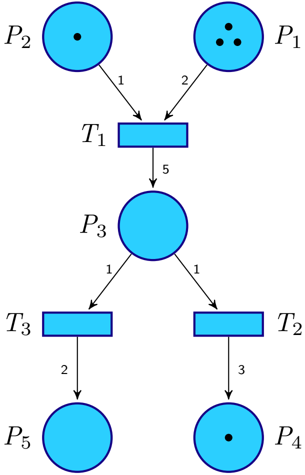
Applications

- Switched electric circuits
- Controlled systems
- Petri Nets



Hybrid models

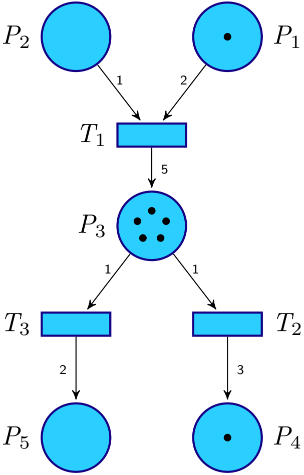
Hybrid Petri Net



Elements of a Petri Net

- Fundamental items are places and transitions
- Directed arcs connect items

Hybrid models
Hybrid Petri Net

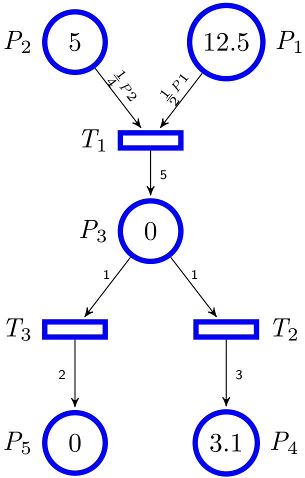


Elements of a Petri Net

- Fundamental items are places and transitions
- Directed arcs connect items

Hybrid models

Hybrid Petri Net



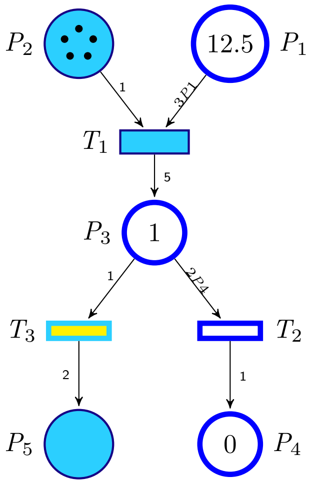
Elements of a Petri Net

- Fundamental items are places and transitions
- Directed arcs connect items

Modifications

- The fireing speed can be discribed by differential equations for continuous behaviour
- Transistions may have a stochastic delay
- Edges may have weightings, threshold and inhibition

Hybrid models
Hybrid Petri Net



Elements of a Petri Net

- Fundamental items are places and transitions
- Directed arcs connect items

Modifications

- The fireing speed can be discribed by differential equations for continuous behaviour
- Transitions may have a stochastic delay
- Edges may have weightings, threshold and inhibition

$\Rightarrow$ Hybrid models

Hybrid models
Petri Nets in OpenModelica

Library in OpenModelica

- Continuous, discrete and stochastic places and transitions can be combined
- Combined Petri Nets are versatile applicable
  For example: production or biological processes

Problems in OpenModelica

- "`when equations`" were not treated synchronously

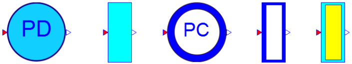


Figure: Elements of the PetriNet-Library

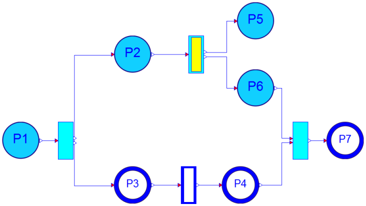
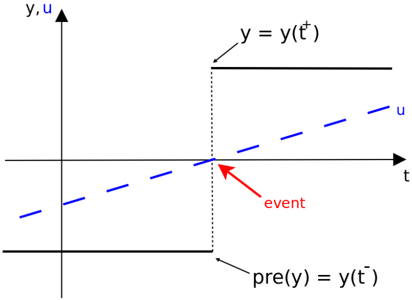


Figure: Example network

Hybrid models

Modelling events with Modelica

```
model example_if
 input Real u;
 Real y;
equation
 y = if u > 0 then 1 else -1;
end example_if;
```



- Conditional expressions like `u > 0` trigger events.
- If events occurs the value is stored twice.
- In this example `y` is the right limit and `pre(y)` is the left limit.

Hybrid models
Hybrid Modelica DAE-System

Flatten Modelica model:

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

$\downarrow$ matching and sorting algorithm transform to

$$\underline{z} = \begin{pmatrix} \underline{\dot{x}}(t) \\ \underline{y}(t) \\ \underline{q}(t_e) \end{pmatrix} = \begin{pmatrix} \underline{f}(\underline{x}(t), \underline{u}(t), \underline{q_{pre}}(t_e), \underline{c}(t_e), \underline{p}, t) \\ \underline{g}(\underline{x}(t), \underline{u}(t), \underline{q_{pre}}(t_e), \underline{c}(t_e), \underline{p}, t) \\ \underline{h}(\underline{x}(t), \underline{u}(t), \underline{q_{pre}}(t_e), \underline{c}(t_e), \underline{p}, t) \end{pmatrix}$$

Hybrid models
Synchronous Data-flow principle

Sorting of when equations

```
//known Variable: u
  when y2 > 2 then
    y3 = f1(y4);
  end when;
  y4 = f2(y5);
  when y1 > 0 then
    y5 = f3(u);
    y2 = f4(y4);
  end when;
  y1 = f5(u);
```

Incidence-Matrix

$$
\begin{array}{c}
\\
y3 = f1(y4) \\
y4 = f2(y5) \\
y5 = f3(u) \\
y2 = f4(y4) \\
y1 = f5(u)
\end{array}
\quad
\begin{array}{c}
\begin{array}{ccccc} y1 & y2 & y3 & y4 & y5 \end{array} \\
\begin{pmatrix}
0 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 1 & 1 \\
1 & 0 & 0 & 0 & 1 \\
1 & 1 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 0
\end{pmatrix}
\end{array}
$$

Hybrid models
Synchronous Data-flow principle

Sorting of when equations

```
//known Variable: u
  y1 = f5(u);
  when y1 > 0 then
    y5 = f2(u);
  end when;
  y4 = f3(y5);
  when y1 > 0 then
    y2 = f4(y4);
  end when;
  when y2 > 2 then
    y3 = f1(y4);
  end when;
```

Incidence-Matrix

$$
\begin{array}{r c}
 & \begin{array}{ccccc} y1 & y5 & y4 & y2 & y3 \end{array} \\
\begin{array}{r} y1 = f5(u) \\ y5 = f3(u) \\ y4 = f3(y5) \\ y2 = f4(y4) \\ y3 = f1(y4) \end{array} &
\begin{pmatrix}
\mathbf{1} & 0 & 0 & 0 & 0 \\
1 & \mathbf{1} & 0 & 0 & 0 \\
0 & 1 & \mathbf{1} & 0 & 0 \\
1 & 0 & 1 & \mathbf{1} & 0 \\
0 & 0 & 1 & 1 & \mathbf{1}
\end{pmatrix}
\end{array}
$$

Hybrid models
Synchronous Data-flow principle

Sorting of when equations

```
//known Variable: u
  y1 = f5(u);
  when y1 > 0 then
    y5 = f2(u);
  end when;
  y4 = f3(y5);
  when y1 > 0 then
    y2 = f4(y4);
  end when;
  when y2 > 2 then
    y3 = f1(y4);
  end when;
```

Incidence-Matrix

$$\begin{array}{c} \\ y1 = f5(u) \\ y5 = f3(u) \\ y4 = f3(y5) \\ y2 = f4(y4) \\ y3 = f1(y4) \end{array} \quad \begin{array}{c} \begin{array}{ccccc} y1 & y5 & y4 & y2 & y3 \end{array} \\ \begin{pmatrix} \mathbf{1} & 0 & 0 & 0 & 0 \\ 1 & \mathbf{1} & 0 & 0 & 0 \\ 0 & 1 & \mathbf{1} & 0 & 0 \\ 1 & 0 & 1 & \mathbf{1} & 0 \\ 0 & 0 & 1 & 1 & \mathbf{1} \end{pmatrix} \end{array}$$

Sorting equations

Sorting is based on all equations due to the correct order of evaluation at all time points.

Hybrid models
Hybrid Modelica DAE-System

Flatten Modelica model:

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

$\downarrow$ matching and sorting algorithm transform to

$$\underline{z} = \begin{pmatrix} \underline{\dot{x}}(t) \\ \underline{y}(t) \\ \underline{q}(t_e) \end{pmatrix} = \begin{pmatrix} \underline{f}(\underline{x}(t), \underline{u}(t), \underline{q_{pre}}(t_e), \underline{c}(t_e), \underline{p}, t) \\ \underline{g}(\underline{x}(t), \underline{u}(t), \underline{q_{pre}}(t_e), \underline{c}(t_e), \underline{p}, t) \\ \underline{h}(\underline{x}(t), \underline{u}(t), \underline{q_{pre}}(t_e), \underline{c}(t_e), \underline{p}, t) \end{pmatrix}$$

Hybrid models
Hybrid Modelica DAE-System

Flatten Modelica model:

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

$\downarrow$ matching and sorting algorithm transform to

$$\underline{z} = \begin{pmatrix} \underline{\dot{x}}(t) \\ \underline{y}(t) \\ \underline{q}(t_e) \end{pmatrix} = \begin{pmatrix} \underline{f}(\underline{x}(t), \underline{u}(t), \underline{q_{pre}}(t_e), \underline{c}(t_e), \underline{p}, t) \\ \underline{g}(\underline{x}(t), \underline{u}(t), \underline{q_{pre}}(t_e), \underline{c}(t_e), \underline{p}, t) \\ \underline{h}(\underline{x}(t), \underline{u}(t), \underline{q_{pre}}(t_e), \underline{c}(t_e), \underline{p}, t) \end{pmatrix}$$

We get four blocks:

continuous $\underline{f}$ and $\underline{g}$ $\rightarrow$ derivative states and algebraic variables

discrete $\underline{h}$ $\rightarrow$ discrete algebraic variables

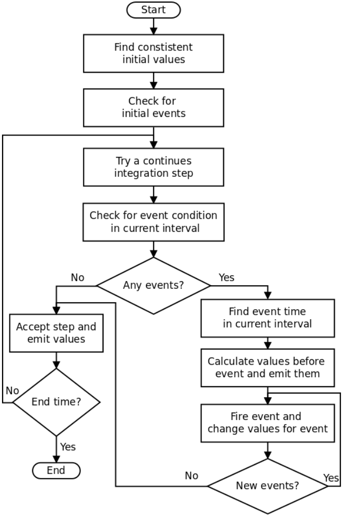
all $\underline{z}$ $\rightarrow$ all three block together

An additional block to manage the conditions for events:

$\underline{c}(t) \rightarrow$ Zero Crossing functions

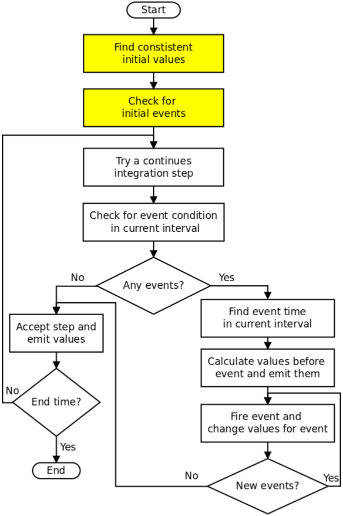
Implementation in OpenModelica

Approach to simulate hybrid models



Implementation in OpenModelica

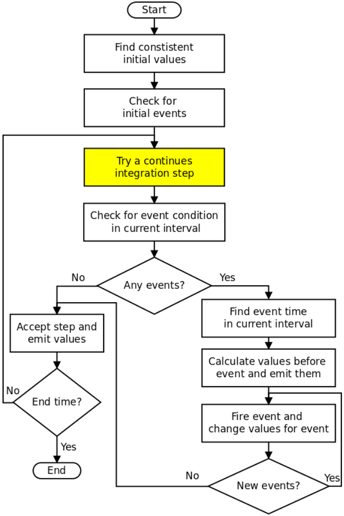
Approach to simulate hybrid models



Intial Step

- Initial-value problem is solved by a simplex-method
- Initial Zero-crossing functions and check for initial events

Implementation in OpenModelica
Continuous integration



Integration step

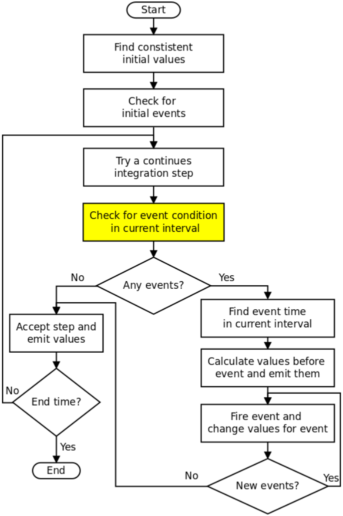
- Integration method: $x_{i+1} = \Phi(x_i)$
- Calculate for the next step t_{i+1} the new state vector $\underline{x}(t_{i+1})$
- Evaluate continuous blocks $\underline{f}$ and $\underline{g}$

$$\underline{\dot{x}}(t_{i+1}) = \underline{f}(\underline{x}(t_{i+1}), \underline{q}(t_i), \underline{q_{pre}}(t_i), \underline{c}(t_{i+1}), t)$$

$$\underline{y}(t_{i+1}) = \underline{g}(\underline{x}(t_{i+1}), \underline{q}(t_i), \underline{q_{pre}}(t_i), \underline{c}(t_{i+1}), t)$$

Implementation in OpenModelica

Check for event conditions

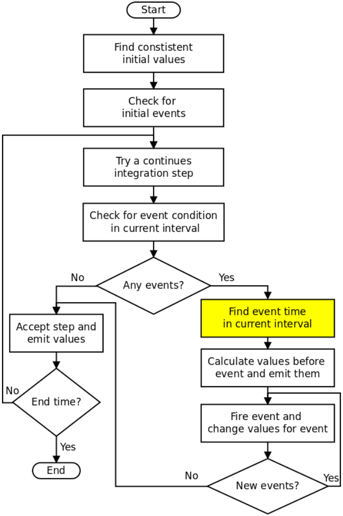


Check for zero-crossing

- Conditions are converted into zero-crossing functions
- $x < 2$ changes from `false` to `true` when $x - 2$ crosses zero
- If any zero-crossing becomes `true` an event is fired

Implementation in OpenModelica

Find event time

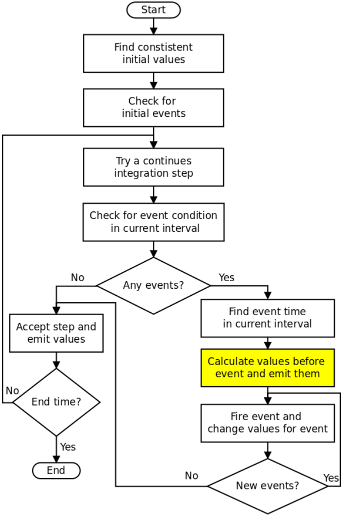


Root-finding method

- Find root in interval $[t_i; t_{i+1}]$ as event time t_e
- Bisection is a simple and robust method, but it is also relative slowly
- All methods approximate the root by setting limits on each side of t_e
- Additional we have $t_e - \epsilon$ and $t_e + \epsilon$

Implementation in OpenModelica

Handle event



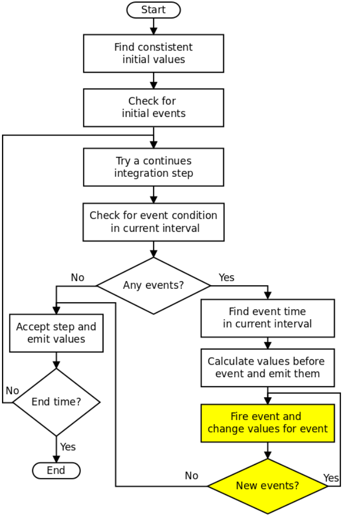
Handle event

1. Determine states at $t_e - \epsilon$ with interpolation
2. Determine continuous blocks by using functions $\underline{f}$ and $\underline{g}$
3. Save all variables as values for `pre()` and emit them to result file

$$\dot{\underline{x}}(t_e - \epsilon) = \underline{f}(\underline{x}(t_e - \epsilon), \underline{q}(t_i), \underline{q_{pre}}(t_i), \underline{c}(t_e - \epsilon), t)$$

$$\underline{y}(t_e - \epsilon) = \underline{g}(\underline{x}(t_e - \epsilon), \underline{q}(t_i), \underline{q_{pre}}(t_i), \underline{c}(t_e - \epsilon), t)$$

Implementation in OpenModelica
Handle event



Handle event

1. Determine states at $t_e + \epsilon$ with interpolation
2. Evaluate all blocks by using function $\underline{z}$
3. Check for changes of discrete variables
4. Event Iteration

$$\underline{z} = \begin{pmatrix} \dot{\underline{x}}(t_e+\epsilon) \\ \underline{y}(t_e+\epsilon) \\ \underline{q}(t_e+\epsilon) \end{pmatrix} = \begin{pmatrix} \underline{f}(\underline{x}(t_e+\epsilon), \underline{q_{pre}}(t_e), \underline{c}(t_e), t) \\ \underline{g}(\underline{x}(t_e+\epsilon), \underline{q_{pre}}(t_e), \underline{c}(t_e), t) \\ \underline{h}(\underline{x}(t_e+\epsilon), \underline{q_{pre}}(t_e), \underline{c}(t_e), t) \end{pmatrix}$$

Examples

Example for correct sorting of when-equations

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

Examples

Example for correct sorting of when-equations

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

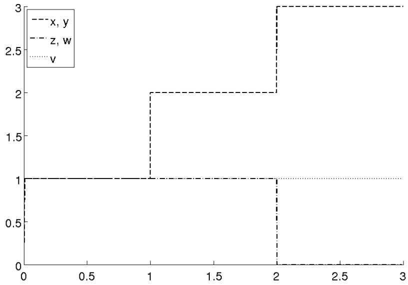
Incidence-Matrix

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

Examples

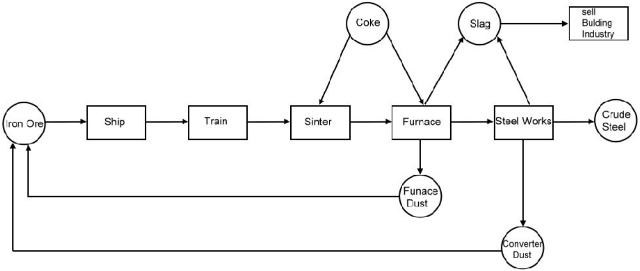
Example for correct sorting of when-equations

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



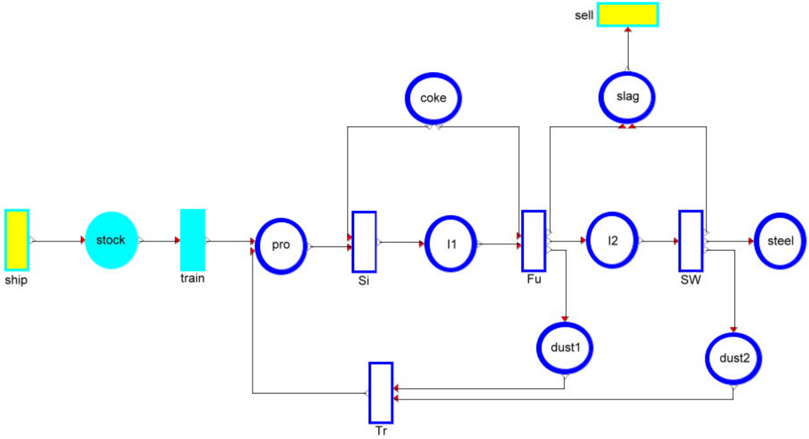
Examples

Petri Net example for a production process



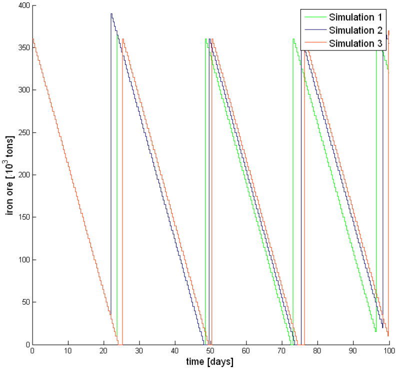
Examples

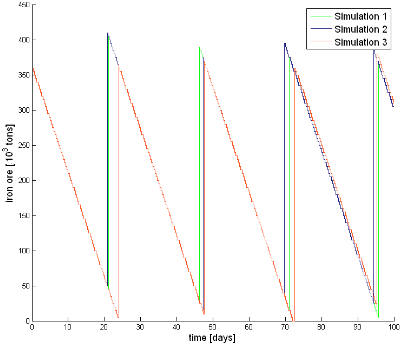
Petri Net example for a production process



Examples

Petri Net example for a production process





Summary

- With this algorithmic approach we can simulate many synchronously appearing events.
- The Petri Net Library for OpenModelica has been optimized, so that now events can be specified with the aid of "when-equation".

- The performance of the current implementation can be further improved:
    - by handling time events separately with a suitable step-size control.
    - by achieving more advanced root finding methods.