Modelica Tutorial –
Modeling and Simulation with
OpenModelica and MathModelica

Jan Brugård
MathCore Engineering AB
jan.brugard@mathcore.com

David Broman
Linköping University
Department of Computer and Information Science
davbr@ida.liu.se

Copyright © Peter Fritzson, David Broman, and Jan Brugård

Agenda

Part I
Introduction to Modelica and a
demo example

Part II
Modelica environments

Part III
Modelica language concepts
and textual modeling

Part IV
Graphical modeling and the
Modelica standard library
Part I

Introduction to Modelica and a demo example

What is Modelica?

A language for modeling of complex physical systems

- Robotics
- Automotive
- Aircrafts
- Satellites
- Power plants
- Systems biology
What is Modelica?

A language for **modeling** of complex physical systems

Primary designed for **simulation**, but there are also other usages of models, e.g. optimization.

Free, open language specification:

There exist several free and commercial tools, for example:

- OpenModelica from OSMC
- MathModelica by MathCore
- Dymola by Dassault systems / Dynasim
- SimulationX by ITI
- MapleSim by MapleSoft

Available at: www.modelica.org
What is special about Modelica?

Multi-Domain Modeling

Keeps the physical structure

Acausal model (Modelica)

Causal block-based model (Simulink)
What is special about Modelica?

**Multi-Domain Modeling**

A textual *class-based* language

**OO primary used for as a structuring concept**

**Behaviour described declaratively using**

- Differential algebraic equations (DAE) (continuous-time)
- Event triggers (discrete-time)

```modelica
class VanDerPol "Van der Pol oscillator model"
  Real x(start = 1) "Descriptive string for x";
  Real y(start = 1) "y coordinate";
  parameter Real lambda = 0.3;
  equation
    der(x) = y;
    der(y) = -x + lambda*(1 - x*x)*y;
end VanDerPol;
```

**Visual Acausal Component Modeling**

**Typed Declarative Textual Language**

Variable declarations

Differential equations
What is special about Modelica?

- **Multi-Domain Modeling**
- **Visual Acausal Component Modeling**
- **Typed Declarative Textual Language**

Hybrid modeling = continuous-time + discrete-time modeling

- Continuous-time
- Discrete-time

Coming up in part III

- **Typed Declarative Textual Language**
- **Hybrid Modeling**

Part I: Introduction and demo example
Part II: Overview of environments
Part III: Modelica language and textual modeling
Part IV: Graphical modeling and the standard library
Coming up in part IV

- Multi-Domain Modeling
- Visual Acausal Component Modeling

Part I
Introduction and demo example

Part II
Overview of environments

Part III
Modelica language and textual modeling

Part IV
Graphical modeling and the standard library

Brief Modelica History

Modelica design group meetings
- First meeting in fall 1996
- International group of people with expert knowledge in both language design and physical modeling
- Industry and academia

Modelica Language Versions
- v1.0 (1997), v2.0 (2002) v.2.2 (2005)
- 3.0 released September 2007 (latest)

Modelica Association established 2000
- Open, non-profit organization

Modelica Conferences
- 6 international conferences (2000-2008)
- The 7th International Modelica conference September 20-22, Como, Italy
Part II

Modelica environments
Dymola

- Dynasim (Dassault Systemes)
- Sweden
- First Modelica tool on the market
- Main focus on automotive industry
- www.dynasim.com

Simulation X

- ITI
- Germany
- Mechatronic systems
- www.simulationx.com
MapleSim

- Maplesoft
- Canada
- Latest Modelica tool on the market
- Integrated with Maple
- www.maplesoft.com

OpenModelica

- Open Source Modelica Consortium (OSMC)
- Sweden
- Open source
- www.openmodelica.org
MathModelica

- MathCore
- Sweden
- Released 2006
- General purpose
- Mathematica connection
- www.mathcore.com

OpenModelica

- Advanced Interactive Modelica compiler (OMC)
  - Supports parts of the Modelica Language
- Basic environment for creating models
  - OMSHELL – an interactive command handler
  - OMNotebook – a literate programming notebook
  - MDT – an advanced textual environment in Eclipse
Open Source Modelica Consortium

Part I
Introduction and demo example

Part II
Overview of environments

Part III
Modelica language and textual modeling

Part IV
Graphical modeling and the standard library

Open-source community services
- Website and Support Forum
- Version-controlled source base
- Bug database
- Development courses
- www.openmodelica.org

Founded Dec 4, 2007

Industrial members
- Bosch-Rexroth AG, Germany
- ABB Corporate Research AB, Sweden
- Siemens Industrial Turbomachinery, Sweden
- Equa Simulation AB, Sweden
- TLK Thermo, Germany
- VTT, Finland
- MostforWater, Belgium
- MapleSoft, Canada
- Emmeskay Inc., USA
- IFP, Paris, France
- MathCore Engineering AB

Academic members
- Linköping University, Sweden
- Technical Univ of Hamburg-Harburg, Germany
- Technical Univ of Braunschweig, Germany
- Université Laval, Canada
- University of Queensland, Australia
- Griffith University, Australia
- Politecnico di Milano, Italy

Code Statistics

MathModelica Components

Modeling

Simulation

Documentation

Analysis

Part I
Introduction and demo example

Part II
Overview of environments

Part III
Modelica language and textual modeling

Part IV
Graphical modeling and the standard library
Part III

Modelica language concepts and textual modeling
## Acausal Modeling

The order of computations is not decided at modeling time

<table>
<thead>
<tr>
<th>Acausal</th>
<th>Causal</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Visual Component Level</strong></td>
<td><img src="image" alt="Diagram" /></td>
</tr>
</tbody>
</table>
| **Equation Level** | A resistor equation: 
\[ R \cdot i = v; \] | Causal possibilities:  
\[ i := \frac{v}{R}; \]  
\[ v := R \cdot i; \]  
\[ R := \frac{v}{i}; \] |

### Typical Simulation Process

```
“Static” semantics / compile time
Modelica model \[\rightarrow\] Hybrid DAE \[\rightarrow\] Executable
```

```
“Dynamic” semantics / run time
Elaboration
Equation Transformation & Code generation
Simulation
Simulation Result
```
Simple model - Hello World!

Equation: \( x' = -x \)
Initial condition: \( x(0) = 1 \)

```
model HelloWorld "A simple equation"
  Real x(start=1);
  parameter Real a = -1;
  equation
    der(x) = a*x;
end HelloWorld;
```

Simulation in OpenModelica environment

```
simulate(HelloWorld, stopTime = 2)
plot(x)
```

Differential Algebraic Equations

Informally: ODE + algebraic equations = DAE

General representation of DAEs:

\[
0 = f(t, \dot{x}(t), x(t), y(t), u(t), p)
\]

- \( t \): time
- \( \dot{x}(t) \): vector of differentiated state variables
- \( x(t) \): vector of state variables
- \( y(t) \): vector of algebraic variables
- \( u(t) \): vector of input variables
- \( p \): vector of parameters and/or constants

```
model DAEexample
  Real x(start=0.9, fixed=true);
  Real y;
  equation
    der(y)+sin(x) = sin(time);
    x - y = exp(-0.9*x)*cos(y);
end DAEexample;
```

Typically, the compiler transforms the DAE to an ODE before simulation, sometimes using an index reduction algorithms.
Textual and Graphical Models

Part I: Introduction and demo example
Part II: Overview of environments
Part III: Modelica language and textual modeling
Part IV: Graphical modeling and the standard library

Equations and Inheritance

model TwoPin
Pin p;
Pin n;
Real v;
Real i;
equation
v = p.v - n.v;
0 = p.i + n.i;
i = p.i;
end TwoPin;

model Resistor
extends TwoPin;
Real R = 100;
equation
R*i = v;
end Resistor;

model Inductor
extends TwoPin;
Real L = 1;
equation
R*i = v;
end Inductor;

Inherits equations and components from TwoPin
Pin p, n and Reals v and i are copied to the subclass
Equations are copied as well.

Enables later modification of component
Modification of parameter value
Named component = model instance
Connect equations
Used model (defined elsewhere)
Connectors (Ports)

```model TwoPin
  Pin p;
  Pin n;
  Real v;
  Real i;
  equation
    v = p.v - n.v;
    0 = p.i + n.i;
    i = p.i;
  end TwoPin;
```

Connectors are instances of a connector class.

```connector Pin
  Real v;
  flow Real i;
end Pin;
```

Connections and Flow Variables

```model Circuit
  protected
    replaceable Resistor R1(R=10);
    replaceable Inductor L(L=0.1);
    VsourceAC AC;
    Ground G;
  equation
    connect(AC.p, R1.p);
    connect(R1.n, L.p);
    connect(L.n, AC.n);
  end Circuit;
```

Equations from potential variables:
- \( L.n.v = AC.n.v \)
- \( AC.n.v = G.p.v \)

Equation from flow variables:
- \( L.n.i + AC.n.i + G.p.i = 0 \)
Hybrid Modeling

Hybrid modeling = continuous-time + discrete-time modeling

- A point in time that is instantaneous, i.e., has zero duration
- An event condition so that the event can take place
- A set of variables that are associated with the event
- Some behavior associated with the event, e.g. conditional equations that become active or are deactivated at the event

Event creation – if

if-equations, if-statements, and if-expressions

```
if <condition> then 
  <equations>
else if <condition> then 
  <equations>
else
  <equations>
end if;
```

```
model Diode "Ideal diode"
  extends TwoPin;
  Real s;
  Boolean off;
  equation
    off = s < 0;
    if off then
      v = s
    else
      v = 0;
    end if;
  i = if off then 0 else s;
end Diode;
```

False if s<0

If-equation choosing equation for v

If-expression
Event creation – when

**when-equations**

```
when <conditions> then
  <equations>
end when;
```

---

**Time event**

```
when time >= 10.0 then
  ...
end when;
```

Only dependent on time, can be scheduled in advance

**State event**

```
when sin(x) > 0.5 then
  ...
end when;
```

Related to a state. Check for zero-crossing

---

Generating Repeated Events

The call `sample(t0,d)` returns true and triggers events at times `t0+i*d`, where `i=0,1,...`

```
model SamplingClock
  Integer i;
  discrete Real r;
equation
  when sample(2,0.5) then
    i = pre(i)+1;
    r = pre(r)+0.3;
  end when;
end SamplingClock;
```

Variables need to be discrete

---

Creates an event after 2 s, then each 0.5 s

`pre(...)` takes the previous value before the event.
Reinit - discontinuous changes

The value of a continuous-time state variable can be instantaneously changed by a reinit-equation within a when-equation.

```model BouncingBall "the bouncing ball model"
parameter Real g=9.81; //gravitational acc.
parameter Real c=0.90; //elasticity constant
Real height(start=10),velocity(start=0);
equation
  der(height) = velocity;
  der(velocity)=-g;
  when height<0 then
    reinit(velocity, -c*velocity);
  end when;
end BouncingBall;
```

Modelica – large and complex

We have just “scratched on the surface of the language”

Examples of the features which has not been covered

- Functions and algorithm sections
- Arrays and matrices
- Inner / outer variables (lookup in instance hierarchy)
- Annotations
- Loop constructs
- Partial classes
- Packages, blocks...

And much more...
Exercise 2.1 Simulate Bouncing Ball

- Open OpenModelica OMNotebook
- Open file "bouncingball.onb"
- Each cell is evaluated by first clicking on an input cell and then pressing <shift>-<enter>. Do this in order for each of the cells below. When the final cell is evaluated a plot of the height variable should be shown in the notebook.
- b) Change the gravitational acceleration to the one on the moon. Change the simulation time to 25 seconds. Evaluate each cell again and plot the result.

Exercise 2.2 Bouncing Ball with Stairs

- Move the model back to earth again...
- Assume that we shall simulate a ball falling down a staircase in a house.
- Each step has a height of 0.2m and a length of 0.3m. The ball is moving in the x direction with a constant speed of 1 m/s. The height between the groundfloor and the first floor is 3m. The ball is dropped 1m above the first floor and 2m from the staircase.

**Task**: After how long time does the ball first hit the groundfloor?

**Tips**: Plot two variables $x$ and $y$:

```
plot({x,y});
```
Part IV

Graphical modeling and the Modelica standard library

Building an Electric Circuit

\[ u(t) = R_1 i_1(t) + \frac{1}{C} \int i_1(t) \, dt \]
\[ u(t) = R_2 i_2(t) + L \frac{d i_2(t)}{dt} \]
\[ i(t) = i_1(t) + i_2(t) \]
\[ i_1(t) = \frac{1}{R_1} \left( u(t) - \frac{1}{C} \int i_1(t) \, dt \right) \]
\[ i_2(t) = \frac{1}{R_2} \left( u(t) - L \frac{d i_2(t)}{dt} \right) \]
Building an Electric Circuit

\[ u(t) = R_i \theta(t) + \frac{1}{C} \int i_1(t) \, dt \]

\[ u(t) = R_i \theta(t) + L \frac{d^2 x(t)}{dt^2} \]

\[ i(t) = i_1(t) + i_2(t) \]

\[ i_1(t) = \frac{1}{R_i} \left[ u(t) - \frac{1}{C} \int i_1(t) \, dt \right] \]

\[ i_2(t) = \frac{1}{L} \left[ U(t) - L \frac{d x(t)}{dt} \right] \]
Some Other Domains

<table>
<thead>
<tr>
<th>Domain Type</th>
<th>Potential</th>
<th>Flow</th>
<th>Carrier</th>
<th>Modelica Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>Electrical</td>
<td>Voltage</td>
<td>Current</td>
<td>Charge</td>
<td>Electrical. Analog</td>
</tr>
<tr>
<td>Translational</td>
<td>Position</td>
<td>Force</td>
<td>Linear momentum</td>
<td>Mechanical. Translational</td>
</tr>
<tr>
<td>Rotational</td>
<td>Angle</td>
<td>Torque</td>
<td>Angular momentum</td>
<td>Mechanical. Rotational</td>
</tr>
<tr>
<td>Magnetic</td>
<td>Magnetic potential</td>
<td>Magnetic flux rate</td>
<td>Magnetic flux</td>
<td></td>
</tr>
<tr>
<td>Hydraulic</td>
<td>Pressure</td>
<td>Volume flow</td>
<td>Volume</td>
<td>HyLibLight</td>
</tr>
<tr>
<td>Heat</td>
<td>Temperature</td>
<td>Heat flow</td>
<td>Heat</td>
<td>HeatFlow1D</td>
</tr>
<tr>
<td>Chemical</td>
<td>Chemical potential</td>
<td>Particle flow</td>
<td>Particles</td>
<td>Under construction</td>
</tr>
<tr>
<td>Pneumatic</td>
<td>Pressure</td>
<td>Mass flow</td>
<td>Air</td>
<td>PneuLibLight</td>
</tr>
</tbody>
</table>

Part I
Introduction and demo example

Part II
Overview of environments

Part III
Modelica language and textual modeling

Part IV
Graphical modeling and the standard library

Modelica Standard Library

1D and 3D mechanics, analog and digital electrical circuits, electrical machines, heat transfer, fluid systems, cont., discrete, logical blocks, state machines
Other Libraries

- Bond graphs
- Magnetic
- Systems biology
- Hydraulics
- Pneumatics
- Powertrain
- Petri Nets
- Etc…

Modelica in Automotive Industry
Part I
Introduction and
demo example

Part II
Overview of
environments

Part III
Modelica language and
textual modeling

Part IV
Graphical modeling
and the standard library

Copyright © Peter Fritzson, David Broman, and Jan Brugård

davbr@ida.liu.se
jan.brugard@mathcore.com

Modelica in Avionics

Copyright © Peter Fritzson, David Broman, and Jan Brugård

davbr@ida.liu.se
jan.brugard@mathcore.com

Modelica in Power Generation
Exercise 3.1

- Draw the DCMotor model using the graphic connection editor using models from the following Modelica libraries:
  Mechanics.Rotational,
  Electrical.Analog.Basic,
  Electrical.Analog.Sources

- Simulate it for 15s and plot the variables for the outgoing rotational speed on the inertia axis and the voltage on the voltage source (denoted u in the figure) in the same plot.
Optional Exercise 3.2

- If there is enough time: Add a torsional spring to the outgoing shaft and another inertia element. Simulate again and see the results. Adjust some parameters to make a rather stiff spring.

Optional Exercise 3.3

- If there is enough time: Add a PI controller to the system and try to control the rotational speed of the outgoing shaft. Verify the result using a step signal for input. Tune the PI controller by changing its parameters in MathModelica.
Live example

- Building a component with icon

Optional Exercise 3.4

- Make a component of the model in Exercise 2.2, and use it when building the model in exercise 2.3.

Replace with component
Learn more…

- **Modelica Association**
  - www.modelica.org

- **OpenModelica**
  - www.openmodelica.org

- **MathModelica**
  - [www.mathcore.com](http://www.mathcore.com)
  - info@mathcore.com

- **Books**
  - Principles of Object Oriented Modeling and Simulation with Modelica 2.1, Peter Fritzson
  - Introducción al Modelado y Simulación de Sistemas Técnicos y Físicos con Modelica, Peter Fritzson
  - Introduction to Modelica, Michael Tiller

---

**Summary**

- **Multi-Domain Modeling**
- **Visual Acausal Component Modeling**
- **Typed Declarative Textual Language**
- **Hybrid Modeling**

Thanks for listening!
Acknowledgements and Copyrights

- If you want to use the Powerpoint version of these slides in your own course, send an email to: peter.fritzson@ida.liu.se, davbr@ida.liu.se, or jan.brugard@mathcore.com
- The slides were created by David Broman, Jan Brugård, Peter Fritzson, and Emma Larsdotter Nilsson,
- Some of the figures in this tutorial are adapted with permission from Peter Fritzson’s book “Principles of Object Oriented Modeling and Simulation with Modelica 2.1”, copyright Wiley-IEEE Press
- Some examples and figures are reproduced with permission from Modelica Association, Martin Otter, Hilding Elmqvist, and MathCore