Introduction to Object-Oriented Modeling, Simulation, Debugging and Dynamic Optimization with Modelica using OpenModelica

Tutorial, Version February 2, 2016

Peter Fritzson
Linköping University, peter.fritzson@liu.se
Director of the Open Source Modelica Consortium
Vice Chairman of Modelica Association

Bernhard Thiele, Ph.D., bernhard.thiele@liu.se
Researcher at PELAB, Linköping University

Slides
Based on book and lecture notes by Peter Fritzson
Contributions 2004-2005 by Emma Larsdotter Nilsson, Peter Bunus
Contributions 2006-2008 by Adrian Pop and Peter Fritzson
Contributions 2009 by David Broman, Peter Fritzson, Jan Brugård, and Mohsen Torabzadeh-Tari
Contributions 2010 by Peter Fritzson
Contributions 2011 by Peter F., Mohsen T., Adeel Asghar,

2016-02-02
Peter Fritzson
Principles of Object Oriented Modeling and Simulation with Modelica 3.3
A Cyber-Physical Approach

Can be ordered from Wiley or Amazon


- OpenModelica
  - www.openmodelica.org
- Modelica Association
  - www.modelica.org
Introductory Modelica Book

September 2011
232 pages

2015 – Translations available in Chinese, Japanese, Spanish

Wiley
IEEE Press

For Introductory Short Courses on Object Oriented Mathematical Modeling
Acknowledgements, Usage, 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
• Thanks to Emma Larsdotter Nilsson, Peter Bunus, David Broman, Jan Brugård, Mohsen-Torabzadeh-Tari, Adeel Asghar, Lena Buffoni, for contributions to these slides.
• Most examples and 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 reproduced with permission from Modelica Association, Martin Otter, Hilding Elmqvist, Wolfram MathCore, Siemens
• Modelica Association: www.modelica.org
• OpenModelica: www.openmodelica.org
Outline

Part I
Introduction to Modelica and a demo example

Part II
Modelica environments

Part III
Modelica language concepts and textual modeling

Part IV and Part V
Graphical modeling and the Modelica standard library
Dynamic Optimization
<table>
<thead>
<tr>
<th>Time</th>
<th>Session</th>
</tr>
</thead>
<tbody>
<tr>
<td>09:00</td>
<td><strong>Introduction to Modeling and Simulation</strong></td>
</tr>
<tr>
<td></td>
<td>- Start installation of OpenModelica including OMEdit graphic editor</td>
</tr>
<tr>
<td>09:10</td>
<td><strong>Modelica – The Next Generation Modeling Language</strong></td>
</tr>
<tr>
<td>09:25</td>
<td><strong>Exercises Part I (15 minutes)</strong></td>
</tr>
<tr>
<td></td>
<td>- Short hands-on exercise on graphical modeling using OMEdit – RL Circuit</td>
</tr>
<tr>
<td>09:50</td>
<td><strong>Part II: Modelica Environments and the OpenModelica Environment</strong></td>
</tr>
<tr>
<td>10:10</td>
<td><strong>Part III: Modelica Textual Modeling</strong></td>
</tr>
<tr>
<td>10:15</td>
<td><strong>Exercises Part IIIa (10 minutes)</strong></td>
</tr>
<tr>
<td></td>
<td>- Hands-on exercises on textual modeling using the OpenModelica environment</td>
</tr>
<tr>
<td>10:25</td>
<td><strong>Coffee Break</strong></td>
</tr>
<tr>
<td>10:40</td>
<td><strong>Modelica Discrete Events, Hybrid, Clocked Properties</strong> (Bernhard Thiele)</td>
</tr>
<tr>
<td>11:00</td>
<td><strong>Exercises Part IIIb (15 minutes)</strong></td>
</tr>
<tr>
<td></td>
<td>- Hands-on exercises on textual modeling using the OpenModelica environment</td>
</tr>
<tr>
<td>11:20</td>
<td><strong>Part IV: Components, Connectors and Connections</strong></td>
</tr>
<tr>
<td></td>
<td>- Modelica Libraries</td>
</tr>
<tr>
<td>11:30</td>
<td><strong>Part V Dynamic Optimization</strong> (Bernhard Thiele)</td>
</tr>
<tr>
<td></td>
<td>- Hands-on exercise on dynamic optimization using OpenModelica</td>
</tr>
<tr>
<td>12:00</td>
<td><strong>Exercise Graphical Modeling DCMotor using OpenModelica</strong></td>
</tr>
</tbody>
</table>
Software Installation - Windows

- Start the software installation

- Install OpenModelica-1.9.4beta.exe from the USB Stick
Software Installation – Linux (requires internet connection)

• Go to https://openmodelica.org/index.php/download/download-linux and follow the instructions.
Software Installation – MAC (requires internet connection)

• Go to
  https://openmodelica.org/index.php/download/download-mac and follow the instructions or follow the instructions written below.

• The installation uses MacPorts. After setting up a MacPorts installation, run the following commands on the terminal (as root):
  • `echo rsync://build.openmodelica.org/macports/ >> /opt/local/etc/macports/sources.conf` # assuming you installed into /opt/local
  • `port selfupdate`
  • `port install openmodelica-devel`
Part I

Introduction to Modelica and a demo example
Modelica Background: Stored Knowledge

Model knowledge is stored in books and human minds which computers cannot access.

"The change of motion is proportional to the motive force impressed."

– Newton
Modelica Background: The Form – Equations

• Equations were used in the third millennium B.C.
• Equality sign was introduced by Robert Recorde in 1557

\[
14.2e^{-1} - 15.9 = 71.9
\]

Newton still wrote text (Principia, vol. 1, 1686)
“The change of motion is proportional to the motive force impressed”

CSSL (1967) introduced a special form of “equation”:  
\[
\text{variable} = \text{expression} \\
\nu = \text{INTEG}(F)/m
\]

Programming languages usually do not allow equations!
What is Modelica?

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

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

A language for modeling of complex cyber-physical systems

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

A language for modeling of complex cyber-physical systems

i.e., Modelica is not a tool

Free, open language specification:

There exist several free and commercial tools, for example:

- OpenModelica from OSMC
- Dymola from Dassault systems
- Wolfram System Modeler fr Wolfram MathCore
- SimulationX from ITI
- MapleSim from MapleSoft
- AMEsim from LMS
- JModelica.org from Modelon
- MWORKS from Tongyang Sw & Control
- IDA Simulation Env, from Equa
- ESI Group Modeling tool, ESI Group

Available at: www.modelica.org

Developed and standardized by Modelica Association
Modelica – The Next Generation Modeling Language

Declarative language
Equations and mathematical functions allow acausal modeling, high level specification, increased correctness

Multi-domain modeling
Combine electrical, mechanical, thermodynamic, hydraulic, biological, control, event, real-time, etc...

Everything is a class
Strongly typed object-oriented language with a general class concept, Java & MATLAB-like syntax

Visual component programming
Hierarchical system architecture capabilities

Efficient, non-proprietary
Efficiency comparable to C; advanced equation compilation, e.g. 300 000 equations, ~150 000 lines on standard PC
What is *acausal* modeling/design?

Why does it increase *reuse*?

The acausality makes Modelica library classes *more reusable* than traditional classes containing assignment statements where the input-output causality is fixed.

Example: a resistor *equation*:

\[ R \cdot i = v; \]

can be used in three ways:

\[ i := v/R; \]
\[ v := R \cdot i; \]
\[ R := v/i; \]
What is Special about Modelica?

- Multi-Domain Modeling
- Visual acausal hierarchical component modeling
- Typed declarative equation-based textual language
- Hybrid modeling and simulation
What is Special about Modelica?

Multi-Domain Modeling

Cyber-Physical Modeling

Physical

3 domains
- electric
- mechanics
- control

Cyber

Reference

Electric

R
L

emf

Axis₁

Bearing

Axis₂

Angle-Sensor

PID

Control System
What is Special about Modelica?

Multi-Domain Modeling

Acausal model (Modelica)

Causal block-based model (Simulink)

Keeps the physical structure

Visual Acausal Hierarchical Component Modeling
What is Special about Modelica?

Multi-Domain Modeling

Hierarchical system modeling

Visual Acausal Hierarchical Component Modeling

Srel = n*transpose(n) + (identity(3) - n*transpose(n))*cos(q) - skew(n)*sin(q);
wrela = n*qd;
zrela = n*qdd;
Sb = Sa*transpose(Srel);
r0b = r0a;
vb = Srel*va;
wv = Srel*(wa + wrela);
ab = Srel*aa;
zv = Srel*(za + zrela + cross(wa, wrela));
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;
```

Typed Declarative Equation-based Textual Language

Visual Acausal Hierarchical Component Modeling
What is Special about Modelica?

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

Hybrid modeling = continuous-time + discrete-time modeling

- Continuous-time
- Discrete-time
- Clocked discrete-time
Modelica – Faster Development, Lower Maintenance than with Traditional Tools

Block Diagram (e.g. Simulink, ...) or Proprietary Code (e.g. Ada, Fortran, C,...) vs Modelica
Modelica vs Simulink Block Oriented Modeling
Simple Electrical Model

**Modelica:**
Physical model – easy to understand

**Simulink:**
Signal-flow model – hard to understand

**Keeps the physical structure**
Graphical Modeling - Using Drag and Drop Composition
A DC motor can be thought of as an electrical circuit which also contains an electromechanical component.

```model DCMotor
    Resistor R(R=100);
    Inductor L(L=100);
    VsourceDC DC(f=10);
    Ground G;
    ElectroMechanicalElement EM(k=10, J=10, b=2);
    Inertia load;

equation
    connect(DC.p, R.n);
    connect(R.p, L.n);
    connect(L.p, EM.n);
    connect(EM.p, DC.n);
    connect(DC.n, G.p);
    connect(EM.flange, load.flange);
end DCMotor```

Multi-Domain (Electro-Mechanical) Modelica Model
Corresponding DCMotor Model Equations

The following equations are automatically derived from the Modelica model:

\[
\begin{align*}
0 &= \text{DC.p.i} + R.n.i & \text{EM.u} &= \text{EM.p.v} - \text{EM.n.v} & R.u &= R.p.v - R.n.v \\
\text{DC.p.v} &= R.n.v & 0 &= \text{EM.p.i} + \text{EM.n.i} & 0 &= R.p.i + R.n.i \\
& & \text{EM.i} &= \text{EM.p.i} & R.i &= R.p.i \\
0 &= R.p.i + L.n.i & \text{EM.u} &= \text{EM.k} \times \text{EM.ω} & R.u &= R.R \times R.i \\
\text{R.p.v} &= L.n.v & \text{EM.i} &= \text{EM.M} / \text{EM.k} & \\
& & \text{EM.J} \times \text{EM.ω} &= \text{EM.M} - \text{EM.b} \times \text{EM.ω} & L.u &= \text{L.p.v} - L.n.v \\
0 &= L.p.i + \text{EM.n.i} & 0 &= \text{DC.p.i} + \text{DC.n.i} & 0 &= L.p.i + L.n.i \\
\text{L.p.v} &= \text{EM.n.v} & \text{DC.u} &= \text{DC.p.v} - \text{DC.n.v} & L.i &= L.p.i \\
& & 0 &= \text{DC.p.i} + \text{DC.n.i} & L.u &= L.L \times L.i' \\
0 &= \text{EM.p.i} + \text{DC.n.i} & \text{DC.i} &= \text{DC.p.i} & \\
\text{EM.p.v} &= \text{DC.n.v} & \text{DC.u} &= \text{DC.Amp} \times \text{Sin}[2 \pi \text{DC.f} \times t] & \\
0 &= \text{DC.n.i} + \text{G.p.i} & \text{DC.n.v} &= \text{G.p.v} & (\text{load component not included})
\end{align*}
\]

Automatic transformation to ODE or DAE for simulation:

\[
\frac{dx}{dt} = f[x, u, t] \quad g\left[\frac{dx}{dt}, x, u, t\right] = 0
\]
Model Translation Process to Hybrid DAE to Code

Modeling Environment

Frontend

"Middle-end"

Backend

Modelica Graphical Editor
Modelica Textual Editor

Modelica Model

Modelica Source code

Translator

Analyzer

Optimizer

Code generator

C Compiler

Simulation

Modelica Model

Flat model Hybrid DAE

Sorted equations

Optimized sorted equations

C Code

Executable
Modelica in Power Generation
GTX Gas Turbine Power Cutoff Mechanism

Developed by MathCore for Siemens

Courtesy of Siemens Industrial Turbomachinery AB
Modelica in Automotive Industry
Modelica in Avionics

Inputs

Outputs

X

Y

Z

CoG

body

g = (0,0)

earth

atmosphere

toEngine

engine

aerodynamics

referencePoint

r_{rel} = (0,0,0)

speed

d_{alpha}

d_{beta}

height

mach
Modelica in Biomechanics
Application of Modelica in Robotics Models
Real-time Training Simulator for Flight, Driving

• Using Modelica models generating real-time code
• Different simulation environments (e.g. Flight, Car Driving, Helicopter)
• Developed at DLR Munich, Germany
• Dymola Modelica tool

Courtesy of Tobias Bellmann, DLR, Oberphaffenhofen, Germany
Combined-Cycle Power Plant
Plant model – system level

- GT unit, ST unit, Drum boilers unit and HRSG units, connected by thermo-fluid ports and by signal buses.

- Low-temperature parts (condenser, feedwater system, LP circuits) are represented by trivial boundary conditions.

- GT model: simple law relating the electrical load request with the exhaust gas temperature and flow rate.

Courtesy Francesco Casella, Politecnico di Milano – Italy and Francesco Pretolani, CESI SpA - Italy
Modelica Spacecraft Dynamics Library

Formation flying on elliptical orbits

Control the relative motion of two or more spacecraft

Attitude control for satellites using magnetic coils as actuators

Torque generation mechanism: interaction between coils and geomagnetic field

Courtesy of Francesco Casella, Politecnico di Milano, Italy
System Dynamics – World Society Simulation
Limits to Material Growth; Population, Energy and Material flows

Left. World3 simulation with OpenModelica
- 2 collapse scenarios (close to current developments)
- 1 sustainable scenario (green).

CO2 Emissions per person:
- USA 17 ton/yr
- Sweden 7 ton/yr
- India 1.4 ton/yr
- Bangladesh 0.3 ton/yr

- System Dynamics Modelica library by Francois Cellier (ETH), et al in OM distribution.
- Warming converts many agriculture areas to deserts (USA, Europe, India, Amazonas)
- Ecological breakdown around 2080-2100, drastic reduction of world population
- To avoid this: Need for massive investments in sustainable technology and renewable energy sources
What Can You Do?
Need Global Sustainability Mass Movement

- Book: Current catastrophic scenarios: Mark Lynas: ”6 Degrees”
- Book: How to address the problems: Tim Jackson ”Prosperity without Growth”
- Promote sustainable lifestyle and technology
- Install electric solar PV panels
- Buy shares in cooperative wind power

20 sqm solar panels on garage roof, Nov 2012
Generated 2700 W at noon March 10, 2013

Expanded to 93 sqm, 12 kW, March 2013
House produced 11600 kwh, used 9500 kwh
Avoids 10 ton CO2 emission per year
Example Electric Cars
Can be charged by electricity from own solar panels

- Renault ZOE; 5 seat; Range:
  - EU-drive cycle 210 km
  - Realistic Swedish drive cycle:
    - Summer: 165 km
    - Winter: 100 – 110 km
- Cheap fast supercharger

- DLR ROboMObil
  - experimental electric car
  - Modelica models

- Tesla model S
  range 480 km
Small rectangles – surface needed for 100% solar energy for humanity

Good News

Year 2013 – China installed 12Gw, production 14 Twh/yr
More than doubling capacity. Germany installed 3.3 Gw
Sustainable Society Necessary for Human Survival

Almost Sustainable

- India, 1.4 ton CO2/person/year
- Healthy vegetarian food
- Small-scale agriculture
- Small-scale shops
- Simpler life-style (Mahatma Gandhi)

Non-sustainable

- USA 17 ton CO2, Sweden 7 ton CO2/yr
- High meat consumption (1 kg beef uses ca 4000 L water for production)
- Hamburgers, unhealthy, includes beef
- Energy-consuming mechanized agriculture
- Transport dependent shopping centres
- Stressful materialistic lifestyle

Gandhi – role model for future less materialistic lifestyle
Brief Modelica History

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

• Modelica Versions
  • 1.0 released September 1997
  • 2.0 released March 2002
  • 2.2 released March 2005
  • 3.0 released September 2007
  • 3.1 released May 2009
  • 3.2 released March 2010
  • 3.3 released May 2012
  • 3.2 rev 2 released November 2013
  • 3.3 rev 1 released July 2014

• Modelica Association established 2000 in Linköping
  • Open, non-profit organization
Modelica Conferences

- The 1st International Modelica conference October, 2000
- The 2nd International Modelica conference March 18-19, 2002
- The 3rd International Modelica conference November 5-6, 2003 in Linköping, Sweden
- The 4th International Modelica conference March 6-7, 2005 in Hamburg, Germany
- The 5th International Modelica conference September 4-5, 2006 in Vienna, Austria
- The 6th International Modelica conference March 3-4, 2008 in Bielefeld, Germany
- The 7th International Modelica conference Sept 21-22, 2009 in Como, Italy
- The 8th International Modelica conference March 20-22, 2011 in Dresden, Germany
- The 9th International Modelica conference Sept 3-5, 2012 in Munich, Germany
- The 10th International Modelica conference March 10-12, 2014 in Lund, Sweden
- The 11th International Modelica conference Sept 21-23, 2015 in Versailles, Paris
Exercises Part I
Hands-on graphical modeling
(15 minutes)
Exercises Part I – Basic Graphical Modeling

• (See instructions on next two pages)
• Start the OMEdit editor (part of OpenModelica)
• Draw the RLCircuit
• Simulate

The RLCircuit

Simulation
Exercises Part I – OMEdit Instructions (Part I)

- Start OMEdit from the Program menu under OpenModelica
- Go to **File** menu and choose **New**, and then select **Model**.
- E.g. write *RLCircuit* as the model name.
- For more information on how to use OMEdit, go to **Help** and choose **User Manual** or press **F1**.

• Under the **Modelica Library**:
  • Contains The standard Modelica library components
  • The **Modelica files** contains the list of models you have created.
Exercises Part I – OMEdit Instructions (Part II)

- For the RLCircuit model, browse the Modelica standard library and add the following component models:

- Make the corresponding connections between the component models as shown in slide 38.

- Simulate the model
  - Go to Simulation menu and choose simulate or click on the simulate button in the toolbar.

- Plot the instance variables
  - Once the simulation is completed, a plot variables list will appear on the right side. Select the variable that you want to plot.
Part II

Modelica environments and OpenModelica
Wolfram System Modeler – Wolfram MathCore

- Wolfram Research
- USA, Sweden
- General purpose
- Mathematica integration
- [www.wolfram.com](http://www.wolfram.com)
- [www.mathcore.com](http://www.mathcore.com)

Car model graphical view
Dymola

- Dassault Systemes Sweden
- Sweden
- First Modelica tool on the market
- Initial main focus on automotive industry
- www.dymola.com
Simulation X

- ITI GmbH (Just bought by ESI Group)
- Germany
- Mechatronic systems
- www.simulationx.com
MapleSim

- Maplesoft
- Canada
- Recent Modelica tool on the market
- Integrated with Maple
- www.maplesoft.com
The OpenModelica Environment

www.OpenModelica.org
The OpenModelica Open Source Environment
www.openmodelica.org

- Advanced Interactive Modelica compiler (OMC)
  - Supports most of the Modelica Language
  - Modelica and Python scripting
- Basic environment for creating models
  - OMShell – an interactive command handler
  - OMNotebook – a literate programming notebook
  - MDT – an advanced textual environment in Eclipse

- OMEdit graphic Editor
- OMDdebugger for equations
- OMOptim optimization tool
- OM Dynamic optimizer collocation
- ModelicaML UML Profile
- MetaModelica extension
- ParModelica extension

The OpenModelica Open Source Environment
www.openmodelica.org

Founded Dec 4, 2007

Open-source community services

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

Industrial members

- ABB AB, Sweden
- Bosch Rexroth AG, Germany
- Siemens Turbo, Sweden
- CDAC Centre, Kerala, India
- Creative Connections, Prague
- DHI, Aarhus, Denmark
- Dynamica s.r.l., Cremona, Italy
- EDF, Paris, France
- Equa Simulation AB, Sweden
- Fraunhofer IWES, Bremerhaven
- IFPEN, Paris, France
- ISID Dentsu, Tokyo, Japan
- Maplesoft, Canada
- Ricardo Inc., USA
- RTE France, Paris, France
- Saab AB, Linköping, Sweden
- Scilab Enterprises, France
- SKF, Göteborg, Sweden
- TLK Thermo, Germany
- Sozhou Tongyuan, China
- VTI, Linköping, Sweden
- VTT, Finland
- Wolfram MathCore, Sweden

Code Statistics

/trunk: Lines of Code

University members

- Austrian Inst. of Tech, Austria
- TU Berlin, Inst. UEBB, Germany
- FH Bielefeld, Bielefeld, Germany
- TU Braunschweig, Germany
- University of Calabria, Italy
- Univ California, Berkeley, USA
- Chalmers Univ Techn, Sweden
- TU Dortmund, Germany
- TU Dresden, Germany
- Université Laval, Canada
- Ghent University, Belgium
- Halmstad University, Sweden
- Heidelberg University, Germany
- Linköping University, Sweden
- TU Hamburg/Harburg Germany
- IIT Bombay, Mumbai, India
- KTH, Stockholm, Sweden
- Univ of Maryland, Syst Eng USA
- Univ of Maryland, CEEE, USA
- Politecnico di Milano, Italy
- Ecoles des Mines, CEP, France
- Mälardalen University, Sweden
- Univ Pisa, Italy
- StellenBosch Univ, South Africa
- Telemark Univ College, Norway
OMNotebook Electronic Notebook with DrModelica

- Primarily for teaching
- Interactive electronic book
- Platform independent

Commands:
- **Shift-return (evaluates a cell)**
- File Menu (open, close, etc.)
- Text Cursor (vertical), Cell cursor (horizontal)
- Cell types: text cells & executable code cells
- Copy, paste, group cells
- Copy, paste, group text
- Command Completion (shift-tab)
1 Kalman Filter

Often we don't have access to the internal states of a system and have to reconstruct the state of the system based on measurements. The idea with an observer is that we feedback the estimated state error. If the estimation is correct then the difference should be zero.

Another difficulty is that the measured quantities of the system may be evaluated by the difference

\[
\dot{x} = A\hat{x}(t) + Bu(t) + L(y(t) - x(t))
\]

Here \( e \) denoting a disturbance in the input signal.

By using this quantity as feedback we obtain the observer equation

\[
\hat{x} = A\hat{x}(t) + Bu(t)
\]

Now form the error as

\[
\hat{y} = y(t) - x(t)
\]

The differential error is

\[
\dot{\hat{y}} = \dot{y} - \dot{x}
\]
First Basic Class

1 HelloWorld

The program contains a declaration of a class called HelloWorld with two fields and one equation. The first field is the variable x which is initialized to a start value of 1 at the time when the simulation starts. The second field is the variable a, which is a constant that is initialized to 2 at the beginning of the simulation. Such a constant is prefixed by the keyword parameter in order to indicate that it is constant during simulation but is a model parameter that can be changed between simulations.

The Modelica program solves a trivial differential equation: \( x' = -a \cdot x \). The variable x is a state variable that can change value over time. The \( x' \) is the time derivative of x.

```modelica
class HelloWorld
  Real x(start = 1, fixed = true);
  parameter Real a = 1;
  equation
    der(x) = -a * x;
end HelloWorld;
```

2 Simulation of HelloWorld

```modelica
simulate( HelloWorld, startTime=0, stopTime=3 )
```

Plot the results.
```modelica
plot( x )
```

Simulation of HelloWorld

The program contains a declaration of a class called HelloWorld with two fields and one equation. The first field is the variable x which is initialized to a start value of 1 at the time when the simulation starts. The second field is the variable a, which is a constant that is initialized to 1 at the beginning of the simulation. Such a constant is prefixed by the keyword parameter in order to indicate that it is constant during simulation but is a model parameter that can be changed between simulations. The Modelica program solves a trivial differential equation: \( x' = -a \cdot x \). The variable x is a state variable that can change value over time. The \( x' \) is the time derivative of x.

```modelica
class HelloWorld
  Real x(start = 1, fixed = true);
  parameter Real a = 1;
  equation
    der(x) = -a * x;
end HelloWorld;
```

Plot the results.
```modelica
plot(x)```
OpenModelica Environment Demo
OpenModelica MDT – Eclipse Plugin

- Browsing of packages, classes, functions
- Automatic building of executables; separate compilation
- Syntax highlighting
- Code completion, Code query support for developers
- Automatic Indentation
- Debugger
  (Prel. version for algorithmic subset)
OpenModelica MDT: Code Outline and Hovering Info

- **Code Outline for easy navigation within Modelica files**
- **Identifier Info on Hovering**
OpenModelica Simulation in Web Browser Client

OpenModelica compiles to efficient Java Script code which is executed in web browser

MultiBody RobotR3.FullRobot
OMPython – Python Scripting with OpenModelica

- Interpretation of Modelica commands and expressions
- Interactive Session handling
- Library / Tool
- Optimized Parser results
- Helper functions
- Deployable, Extensible and Distributable
PySimulator Package

- PySimulator, a simulation and analysis package developed by DLR
- Free, downloadable
- Uses OMPython to simulate Modelica models by OpenModelica
Modelica3D Library

- Modelica 3D Graphics Library by Fraunhofer FIRST, Berlin
- Part of OpenModelica distribution
- Can be used for 3D graphics in OpenModelica
Extending Modelica with PDEs for 2D, 3D flow problems – Research

Prototype in OpenModelica 2005
PhD Thesis by Levon Saldamli
www.openmodelica.org
Currently not operational
Failure Mode and Effects Analysis (FMEA) in OM

- Modelica models augmented with reliability properties can be used to generate reliability models in Figaro, which in turn can be used for static reliability analysis.
- Prototype in OpenModelica integrated with Figaro tool (which is becoming open-source).
OMOptim – Optimization (1)

Model structure

Model Variables

Optimized parameters

Optimized Objectives

MinEIT

Variables

Optimized variables

Scanned variables

Optimization objectives
OMOptim – Optimization (2)

Solved problems

Result plot

Export result data .csv
Multiple-Shooting and Collocation
Dynamic Trajectory Optimization

- Minimize a goal function subject to model equation constraints, useful e.g. for NMPC
- Multiple Shooting/Collocation
  - Solve sub-problem in each sub-interval

\[ x_i(t_{i+1}) = h_i + \int_{t_i}^{t_{i+1}} f(x_i(t), u(t), t) \, dt \approx F(t_i, t_{i+1}, h_i, u_i), \quad x_i(t_i) = h_i \]

Example speedup, 16 cores:
OpenModelica Dynamic Optimization Collocation

- **DAE**
- **ODE**

**Cost function**

**Weight sum of the cost**

**Constraints**

**Residual equations**

**Collocation technique**

**Discrete NLP**

**OpenModelica**

- **Discrete goal function**
- **Discrete constraint function**
- **Gradient**
- **Jacobian**
- **Hessian**
General Tool Interoperability & Model Exchange Functional Mock-up Interface (FMI)

- FMI development was started by ITEA2 MODELISAR project. FMI is a Modelica Association Project now
- **Version 1.0**
  - FMI for Model Exchange (released Jan 26, 2010)
  - FMI for Co-Simulation (released Oct 12, 2010)
- **Version 2.0**
  - FMI for Model Exchange and Co-Simulation (released July 25, 2014)
  - > 60 tools supporting it (https://www.fmi-standard.org/tools)

functional mockup interface for model exchange and tool coupling
courtesy Daimler
**Functional Mockup Units**

- Import and export of input/output blocks – **Functional Mock-Up Units** – FMUs, described by
  - differential-, algebraic-, discrete equations,
  - with time-, state, and step-events
- An FMU can be large (e.g. 100 000 variables)
- An FMU can be used in an embedded system (small overhead)
- FMUs can be connected together
OpenModelica Functional Mockup Interface (FMI)

FMI Export

1. Modelica Code
2. OpenModelica Compiler
3. Code Generation
4. FMU

FMI Import

1. FMU
2. OpenModelica Compiler
3. Code Generation
4. Modelica Code

Translator, Analyzer & Optimizer
Model Description, DLL & FMI interface functions

FMU parsing, reading states & events
FMI in OpenModelica

- Model Exchange implemented (FMI 1.0 and FMI 2.0)
- FMI 2.0 Co-simulation available
- The FMI interface is accessible via the **OpenModelica scripting environment** and the **OpenModelica connection editor**
OPENPROD – Large 28-partner European Project, 2009-2012
Vision of Cyber-Physical Model-Based Product Development

OPENPROD Vision of unified modeling framework for model-based product development.

Open Standards – Modelica (HW, SW) and UML (SW)
OPENPROD Model-Based Development Environment
Covers Product-Design V

Business Process Control -> Requirements Capture -> ModelDriven Design -> Compilation & Code Gen

Requirements models -> Process models
Product models
Platform models

Unified Modeling: Metamodeling & Modelica & UML

System Simulation
Software & System Product

Level of Abstraction

Experience Feedback

System requirements
Preliminary feature design
Architectural design and system functional design
Detailed feature design and implementation

Verification
Integration
Component verification
Subsystem level integration and verification
Subsystem level integration test calibration and verification
Product verification and deployment
Maintenance
Calibration

Documentation, Version and Configuration Management
Business Process Control and Modeling

OpenModelica based simulation

Metso Business model & simulation

VTT Simantics Graphic Modeling Tool

Simulation of 3 strategies with outcomes
Requirement Capture

Business Process Control → Requirements Capture → Model Driven Design → Compilation & Code Gen → System Simulation

Process models → Requirements models → Product models → Platform models

Unified Modeling: Meta modeling & Modelica & UML

vVDR (virtual Verification of Designs against Requirements)
in ModelicaML UML/Modelica Profile, part of OpenModelica

OpenModelica based simulation

Verification Model → Design Model → Scenario Model → Requirement Models

Client from requirement model

Provider from design model

Bindings:
- \(\text{tankIsEmpty} = \text{sm_spws\_environment.spws\_tank\_level} < 0.001\)
- \(\text{tankIsFull} = \text{sm_spws\_environment.spws\_tank\_level} > 0.98\)
- Real timeLimit = 300
- output violated
- output evaluationStarted
- \_reqVerificationVerdict
OpenModelica – ModelicaML UML Profile
SysML/UML to Modelica OMG Standardization

• ModelicaML is a UML Profile for SW/HW modeling
  • Applicable to “pure” UML or to other UML profiles, e.g. SysML

• Standardized Mapping UML/SysML to Modelica
  • Defines transformation/mapping for **executable** models
  • Being **standardized** by OMG

• ModelicaML
  • Defines graphical concrete syntax (graphical notation for diagram) for representing Modelica constructs integrated with UML
  • Includes graphical formalisms (e.g. State Machines, Activities, Requirements)
    • Which do not exist in Modelica language
    • Which are translated into executable Modelica code
  • Is defined towards generation of executable Modelica code
  • Current implementation based on the Papyrus UML tool + OpenModelica
Example: Simulation and Requirements Evaluation

«model»
(TwoTanksSystemExample::SystemSimulations)
TankSystemSimulation

«component» dm: TanksConnectedPI
«requirementInstance» r001_tank1: Max level of liquid in a tank
«requirementInstance» r001_tank2: Max level of liquid in a tank
«requirementInstance» r002_tank1: Volume of the tank1

Req. 001 is instantiated 2 times (there are 2 tanks in the system)

tank-height is 0.6m

Req. 001 for the tank2 is violated

Req. 001 for the tank1 is not violated
Goal: Enable on-demand verification of designs against requirements using automated model composition at any time during development.
Industrial Product with OEM Usage of OpenModelica

- Includes a large part of the OpenModelica compiler using the OSMC OEM license.
- Images show a house heating application and an excavator dynamics simulation.
• ABB OPTIMAX® provides advanced model based control products for power generation and water utilities

• ABB: “ABB uses several compatible Modelica tools, including OpenModelica, depending on specific application needs.”

• ABB: “OpenModelica provides outstanding debugging features that help to save a lot of time during model development.”
Performance Profiling
(Below: Profiling all equations in MSL 3.2.1 DoublePendulum)

- Measuring performance of equation blocks to find bottlenecks
  - Useful as input before model simplification for real-time platforms
- Integrated with the debugger so it is possible to show what the slow equations compute
- Suitable for real-time profiling (less information), or a complete view of all equation blocks and function calls
OpenModelica MDT Algorithmic Code Debugger

List of Stack Frames

Variables View

Output View

Copyright © Open Source Modelica Consortium
The OpenModelica MDT Debugger (Eclipse-based)
Using Japanese Characters
Mapping run-time error to source model position
Debugging Example – Detecting Source of Chattering (excessive event switching) causing bad performance

Equation:
\[
z = \text{if } x > 0 \text{ then } -1 \text{ else } 1;
\]
\[
y = 2 \times z;
\]
Error Indication – Simulation Slows Down


OMEdit - Debugging.Chattering.ChatteringEvents1 Simulation Output

```
/tmp/OpenModelica/OMEdit/Debugging.Chattering.ChatteringEvents1 -port=50212 -logFormat=xml -w -lv=LOG_STATS
stdout | info | Chattering detected around time 0.500000005..0.500000995001 (100 state events in a row with a total time delta less than the step size 0.002). This can be a performance bottleneck. Use -lv LOG_EVENTS for more information. The zero-crossing was: x > 0.0 Debug more
```
Exercise 1.2 – Equation-based Model Debugger

In the model ChatteringEvents1, chattering takes place after $t = 0.5$, due to the discontinuity in the right hand side of the first equation. Chattering can be detected because lots of tightly spaced events are generated. The debugger allows to identify the (faulty) equation that gives rise to all the zero crossing events.

```model ChatteringNoEvents1
  Real x(start=1, fixed=true);
  Real y;
  Real z;
  equation
    z = noEvent(if x > 0 then -1 else 1);
    y = 2*z;
    der(x) = y;
  end ChatteringNoEvents1;
```

- Switch to OMEdit text view (click on text button upper left)
- Open the Debugging.mo package file using OMEdit
- Open subpackage Chattering, then open model ChatteringNoEvents1
- Simulate in debug mode
- Click on the button Debug more (see prev. slide)
- Possibly start task manager and look at CPU. Then click stop simulation button

Uses 25% CPU
Part III

Modelica language concepts and textual modeling

```
record ColorData
  parameter Real red = 0.2;
  parameter Real blue = 0.6;
  Real green;
end ColorData;

class Color
  extends ColorData;
  equation
  red + blue + green = 1;
end Color;

class ExpandedColor
  parameter Real red=0.2;
  parameter Real blue=0.6;
  Real green;
  equation
  red + blue + green = 1;
end ExpandedColor;
```
Acausal Modeling

The order of computations is not decided at modeling time

<table>
<thead>
<tr>
<th>Visual Component Level</th>
<th>Acausal</th>
<th>Causal</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Acausal Diagram" /></td>
<td><img src="image2.png" alt="Causal Diagram" /></td>
<td></td>
</tr>
</tbody>
</table>

Equation Level

A resistor *equation*:

\[ R \times i = v; \]

Causal possibilities:

\[ i := v/R; \]
\[ v := R \times i; \]
\[ R := v/i; \]
Typical Simulation Process

“Static” semantics / compile time

Modelica model

Elaboration

Hybrid DAE

Equation Transformation & Code generation

“Dynamic” semantics / run time

Executable

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)
```
Modelica Variables and Constants

- **Built-in primitive data types**
  - **Boolean**
    - true or false
  - **Integer**
    - Integer value, e.g. 42 or -3
  - **Real**
    - Floating point value, e.g. 2.4e-6
  - **String**
    - String, e.g. “Hello world”
  - **Enumeration**
    - Enumeration literal e.g. ShirtSize.Medium

- **Parameters are constant during simulation**

- **Two types of constants in Modelica**
  - **constant**
  - **parameter**

```modelica
constant Real PI = 3.141592653589793;
constant String redcolor = "red";
constant Integer one = 1;
parameter Real mass = 22.5;
```
A Simple Rocket Model

\[ \text{acceleration} = \frac{\text{thrust} - \text{mass} \cdot \text{gravity}}{\text{mass}} \]

\[ \text{mass}' = -\text{massLossRate} \cdot \text{abs}(\text{thrust}) \]

\[ \text{altitude}' = \text{velocity} \]

\[ \text{velocity}' = \text{acceleration} \]

class Rocket "rocket_class"

parameter String name;
Real mass(start=1038.358);
Real altitude(start=59404);
Real velocity(start=-2003);
Real acceleration;
Real thrust; // Thrust force on rocket
Real gravity; // Gravity forcefield
parameter Real massLossRate=0.000277;

equation

\[(\text{thrust} - \text{mass} \cdot \text{gravity})/\text{mass} = \text{acceleration};\]
\[\text{der}(\text{mass}) = -\text{massLossRate} \cdot \text{abs}(\text{thrust});\]
\[\text{der}(\text{altitude}) = \text{velocity};\]
\[\text{der}(\text{velocity}) = \text{acceleration};\]
end Rocket;
Celestial Body Class

A class declaration creates a *type name* in Modelica

```modelica
class CelestialBody
  constant Real    g = 6.672e-11;
  parameter Real   radius;
  parameter String name;
  parameter Real   mass;
end CelestialBody;
```

An *instance* of the class can be declared by *prefixing* the type name to a variable name

```modelica
... CelestialBody moon;
...```

The declaration states that *moon* is a variable containing an object of type *CelestialBody*
class MoonLanding
    parameter Real force1 = 36350;
    parameter Real force2 = 1308;
    protected
        parameter Real thrustEndTime = 210;
        parameter Real thrustDecreaseTime = 43.2;
    public
        Rocket apollo(name="apollo13");
        CelestialBody moon(name="moon", mass=7.382e22, radius=1.738e6);
    equation
        apollo.thrust = if (time < thrustDecreaseTime) then force1
                       else if (time < thrustEndTime) then force2
                       else 0;
        apollo.gravity = moon.g * moon.mass / (apollo.altitude + moon.radius)^2;
end MoonLanding;

\[
\text{apollo.gravity} = \frac{\text{moon.g} \cdot \text{moon.mass}}{(\text{apollo.altitude} + \text{moon.radius})^2}
\]
Simulation of Moon Landing

```plaintext
simulate(MoonLanding, stopTime=230)
plot(apollo.altitude, xrange={0,208})
plot(apollo.velocity, xrange={0,208})
```

It starts at an altitude of 59404 (not shown in the diagram) at time zero, gradually reducing it until touchdown at the lunar surface when the altitude is zero.

The rocket initially has a high negative velocity when approaching the lunar surface. This is reduced to zero at touchdown, giving a smooth landing.
Specialized Class Keywords

• Classes can also be declared with other keywords, e.g.: `model`, `record`, `block`, `connector`, `function`, ...

• Classes declared with such keywords have specialized properties

• Restrictions and enhancements apply to contents of specialized classes

• After Modelica 3.0 the `class` keyword means the same as `model`

• Example: (Modelica 2.2). A `model` is a class that cannot be used as a connector class

• Example: A `record` is a class that only contains data, with no equations

• Example: A `block` is a class with fixed input-output causality

```model CelestialBody
    constant Real g = 6.672e-11;
    parameter Real radius;
    parameter String name;
    parameter Real mass;
end CelestialBody;
```
Modelica Functions

• Modelica Functions can be viewed as a specialized class with some restrictions and extensions
• A function can be called with arguments, and is instantiated dynamically when called

```modelica
function sum
  input Real arg1;
  input Real arg2;
  output Real result;
  algorithm
    result := arg1+arg2;
end sum;
```
Example Modelica function call:

```
... p = polynomialEvaluator({1,2,3,4},21)
```

The function `polynomialEvaluator` computes the value of a polynomial given two arguments: a coefficient vector `A` and a value of `x`.

```
function PolynomialEvaluator

input Real A[:]; // array, size defined
// at function call time
input Real x := 1.0; // default value 1.0 for x
output Real sum;
protected
Real xpower; // local variable xpower
algorithm
sum := 0;
xpower := 1;
for i in 1:size(A,1) loop
    sum := sum + A[i]*xpower;
xpower := xpower*x;
end for;
end PolynomialEvaluator;
```
Inheritance

Data and behavior: field declarations, equations, and certain other contents are *copied* into the subclass
Multiple Inheritance is fine – inheriting both geometry and color

class Color
    parameter Real red=0.2;
    parameter Real blue=0.6;
    Real green;
end Color;

class Point
    Real x;
    Real y,z;
end Point;

class ColoredPointWithoutInheritance
    Real x;
    Real y, z;
    parameter Real red = 0.2;
    parameter Real blue = 0.6;
    Real green;
end ColoredPointWithoutInheritance;

class ColoredPoint
    extends Point;
    extends Color;
end ColoredPoint;

Equivalent to
Multiple Inheritance cont’

Only one copy of multiply inherited class `Point` is kept

```modelica
class Point
  Real x;
  Real y;
end Point;
```

```modelica
class VerticalLine
  extends Point;
  Real vlength;
end VerticalLine;
```

```modelica
class HorizontalLine
  extends Point;
  Real hlength;
end HorizontalLine;
```

```modelica
class Rectangle
  extends VerticalLine;
  extends HorizontalLine;
end Rectangle;
```

Diamond Inheritance
Simple Class Definition

- Simple Class Definition
  - Shorthand Case of Inheritance
- Example:

```modelica
class SameColor = Color;
end SameColor;
```

Equivalent to:

```modelica
type Resistor = Real;
connector MyPin = Pin;
class SameColor extends Color;
end SameColor;
```

- Often used for introducing new names of types:
Inheritance Through Modification

• Modification is a concise way of combining inheritance with declaration of classes or instances.

• A modifier modifies a declaration equation in the inherited class.

• Example: The class `Real` is inherited, modified with a different `start` value equation, and instantiated as an altitude variable:

```modelica
Real altitude(start= 59404); 
```

...
model Body "generic body"
Real mass;
String name;
end Body;

model CelestialBody
extends Body;
constant Real g = 6.672e-11;
parameter Real radius;
end CelestialBody;

model Rocket "generic rocket class"
extends Body;
  parameter Real massLossRate=0.000277;
  Real altitude(start= 59404);
  Real velocity(start= -2003);
  Real acceleration;
  Real thrust;
  Real gravity;
equation
  thrust-mass*gravity= mass*acceleration;
  der(mass)= -massLossRate*abs(thrust);
  der(altitude)= velocity;
  der(velocity)= acceleration;
end Rocket;
The Moon Landing - Example using Inheritance (II)

```model MoonLanding
  parameter Real force1 = 36350;
  parameter Real force2 = 1308;
  parameter Real thrustEndTime = 210;
  parameter Real thrustDecreaseTime = 43.2;
  Rocket apollo(name="apollo13", mass(start=1038.358) );
  CelestialBody moon(mass=7.382e22, radius=1.738e6, name="moon");
  equation
    apollo.thrust = if (time<thrustDecreaseTime) then force1
                    else if (time<thrustEndTime) then force2
                    else 0;
    apollo.gravity = moon.g*moon.mass/(apollo.altitude+moon.radius)^2;
end Landing;
```
Inheritance of Protected Elements

If an `extends`-clause is preceded by the `protected` keyword, all inherited elements from the superclass become protected elements of the subclass.

The inherited fields from `Point` keep their protection status since that `extends`-clause is preceded by `public`.

A protected element cannot be accessed via dot notation!
Exercises Part III a
(15 minutes)
Exercises Part III a

• Start OMNotebook (part of OpenModelica)
  • Start->Programs->OpenModelica->OMNotebook
  • Open File: Exercises-ModelicaTutorial.onb from the directory you copied your tutorial files to.
  • Note: The DrModelica electronic book has been automatically opened when you started OMNotebook.

• Open Exercises-ModelicaTutorial.pdf (also available in printed handouts)
Exercises 2.1 and 2.2 (See also next two pages)

• Open the Exercises-ModelicaTutorial.onb found in the Tutorial directory you copied at installation.

• Exercise 2.1. Simulate and plot the HelloWorld example. Do a slight change in the model, re-simulate and re-plot. Try command-completion, val( ), etc.

```modelica
class HelloWorld "A simple equation"
  Real x(start=1);
equation
  der(x) = -x;
end HelloWorld;
```

• Locate the VanDerPol model in DrModelica (link from Section 2.1), using OMNotebook!

• (extra) Exercise 2.2: Simulate and plot VanDerPol. Do a slight change in the model, re-simulate and re-plot.
Exercise 2.1 – Hello World!

A Modelica “Hello World” model

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

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

Simulation in OpenModelica environment

```modelica
simulate(HelloWorld, stopTime = 2)
plot(x)
```
class VanDerPol  "Van der Pol oscillator model"
  Real x(start = 1)  "Descriptive string for x"; // x starts at 1
  Real y(start = 1)  "y coordinate";             // y starts at 1
  parameter Real lambda = 0.3;

equation
  der(x) = y;                                    // This is the 1st diff equation //
  der(y) = -x + lambda*(1 - x*x)*y; /* This is the 2nd diff equation */
end VanDerPol;

simulate(VanDerPol,stopTime = 25)
plotParametric(x,y)
(extra) Exercise 2.3 – DAE Example

Include algebraic equation

Algebraic equations contain no derivatives

Exercise: Locate in DrModelica. Simulate and plot. Change the model, simulate+plot.

Simulation in OpenModelica environment

class DAEexample
    Real x(start=0.9);
    Real y;
    equation
        der(y)+(1+0.5*sin(y))*der(x) = sin(time);
        x - y = exp(-0.9*x)*cos(y);
    end DAEexample;

simulate(DAEexample, stopTime = 1)
plot(x)
Exercise 2.4 – Model the system below

- Model this Simple System of Equations in Modelica

\[
\begin{align*}
\dot{x} &= 2 \times x \times y - 3 \times x \\
\dot{y} &= 5 \times y - 7 \times x \times y \\
x(0) &= 2 \\
y(0) &= 3
\end{align*}
\]
(extra) Exercise 2.5 – Functions

- a) Write a function, `sum2`, which calculates the sum of Real numbers, for a vector of arbitrary size.

- b) Write a function, `average`, which calculates the average of Real numbers, in a vector of arbitrary size. The function `average` should make use of a function call to `sum2`. 
Part III b
Discrete Events and Hybrid Systems

Picture: Courtesy Hilding Elmqvist
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*

```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**

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

---

**Time event**

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

Only dependent on time, can be scheduled in advance

**State event**

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

Related to a state. Check for zero-crossing

---

Equations only active at event times
Generating Repeated Events

The call \texttt{sample(t0,d)} returns true and triggers events at times \(t0+i\times d\), where \(i=0,1,...\)

\begin{verbatim}
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;
\end{verbatim}

Variables need to be discrete

Creates an event after 2 s, then each 0.5 s

\texttt{pre(...)} takes the previous value before the event.
Reinit - Discontinuous Changes

The value of a \textit{continuous-time} state variable can be instantaneously changed by a \texttt{reinit}-equation within a \texttt{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;
```

Reinit "assigns" continuous-time variable \texttt{velocity} a new value
Application: Digital Control Systems

- Discrete-time controller + continuous-time plant = hybrid system or sampled-data system
- Typically periodic sampling, can be modeled with “when sample(t0, td) then ...”
Sampled Data-Systems in Modelica

// time-discrete controller
when {initial(), sample(3, 3)} then
    E*xd = A*pre(xd) + B*y;
    ud = C*pre(xd) + D*y;
end when;

// plant (continuous-time process)
0 = f(der(x), x, ud);
y = g(x);

• y is automatically sampled at t = 3, 6, 9,…;
• xd, u are piecewise-constant variables that change values at sampling events (implicit zero-order hold)
• initial() triggers event at initialization (t=0)
Exercise 2.6 – BouncingBall

• Locate the BouncingBall model in one of the hybrid modeling sections of DrModelica (the When-Equations link in Section 2.9), run it, change it slightly, and re-run it.
Part IIIc “Technology Preview”

Clocked Synchronous Models and State Machines
Clocked Synchronous Extension in Modelica 3.3

// time-discrete controller
when Clock() then
    E*xd = A*previous(xd) + B*yd;
    ud = C*previous(xd) + D*yd;
end when;

// hold
u = hold(ud)

// plant
Ω = f(der(x), x, u);
y = g(x);

// sample continuous signal
yd = sample(y, Clock(3));

\[ t_i \in \{0, 3, 6, \ldots \}, \quad i = 0, 1, 2, \ldots \]  
\[ yd(t_i) = y(t_i) \]  
\[ E \cdot xd(t_i) = A \cdot xd(t_{i-1}) + B \cdot yd(t_i) \]  
\[ ud(t_i) = C \cdot xd(t_{i-1}) + D \cdot yd(t_i) \]  
\[ u(t) = ud(t_i), \quad t_i \leq t < t_{i+1} \]

Clocked variables \( r(t_i) \) are of base type Real, Integer, etc. They are uniquely associated with a clock \( c(t_i) \). Can only be accessed when its clock is active (ticks).

Clock variables \( c(t_i) \) are of base type Clock. They are defined by constructors such as \( \text{Clock}(3) \) or by clock operators relatively to other clocks.

Continuous variables are Real numbers defined as piecewise continuous functions of time.

Piecewise-constant variables \( m(t) \) are constant inside each
\[ t_i \leq t < t_{i+1} \]
State Machines in Modelica 3.3: Simple Example

- Equations are active if corresponding clock ticks. Defaults to periodic clock with 1.0 s sampling period
- “i” is a shared variable, “j” is a local variable. Transitions are “delayed” and enter states by “reset”
model Simple_NoAnnotations "Simple state machine"
    inner Integer i(start=0);
    block State1
        outer output Integer i;
        output Integer j(start=10);
    equation
        i = previous(i) + 2;
        j = previous(j) - 1;
    end State1;
State1 state1;
block State2
    outer output Integer i;
    equation
        i = previous(i) - 1;
    end State2;
State2 state2;

equation
    transition(state1,state2,i > 10,immediate=false);
    transition(state2,state1,i < 1,immediate=false);
    initialState(state1);
end Simple_NoAnnotations;
Technology Preview

• The clocked synchronous language extension not yet ready in OpenModelica (under development)
  • However some simple models can be simulated.

• No graphical editing support for state machine in OMEdit, yet.

• Full state machine extension requires that clocked synchronous support is available

• However, many state machines can already be simulated
  • By using a workaround that restricts the sampling period of a state machine to a fixed default value of 1s.
The OMNotebook ebook “SynchronousAndStateMachinePreview.onb” provides one example featuring clocked synchronous language elements and two state machine examples.

Open this and simulate. (If there is time)
Part IV

Components, Connectors and Connections – Modelica Libraries and Graphical Modeling
A component class should be defined *independently of the environment*, very essential for *reusability*

A component may internally consist of other components, i.e. *hierarchical* modeling

Complex systems usually consist of large numbers of *connected* components
Connectors and Connector Classes

Connectors are instances of **connector classes**

- **electrical connector**
  - connector class
  - keyword `flow` indicates that currents of connected pins sum to zero.

- **mechanical connector**
  - connector class
  - an instance `flange` of class `Flange`
Two kinds of variables in connectors:

- *Non-flow variables* potential or energy level
- *Flow variables* represent some kind of flow

**Coupling**

- *Equality coupling*, for non-*flow* variables
- *Sum-to-zero coupling*, for *flow* variables

The value of a *flow* variable is *positive* when the current or the flow is *into* the component.

[Diagram of a pin with positive flow direction]
## Physical Connector

- **Classes Based on Energy Flow**

<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>
Connections between connectors are realized as *equations* in Modelica.

**Connectors**

The two arguments of a **connect-equation** must be references to **connectors**, either to be declared directly *within* the *same class* or be *members* of one of the declared variables in that class.

Pin pin1,pin2;
//A connect equation
//in Modelica:
connect(pin1,pin2);

Corresponds to

pin1.v = pin2.v;
pin1.i + pin2.i = 0;
Connection Equations

Pin pin1,pin2;
//A connect equation
//in Modelica
connect(pin1,pin2);

Corresponds to
pin1.v = pin2.v;
pin1.i + pin2.i = 0;

Multiple connections are possible:
connect(pin1,pin2); connect(pin1,pin3); ... connect(pin1,pinN);

Each primitive connection set of nonflow variables is used to generate equations of the form:

\[ v_1 = v_2 = v_3 = \ldots v_n \]

Each primitive connection set of flow variables is used to generate sum-to-zero equations of the form:

\[ i_1 + i_2 + \ldots (-i_k) + \ldots i_n = 0 \]
Common Component Structure

The base class **TwoPin** has two connectors **p** and **n** for positive and negative pins respectively.

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

// TwoPin is same as OnePort in
// Modelica.Electrical.Analog.Interfaces

---

**partial class** (cannot be instantiated)

**positive pin**

**negative pin**
**Electrical Components**

```model Resistor "Ideal electrical resistor"
extends TwoPin;
  parameter Real R;
equation
  R*i = v;
end Resistor;
```

```model Inductor "Ideal electrical inductor"
extends TwoPin;
  parameter Real L "Inductance";
equation
  L*der(i) = v;
end Inductor;
```

```model Capacitor "Ideal electrical capacitor"
extends TwoPin;
  parameter Real C ;
equation
  i=C*der(v);
end Capacitor;
```
model Source
  extends TwoPin;
  parameter Real A, w;
  equation
    v = A*sin(w*time);
end Resistor;

model Ground
  Pin p;
  equation
    p.v = 0;
end Ground;
Resistor Circuit

\[
\begin{align*}
R1.p.v &= R2.p.v; \\
R1.p.v &= R3.p.v; \\
R1.p.i + R2.p.i + R3.p.i &= 0;
\end{align*}
\]

model ResistorCircuit
    Resistor R1(R=100);
    Resistor R2(R=200);
    Resistor R3(R=300);

end ResistorCircuit;
Modelica Standard Library - Graphical Modeling

- **Modelica Standard Library** (called Modelica) is a standardized predefined package developed by Modelica Association

- It can be used freely for both commercial and noncommercial purposes under the conditions of *The Modelica License*.

- Modelica libraries are available online including documentation and source code from [http://www.modelica.org/library/library.html](http://www.modelica.org/library/library.html)
The Modelica Standard Library contains components from various application areas, including the following sublibraries:

- **Blocks**: Library for basic input/output control blocks
- **Constants**: Mathematical constants and constants of nature
- **Electrical**: Library for electrical models
- **Icons**: Icon definitions
- **Fluid**: 1-dim Flow in networks of vessels, pipes, fluid machines, valves, etc.
- **Math**: Mathematical functions
- **Magnetic**: Magnetic.Fluxtubes – for magnetic applications
- **Mechanics**: Library for mechanical systems
- **Media**: Media models for liquids and gases
- **SIunits**: Type definitions based on SI units according to ISO 31-1992
- **Stategraph**: Hierarchical state machines (analogous to Statecharts)
- **Thermal**: Components for thermal systems
- **Utilities**: Utility functions especially for scripting
Modelica.Blocks

Continuous, discrete, and logical input/output blocks to build block diagrams.

Examples:
Electrical components for building analog, digital, and multiphase circuits

Examples:
Modelica.Mechanics

Package containing components for mechanical systems

Subpackages:

- Rotational 1-dimensional rotational mechanical components
- Translational 1-dimensional translational mechanical components
- MultiBody 3-dimensional mechanical components
Modelica.Stategraph

Hierarchical state machines (similar to Statecharts)
## Other Free Libraries

<table>
<thead>
<tr>
<th>Library</th>
<th>Description</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>WasteWater</td>
<td>Wastewater treatment plants, 2003</td>
<td></td>
</tr>
<tr>
<td>ATPlus</td>
<td>Building simulation and control (fuzzy control included), 2005</td>
<td></td>
</tr>
<tr>
<td>MotorCycleDynamics</td>
<td>Dynamics and control of motorcycles, 2009</td>
<td></td>
</tr>
<tr>
<td>NeuralNetwork</td>
<td>Neural network mathematical models, 2006</td>
<td></td>
</tr>
<tr>
<td>VehicleDynamics</td>
<td>Dynamics of vehicle chassis (obsolete), 2003</td>
<td></td>
</tr>
<tr>
<td>SPICElib</td>
<td>Some capabilities of electric circuit simulator PSPICE, 2003</td>
<td></td>
</tr>
<tr>
<td>SystemDynamics</td>
<td>System dynamics modeling a la J. Forrester, 2007</td>
<td></td>
</tr>
<tr>
<td>BondLib</td>
<td>Bond graph modeling of physical systems, 2007</td>
<td></td>
</tr>
<tr>
<td>MultiBondLib</td>
<td>Multi bond graph modeling of physical systems, 2007</td>
<td></td>
</tr>
<tr>
<td>ModelicaDEVS</td>
<td>DEVS discrete event modeling, 2006</td>
<td></td>
</tr>
<tr>
<td>ExtendedPetriNets</td>
<td>Petri net modeling, 2002</td>
<td></td>
</tr>
<tr>
<td>External.Media Library</td>
<td>External fluid property computation, 2008</td>
<td></td>
</tr>
<tr>
<td>VirtualLabBuilder</td>
<td>Implementation of virtual labs, 2007</td>
<td></td>
</tr>
<tr>
<td>SPOT</td>
<td>Power systems in transient and steady-state mode, 2007</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Some Commercial Libraries

- Powertrain
- SmartElectricDrives
- VehicleDynamics
- AirConditioning
- HyLib
- PneuLib
- CombiPlant
- HydroPlant
- …
Connecting Components from Multiple Domains

- Block domain
- Mechanical domain
- Electrical domain

model Generator
Modelica.Mechanics.Rotational.Inertia iner;
Modelica.Electrical.Analog.Basic.EMF emf(k=-1);
Modelica.Electrical.Analog.Basic.Inductor ind(L=0.1);
Modelica.Electrical.Analog.Basic.Resistor R1,R2;
Modelica.Blocks.Sources.Exponentials ex(riseTime={2},riseTimeConst={1});
equation
    connect(ac.flange_b, iner.flange_a); connect(iner.flange_b, emf.flange_b);
    connect(emf.p, ind.p); connect(ind.n, R1.p); connect(emf.n, G.p);
    connect(emf.n, R2.n); connect(R1.n, R2.p); connect(R2.p, vsens.n);
    connect(R2.n, vsens.p); connect(ex.outPort, ac.inPort);
end Generator;
A DC motor can be thought of as an electrical circuit which also contains an electromechanical component.

```model DCMotor
    Resistor R(R=100);
    Inductor L(L=100);
    VsourceDC DC(f=10);
    Ground G;
    EMF emf(k=10, J=10, b=2);
    Inertia load;

equation
    connect(DC.p, R.n);
    connect(R.p, L.n);
    connect(L.p, emf.n);
    connect(emf.p, DC.n);
    connect(DC.n, G.p);
    connect(emf.flange, load.flange);
end DCMotor;
```
Part V
Dynamic Optimization
Theory and Exercises

using
OpenModelica
Built-in Dynamic Optimization - Motivation

Simulation

Inputs (known)  →  Simulation  →  Output (result)

Optimization – Try to find the inputs that result in a desired output

Inputs (result)  →  Simulation  →  Output (desired)
Optimization of Dynamic Trajectories Using Multiple-Shooting and Collocation

- Minimize a goal function subject to model equation constraints, useful e.g. for NMPC
- Multiple Shooting/Collocation
  - Solve sub-problem in each sub-interval

\[
x_i(t_{i+1}) = h_i + \int_{t_i}^{t_{i+1}} f(x_i(t), u(t), t) \, dt \approx F(t_i, t_{i+1}, h_i, u_i),
\]

This approach uses a single optimization run and is different from classical parameter sweep optimization typically using a large number of simulations

\[
x_i(t_i) = h_i
\]

**Example speedup, 16 cores:**

MULTIPLE_COLLOCATION
Optimal Control Problem (OCP)

Cost function
\[
\min_{u(t)} J(x(t), u(t), t) = E(x(t_f), u(t_f), t_f) + \int_{t_0}^{t_f} L(x(t), u(t), t) \, dt
\]  
\[\text{Mayer–Term} \]
\[\text{Lagrange–Term}\]  

Subject to

Initial conditions
\[x(t_0) = x_0\]  

Nonlinear dynamic model
\[\dot{x} = f(x(t), u(t), t)\]  

Path constraints
\[g(x(t), u(t), t) \leq 0\]  

Terminal constraints
\[r(x(t_f)) = 0\]  

where
\[x(t) = [x^1(t), ..., x^{nx}]^T\] is the state vector and
\[u(t) = [u^1(t), ..., u^{nu}(t)]^T\] is the control variable vector for
\[t \in [t_0, t_f]\] respectively.
OCP Formulation in OpenModelica

The path constraints $\hat{g}(x(t), u(t), t) \leq 0$ can be split into box constraints

$$x_{\text{min}} \leq x(t) \leq x_{\text{max}}$$
$$u_{\text{min}} \leq u(t) \leq u_{\text{max}}$$

Variable attributes \textit{min and max} are reused for describing constraints, annotations are used for specifying the OCP

<table>
<thead>
<tr>
<th>Annotation</th>
<th>Real (\text{costM} \ annotation(\text{isMayer=true});)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mayer-Term</td>
<td>\textit{Real costL annotation(isLagrange=true);}</td>
</tr>
<tr>
<td>Lagrange-Term</td>
<td>\textit{Real x(max=0) annotation(isConstraint=true);}</td>
</tr>
<tr>
<td>Constraints</td>
<td>\textit{Real y(min=0) annotation(isFinalConstraint=true);}</td>
</tr>
</tbody>
</table>
Predator-Prey Example – The Forest Model

Dynamic model of a forest with foxes $x_f$, rabbits $x_r$, fox hunters $u_{hf}$ and rabbit hunters $u_{hr}$ (adapted from Vitalij Ruge, “Native Optimization Features in OpenModelica”, part of the OpenModelica documentation)

$$\dot{x}_r = g_r \cdot x_r - d_{rf} \cdot x_r \cdot x_f - d_{rh} \cdot u_{hr}$$

$$\dot{x}_f = g_{fr} \cdot d_{rf} \cdot x_r \cdot x_f - d_f \cdot x_f - d_{fh} \cdot u_{hf}$$

IC: $x_r(t_0) = 700$, $x_f(t_0) = 10$

where

$g_r = 4 \cdot 10^{-2}$, Natural growth rate for rabbits

$g_{fr} = 1 \cdot 10^{-1}$, Efficiency in growing foxes from rabbits

$d_{rf} = 5 \cdot 10^{-3}$, Death rate of rabbits due to foxes

$d_{rh} = 5 \cdot 10^{-3}$, Death rate of rabbits due to hunters

$d_f = 9 \cdot 10^{-2}$, Natural death rate for foxes

$d_{fh} = 9 \cdot 10^{-2}$, Death rate of foxes due to hunters
model Forest "Predator-prey model"

  parameter Real g_r = 4e-2 "Natural growth rate for rabbits";
  parameter Real g_fr = 1e-1 "Efficiency in growing foxes from rabbits";
  parameter Real d_rf = 5e-3 "Death rate of rabbits due to foxes";
  parameter Real d_rh = 5e-2 "Death rate of rabbits due to hunters";
  parameter Real d_f = 9e-2 "Natural deathrate for foxes";
  parameter Real d_fh = 9e-2 "Death rate of foxes due to hunters";

  Real x_r(start=700,fixed=true) "Rabbits with start population of 700";
  Real x_f(start=10,fixed=true) "Foxes with start population of 10";

  input Real u_hr "Rabbit hunters";
  input Real u_hf "Fox hunters";

  equation
    der(x_r) = g_r*x_r - d_rf*x_r*x_f - d_rh*u_hr;
    der(x_f) = g_fr*d_rf*x_r*x_f - d_f*x_f - d_fh*u_hf;

end Forest;
Predator-Prey Example – Optimal Control Problem

Objective: Regulate the population in the forest to a desired level (5 foxes, 500 rabbits) at the end of the simulation ($t = t_f$)

$$J_{\text{Mayer}} = 0.1 \cdot (x_f(t_f) - 5)^2 + 0.01 \cdot (x_r(t_f) - 500)^2$$

(desired population at $t = t_f$)

Constraints: $u_{hf} \geq 0$, $u_{hf} \geq 0$, $x_r \geq 0$, $x_f \geq 0$

Modelica model:

```modelica
model ForestOCP;
    extends Forest(
        u_hr(min=0, nominal=1e-4), u_hf(min=0, nominal=1e-4),
        x_r(min=0), x_f(min=0));
    Real J_Mayer =
        0.1*(x_r - 5)^2 + 0.01*(x_r - 500)^2 annotation(isMayer=true);
end ForestOCP;
```

**Extension of the system model**

**Important for scaling, needs to be > 0 to make optimizer converge!**

**Cost function Mayer-term**
Start the optimization from OMNotebook using a time interval \([t_0, t_f] = [0,400]\) seconds

```plaintext
setCommandLineOptions("+gDynOpt");
optimize(ForestOCP, stopTime=400, tolerance=1e-8, numberOfIntervals=50, simflags="-s optimization");
```

<table>
<thead>
<tr>
<th>Option</th>
<th>Example value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>numberOfIntervals</td>
<td>50</td>
<td>collocation intervals</td>
</tr>
<tr>
<td>startTime, stopTime</td>
<td>0, 400</td>
<td>time horizon in seconds</td>
</tr>
<tr>
<td>tolerance</td>
<td>1e-8</td>
<td>solver/optimizer tolerance</td>
</tr>
<tr>
<td>simflags</td>
<td>…</td>
<td>see documentation for details</td>
</tr>
</tbody>
</table>
Predator-Prey Example – Using OMEdit

Tools → Options → Simulation

Simulation → Simulation Setup

+gDynOpt

optimization
Predator-Prey Example – Plots

Simulation of the forest model with control variables $u_{hr} = u_{hf} = 0$

Simulation of the forest model using the control variables computed by the optimization. Notice (not well visible in the plot) that

$$x_r(t_f) = 500, \quad x_f(t_f) = 5$$
Exercise – Optimal Control

Load the OPCExample.onb ebook into OMNotebook and modify the optimization problem in the following ways:

1. Constrain the maximal number of rabbit hunters and fox hunters to five, respectively.
2. Change the Mayer-term of the cost function to a Lagrange-term.
3. Penalize the number of employed hunters by a suitable modification of the cost function and observe how the solution changes for different modifications.
Part Vb
More
Graphical Modeling Exercises

using
OpenModelica
Graphical Modeling - Using Drag and Drop Composition
Graphical Modeling Animation – DCMotor
A DC motor can be thought of as an electrical circuit which also contains an electromechanical component.

```model DCMotor
    Resistor R(R=100);
    Inductor L(L=100);
    VsourceDC DC(f=10);
    Ground G;
    ElectroMechanicalElement EM(k=10,J=10, b=2);
    Inertia load;
    equation
        connect(DC.p,R.n);
        connect(R.p,L.n);
        connect(L.p, EM.n);
        connect(EM.p, DC.n);
        connect(DC.n,G.p);
        connect(EM.flange,load.flange);
end DCMotor```

Multi-Domain (Electro-Mechanical) Modelica Model
The following equations are automatically derived from the Modelica model:

\[
\begin{align*}
0 &= \text{DC} \cdot \text{p} \cdot \text{i} + \text{R} \cdot \text{n} \cdot \text{i} \\
\text{DC} \cdot \text{p} \cdot \text{v} &= \text{R} \cdot \text{n} \cdot \text{v} \\
0 &= \text{EM} \cdot \text{p} \cdot \text{i} + \text{EM} \cdot \text{n} \cdot \text{i} \\
\text{EM} \cdot \text{i} &= \text{EM} \cdot \text{p} \cdot \text{i} \\
0 &= \text{R} \cdot \text{p} \cdot \text{i} + \text{L} \cdot \text{n} \cdot \text{i} \\
\text{R} \cdot \text{p} \cdot \text{v} &= \text{L} \cdot \text{n} \cdot \text{v} \\
0 &= \text{EM} \cdot \text{J} \star \text{EM} \cdot \omega = \text{EM} \cdot \text{M} - \text{EM} \cdot \text{b} \star \text{EM} \cdot \omega \\
\text{L} \cdot \text{u} &= \text{L} \cdot \text{p} \cdot \text{v} - \text{L} \cdot \text{n} \cdot \text{v} \\
0 &= \text{L} \cdot \text{p} \cdot \text{i} + \text{L} \cdot \text{n} \cdot \text{i} \\
\text{L} \cdot \text{p} \cdot \text{v} &= \text{EM} \cdot \text{n} \cdot \text{v} \\
\text{DC} \cdot \text{u} &= \text{DC} \cdot \text{p} \cdot \text{v} - \text{DC} \cdot \text{n} \cdot \text{v} \\
0 &= \text{DC} \cdot \text{p} \cdot \text{i} + \text{DC} \cdot \text{n} \cdot \text{i} \\
0 &= \text{EM} \cdot \text{p} \cdot \text{i} + \text{DC} \cdot \text{n} \cdot \text{i} \\
\text{EM} \cdot \text{p} \cdot \text{v} &= \text{DC} \cdot \text{n} \cdot \text{v} \\
\text{DC} \cdot \text{u} &= \text{DC} \cdot \text{Amp} \star \text{Sin}[2 \pi \text{DC} \cdot \text{f} \star t] \\
0 &= \text{DC} \cdot \text{n} \cdot \text{i} + \text{G} \cdot \text{p} \cdot \text{i} \\
\text{DC} \cdot \text{n} \cdot \text{v} &= \text{G} \cdot \text{p} \cdot \text{v}
\end{align*}
\]

(\text{load component not included})

Automatic transformation to ODE or DAE for simulation:

\[
\frac{dx}{dt} = f[x, u, t] \\
g \left[ \frac{dx}{dt}, x, u, t \right] = 0
\]
Exercise 3.1

• Draw the DCMotor model using the graphic connection editor using models from the following Modelica libraries:

• 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.
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.
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 OMEdit.
Exercise 3.4 – DrControl

- If there is enough time: Open the DrControl electronic book about control theory with Modelica and do some exercises.
  - Open File: C:\OpenModelica1.9.3\share\omnotebook\drcontrol\DrControl.onb
Learn more...

- **OpenModelica**
  - [www.openmodelica.org](http://www.openmodelica.org)

- **Modelica Association**
  - [www.modelica.org](http://www.modelica.org)

- **Books**
  - Introduction to Modelica, Michael Tiller
Summary

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

Thanks for listening!