MOSES 2016 Workshop

Tutorial, Version May 18, 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 Larsson Nilsson, Peter Bune
Contributions 2006-2008 by Ashok Pug 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., Adeeel Aghhar,
Contributions 2012, 2013, 2014, 2015, 2016 by Peter Fritzson,
Lena Buffoni, Mehdi Ghassemehdi, Bernhard Thiele

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

MOSES 2016 Workshop

Tutorial, Version May 18, 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 Larsson Nilsson, Peter Bune
Contributions 2006-2008 by Ashok Pug 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., Adeeel Aghhar,
Contributions 2012, 2013, 2014, 2015, 2016 by Peter Fritzson,
Lena Buffoni, Mehdi Ghassemehdi, Bernhard Thiele

Modelica/OpenModelica Tutorial Plan for the
MOSES 2016 Workshop

• Wednesday slot 24. The Modelica language part 1.
  Introductory hands-on Modelica modeling.
  Slides 8 – 49
• Thursday slot 28. The OpenModelica tool. (slides
  50 – 78) The Modelica language part 2. (slides 95-
  115)
• Thursday slot 29. Hands-on with Modelica textual
  equation-based modeling (slides 95–115)

Tutorial Based on Book, Decembr 2014
Download OpenModelica Software

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

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 Larssdot Nilsson, Peter Bunus, David
  Broman, Jan Brugård, Mohsen-Torabzadeh-Tari, Adeel
  Aghhar, 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

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
**Detailed Schedule (morning version) 09.00-12.30**

09:00 - Introduction to Modeling and Simulation
- Start installation of OpenModelica including OMEdit graphic editor

09:10 - Modelica – The Next Generation Modeling Language

09:25 - Exercises Part I (15 minutes)
- Short hands-on exercise on graphical modeling using OMEdit – RL Circuit

09:50 – Part II: Modelica Environments and the OpenModelica Environment
10:10 – Part III: Modelica Textual Modeling
10:15 – Exercises Part IIIa (10 minutes)
- Hands-on exercises on textual modeling using the OpenModelica environment
10:25 – Coffee Break
10:40 - Modelica Discrete Events, Hybrid, Clocked Properties (Bernhard Thiele)
11:00 - Exercises Part IIIb (15 minutes)
- Hands-on exercises on textual modeling using the OpenModelica environment

11:20 – Part IV: Components, Connectors and Connections
- Modelica Libraries
11:30 – Part V Dynamic Optimization (Bernhard Thiele)
- Hands-on exercise on dynamic optimization using OpenModelica
12:00 – Exercise Graphical Modeling DCMotor using OpenModelica

---

**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](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](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`

---

**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

\[ 14.2 \cdot \frac{3}{2} - 15.9 = -1.7 \cdot 9.8 \]

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":
\[ \text{variable} = \text{expression} \]
\[ v = \text{INTEG}(F)/m \]

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

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

Multi-Domain Modeling

Acausal model (Modelica)

Causal block-based model (Simulink)

Visual Acausal Hierarchical Component Modeling

Keeps the physical structure

Differential equations

Variable declarations

Tipo de Declarativo

Equación de Textual

Typed Declarative Equation-based Textual Language

Hierarchical system modeling

Continuous-time

Discrete-time

Clocked discrete-time

Hybrid Modeling

Continuous-time

Time
**Modelica – Faster Development, Lower Maintenance than with Traditional Tools**

<table>
<thead>
<tr>
<th>Systems Definition</th>
<th>Modeling of Subsystems</th>
<th>Causality Derivation</th>
<th>Implementation</th>
<th>Simulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proprietary Code</td>
<td>Block Diagram</td>
<td>Modelica</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Block Diagram (e.g. Simulink, ...) or Proprietary Code (e.g. Ada, Fortran, C,...) vs Modelica

---

**Graphical Modeling - Using Drag and Drop Composition**

![Graphical Modeling](image)

---

**Multi-Domain (Electro-Mechanical) Modelica Model**

- 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```

---

**Corresponding DCMotor Model Equations**

The following equations are automatically derived from the Modelica model:

- **Resistor**
  
  \[ R \]

- **Inductor**
  
  \[ L \]

- **VsourceDC**
  
  \[ DC \]

- **Ground**
  
  \[ G \]

- **ElectroMechanicalElement**
  
  \[ EM \]

- **Inertia load**
  
  \[ load \]

**Model Translation Process to Hybrid DAE to Code**

- **Modelica Model**
  
  Flattened model

- **Modelica Source code**
  
  Sorted equations

- **Modelica Textual Editor**
  
  Optimized sorted equations

- **Modelica Graphical Editor**
  
  C Code

- **Modeling Environment**
  
  Executable

Frontend

- **Middle-end**

- **Backend**

Automatic transformation to ODE or DAE for simulation:

\[
\frac{dx}{dt} = f(x, u, t) \quad \Rightarrow \quad \frac{dx}{dt} = 0
\]
Modelica in Power Generation

GTX Gas Turbine Power Cutoff Mechanism

Courtesy of Siemens Industrial Turbomachinery AB
Developed by MathCore for Siemens

Modelica in Automotive Industry

Modelica in Avionics

Inputs

Outputs

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

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

Torque generation mechanism: interaction between coils and geomagnetic field

Attitude control for satellites using magnetic coils as actuators

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

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

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

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

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:
  • Add SineVoltage component model from Modelica.Electrical.Analog.Sources package.
• 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
  • www.mathcore.com

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
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

Interactive Simulation

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

```modelica
class PDEModel;
  heatTransfer h_iso;
  heatTransfer h_heated(g=50);
  heatTransfer h_glass(h_heat=30000);
  heatTransfer h_ht;
  equation
    dom.eq=ht;
    dom.left.bc=h_glass;
    dom.right.hc热播=dom.
    dom.top.bc=h_ht;
    dom.bottom.bc=h_heated;
  model PDEModel;
end PDEModel;
```

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)

OMOptim – Optimization (2)
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
  \[ n(t_{k+1}) = n_k + \int_{t_k}^{t_{k+1}} f(t, x(t), u(t)) \, dt \approx P(t_{k+1}, n_k, u_k), \]
  \[ x(t_k) = n_k \]

Example speedup, 16 cores: MULTIPLE_COLLOCATION

---

OpenModelica Dynamic Optimization Collocation

- DAE
- Cost function
- Constraints
- Weight sum of the residual equations
- Collocation technique
- Discrete technique
- Gradient
- 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)

---

OpenModelica Functional Mockup Interface (FMI)

- 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

---

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

---

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 Model-Based Development Environment
Covers Product-Design V

Business Process Control and Modeling

Requirement Capture

OpenModelica – ModelicaML UML Profile
SysML/UML to Modelica OMG Standardization

Example: Simulation and Requirements Evaluation
**vVDR Method – virtual Verification of Designs vs Requirements**

<table>
<thead>
<tr>
<th>Actor</th>
<th>Task</th>
<th>Created Artifact</th>
</tr>
</thead>
<tbody>
<tr>
<td>Formalize</td>
<td>Requirements</td>
<td>Requirement Monitor Models</td>
</tr>
<tr>
<td>Formalize</td>
<td>Designs</td>
<td>Design Alternative Models</td>
</tr>
<tr>
<td>Formalize</td>
<td>Scenarios</td>
<td>Scenario Model</td>
</tr>
<tr>
<td>AUTOMATED</td>
<td>Create Verification Models</td>
<td>Verification Models</td>
</tr>
<tr>
<td>AUTOMATED</td>
<td>Execute and Create Report</td>
<td>Reports</td>
</tr>
<tr>
<td>AUTOMATED</td>
<td>Analyze</td>
<td></td>
</tr>
</tbody>
</table>

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 Industry Use of OpenModelica FMI 2.0 and Debugger**

- 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**

**The OpenModelica MDT Debugger (Eclipse-based) Using Japanese Characters**
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><strong>Equation Level</strong></td>
</tr>
<tr>
<td></td>
<td>A resistor equation: $R \cdot i = v$;</td>
</tr>
<tr>
<td>(excessive event switching) causing bad performance</td>
<td></td>
</tr>
<tr>
<td>Uses 25% CPU</td>
<td></td>
</tr>
</tbody>
</table>

Debugging Example – Detecting Source of Chattering

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.

Exercise 1.2 – Equation-based Model Debugger

- Switch to OMEditor text view (click on text button upper left)
- Open the Debugging.mo package file using OMEditor
- Open subpackage Chattering, then open model ChatteringEvents1
- 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
Typical Simulation Process

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

  constant Real PI=3.141592653589793;
  constant String redcolor = "red";
  parameter Real one = 1;
  parameter Real mass = 22.5;

Modelica Simple - Hello World!

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

Equation: \[ \frac{dx}{dt} = ax \]
Initial condition: \( x(0) = 1 \)

Simulation in OpenModelica environment

Simulation in OpenModelica environment

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

Name of model  Initial condition  Continuous-time variable Parameter, constant during simulation

Celestial Body Class

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

A class declaration creates a type name in Modelica

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

The declaration states that moon is a variable containing an object of type CelestialBody

Moon Landing

```
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
    (thrust-mass*gravity)/mass = acceleration;
    der(mass)  = -massLossRate * abs(thrust);
    der(altitude) = velocity;
    der(velocity) = acceleration;
end Rocket;
```

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

A class declaration creates a type name in Modelica

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

The declaration states that moon is a variable containing an object of type CelestialBody

Moon Landing

```
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
    (thrust-mass*gravity)/mass = acceleration;
    der(mass)  = -massLossRate * abs(thrust);
    der(altitude) = velocity;
    der(velocity) = acceleration;
end Rocket;
```

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

A class declaration creates a type name in Modelica

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

The declaration states that moon is a variable containing an object of type CelestialBody

Moon Landing

```
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;
```
### Inheritance

- Parent class: `Color`
- Restricted kind of class without equations
- Child class: `ColorData`
- Keyword: extends
- Inheritance: subclass

Data and behavior: field declarations, equations, and certain other contents are copied into the subclass.

### Function Call – Example Function with for-loop

Example Modelica function call:

```modelica
function PolynomialEvaluator
  input Real A[:]; // array, size defined at function call time
  output Real sum; // default value 0.0 for a constant output
protected
  Real xpower; // local variable xpower
algorithm
  sum := 0;
  for xpower := 1 to A.length do
    sum := sum + A[i] * xpower;
  end for;
end PolynomialEvaluator;
```

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

### Multiple Inheritance

Multiple Inheritance is fine – inheriting both geometry and color

```modelica
class Color
  parameter Real red = 0.2;
  parameter Real green = 0.8;
end Color;
```

```modelica
class Point
  Real x, y, z;
  model CelestialBody
    constant Real g = 6.672e-11;
    parameter Real radius;
    parameter String name;
    parameter Real mass;
    parameter String label;
  end CelestialBody;
end Point;
```

```modelica
class ColoredPoint
  extends Color;
  extends Point;
end ColoredPoint;
```

```modelica
class ColoredPointWithoutInheritance
  extends ColorData;
end ColoredPointWithoutInheritance;
```

Equivalent to:

```modelica
class ColoredPointWithoutInheritance
  extends Color;
end ColoredPointWithoutInheritance;
```

### 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;
```

### 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

### Simulation of Moon Landing

It starts at an altitude of 59404 (not shown in the diagram) at time zero, gradually reducing it until touchdown at the lunar surface. This is reduced to zero at touchdown, giving a smooth landing...
Multiple Inheritance cont'

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

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

Diamond Inheritance

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

class HorizontalLine
  extends Point;
  Real hlength;
end HorizontalLine;

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

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:

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

The Moon Landing - Example using Inheritance (I)

```
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 MoonLanding;
```

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.

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.

Exercise 2.1 – Hello World!

A Modelica “Hello World” model

Equation: \( x' = -x \)

Initial condition: \( x(0) = 1 \)

Simulation in OpenModelica environment

(Extra) Exercise 2.2 – Van der Pol Oscillator

Include algebraic equation

Algebraic equations contain no derivatives

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

(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.
Exercise 2.4 – Model the system below

- Model this Simple System of Equations in Modelica
  \[
  \begin{align*}
  x &= 2 + xy - 3y \\
  y &= 5 + y - 7x + y \\
  x(0) &= 2 \\
  y(0) &= 3
  \end{align*}
  \]

(extra) Exercise 2.5 – Functions

- a) Write a function, \( \text{sum2} \), which calculates the sum of Real numbers, for a vector of arbitrary size.
- b) Write a function, \( \text{average} \), which calculates the average of Real numbers, in a vector of arbitrary size. The function \( \text{average} \) should make use of a function call to \( \text{sum2} \).

Part III b
Discrete Events and Hybrid Systems

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

- \( \text{if-equation choosing equation for v} \)
- \( \text{if-expression} \)

Event Creation – when

- \( \text{when-equations} \)
- \( \text{Time event} \)
- \( \text{State event} \)
Generating Repeated Events

The call \( \text{sample}(t_0, d) \) returns true and triggers events at times \( t_0 + i \cdot d \), where \( i = 0, 1, \ldots \).

Variables need to be discrete.

Creates an event after 2 s, then each 0.5 s.

\( \text{pre}(\ldots) \) takes the previous value before the event.

Reinit - Discontinuous Changes

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

Application: Digital Control Systems

- Discrete-time controller + continuous-time plant = hybrid system or sampled-data system
- Typically periodic sampling, can be modeled with \texttt{when sample}(t0,td) \texttt{then} ..."

Samples Data-Systems in Modelica

- \( y \) is automatically sampled at \( t = 3, 6, 9, \ldots \);
- \( x_d, u \) are piecewise-constant variables that change values at sampling events (implicit zero-order hold)
- \texttt{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 \texttt{When-Equations} link in Section 2.9), run it, change it slightly, and re-run it.

Part IIC “Technology Preview”

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

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".

Simple Example: Modelica Code

```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;
ii = previous(j(start=10)) - 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;
```

Hierarchical and Parallel Composition

Semantics of Modelica state machines (and example above) inspired by Florence Maraninchi & Yann Rémond’s “Mode-Automata” and by Marc Pouzet’s Lucid Synchron 3.0.

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.

Preview Clocked Synchronous and State Machines

- 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

Connectors and Connector Classes

Connectors are instances of connector classes

The flow prefix

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

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, Ling, Mechanical, Thermal</td>
</tr>
<tr>
<td>Translational</td>
<td>Position</td>
<td>Force</td>
<td>Linear momentum</td>
<td>Mechanical, Thermal</td>
</tr>
<tr>
<td>Rotational</td>
<td>Angle</td>
<td>Torque</td>
<td>Angular momentum</td>
<td>Rotational</td>
</tr>
<tr>
<td>Magnetic</td>
<td>Magnetic potential</td>
<td>Magnetic flux rate</td>
<td>Magnetic flux</td>
<td>Magnetic</td>
</tr>
<tr>
<td>Hydraulic</td>
<td>Pressure</td>
<td>Volume flow</td>
<td>Volume</td>
<td>Hydraulic</td>
</tr>
<tr>
<td>Heat</td>
<td>Temperature</td>
<td>Heat flow</td>
<td>Heat</td>
<td>Heat</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>Pneumatic</td>
</tr>
</tbody>
</table>

Connections between connectors are realized as equations in Modelica

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
connect(pin1, pin2);

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

Pin pin1,pin2;
//A connect equation
connect(pin1,pin2); connect(pin1,pin3); ... connect(pin1,pinN);

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:

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

Corresponds to

\[ \text{pin1}.v = \text{pin2}.v; \quad \text{pin1}.i + \text{pin2}.i = 0 \]

Common Component Structure

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

```
partial class TwoPin
  Voltage v;
  flow Current i;
end TwoPin;
```

Electrical Components

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

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

- **Capacitor**
  ```
  model Capacitor "Ideal electrical capacitor"
    extends TwoPin;
    parameter Real C;
  equation
    i=C*der(v);
  end Capacitor;
  ```

- **Source**
  ```
  model Source extends TwoPin;
    parameter Real A,w;
  equation
    v = A*sin(w*time);
  end Source;
  ```

- **Ground**
  ```
  model Ground
    Pin p;
  equation
    p.v = 0;
  end Ground;
  ```

Resistor Circuit

```
Resistor Circuit
Resistor R1(R=100);
Resistor R2(R=200);
Resistor R3(R=300);

equation
connect(R1.p, R2.p);
connect(R1.p, R3.p);
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
Modelica Standard Library cont'

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
- Sl units 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:

Modelica.Electrical

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

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

- Powertrain
- SmartElectricDrives
- VehicleDynamics
- AirConditioning
- HyLib
- PneuLib
- CombiPlant
- HydroPlant
- ...  

Connecting Components from Multiple Domains

- Block domain
- Mechanical domain
- Electrical domain

DCMotor Model Multi-Domain (Electro-Mechanical)

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;
end DCMotor;

Part V
Dynamic Optimization
Theory and Exercises
using OpenModelica

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

Example speedup, 16 cores:
Optimal Control Problem (OCP)

Cost function

\[ \min_{u(t)} f(x(t), u(t), t) = L(x(t), u(t), t) + \int_{t_0}^{t_f} T(x(t), u(t), t) \, dt \]

Subject to

Initial conditions

\[ x(t_0) = x_0 \]

Nonlinear dynamic model

\[ x = f(x(t), u(t), t) \]

Path constraints

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

Terminal constraints

\[ r(x(t), t) = 0 \]

where

\[ x(t) = [x_1(t), \ldots, x_n(t)] \] is the state vector and

\[ u(t) = [u_1(t), \ldots, u_m(t)] \] is the control variable vector for

\[ t \in [t_0, t_f] \] respectively.

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)

\[ \begin{align*}
x_f &= g_f \cdot x_f - d_{rf} \cdot x_f \cdot x_r - d_{fr} \cdot x_f \\
x_r &= g_r \cdot x_r - d_{rf} \cdot x_r \cdot x_f - d_{rh} \cdot u_{hr} \\
\end{align*} \]

where

\[ g_f = 4 \times 10^{-2}, \] Natural growth rate for rabbits

\[ d_{fr} = 5 \times 10^{-3}, \] Death rate of rabbits due to foxes

\[ d_{rh} = 5 \times 10^{-2}, \] Death rate of rabbits due to hunters

\[ d_{rf} = 5 \times 10^{-3}, \] Death rate of foxes due to rabbits

\[ g_r = 1 \times 10^{-1}, \] Efficiency in growing foxes from rabbits

\[ d_{fr} = 9 \times 10^{-4}, \] Natural deathrate for foxes

\[ d_{rh} = 9 \times 10^{-2}, \] Death rate of foxes due to hunters

\[ d_{rf} = 9 \times 10^{-3}, \] Death rate of foxes due to foxes

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

\[ J_{\text{Mayer}} = 0.01 \cdot (x_r(t_f) - 5)^2 + 0.01 \cdot (x_f(t_f) - 500)^2 \] (desired population at \( t = t_f \))

Constraints:

\[ u_{hr} \geq 0, u_{hf} \geq 0, x_r \geq 0, x_f \geq 0 \]

Modelica model:

```modelica
model ForestOCP
extends Forest;
annotation(isMayer=true);
Real J_Mayer = 0.1*(x_r-5)^2 + 0.01*(x_f-500)^2
end ForestOCP;
```

OCP Formulation in OpenModelica

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

\[ \begin{align*}
x_{\text{min}} &\leq x(t) \leq x_{\text{max}} \\
u_{\text{min}} &\leq u(t) \leq u_{\text{max}}
\end{align*} \]

Variable attributes \( \min \) and \( \max \) are reused for describing constraints, annotations are used for specifying the OCP

<table>
<thead>
<tr>
<th>Annotation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MayerTerm</td>
<td>Real cost annotation(isMayer=true);</td>
</tr>
<tr>
<td>LagrangeTerm</td>
<td>Real cost annotation(isLagrange=true);</td>
</tr>
<tr>
<td>Constraints</td>
<td>Real cost annotation(isConstraint=true);</td>
</tr>
<tr>
<td>Final constraints</td>
<td>Real y(min=0) annotation(isFinalConstraint=true);</td>
</tr>
</tbody>
</table>

Predator-Prey Example – Modelica model

```modelica
model Forest "Predator-prey model"
parameter Real g_r = 1e-1 "Efficiency in growing foxes from rabbits";
parameter Real d_rf = 5e-3 "Death rate of rabbits due to foxes";
parameter Real d_f = 9e-2 "Natural deathrate for foxes";
parameter Real d_rh = 5e-2 "Death rate of rabbits due to hunters";
parameter Real d_fh = 9e-2 "Death rate of foxes due to hunters";
Real x_f(start=10,fixed=true) "Foxes with start population of 10";
Real x_r(start=700,fixed=true) "Rabbits with start population of 700";
input Real u_hr "Rabbit hunters";
input Real u_hf "Fox hunters";
equation
\[ \begin{align*}
\dot{x}_r &= g_r \cdot x_r - d_{rf} \cdot x_r \cdot x_f - d_{hr} \cdot u_hr \\
\dot{x}_f &= g_f \cdot x_f - d_{rf} \cdot x_f \cdot x_r - d_{fr} \cdot x_f \\
\end{align*} \]
end Forest;
```

Predator-Prey Example – Using OMNotebook

Start the optimization from OMNotebook using a time interval \([t_0, t_f] = [0, 400] \) seconds

```modelica
setCommandLineOptions("+gDynOpt");
optimize(ForwardOCP, stepTime=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>startTol, stopTol</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>&quot;s&quot;</td>
<td>see documentation for details</td>
</tr>
</tbody>
</table>
Exercise – Optimal Control

Load the OPCEexample.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
Multi-Domain (Electro-Mechanical) Modelica Model

- 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
```

Corresponding DCMotor Model Equations

The following equations are automatically derived from the Modelica model:

```
0 \rightarrow \text{DC.p} + \text{R.n};
0 \rightarrow \text{DC.p} + \text{R.n};
0 \rightarrow \text{DC.p} + \text{R.n};
0 \rightarrow \text{DC.p} + \text{R.n};
0 \rightarrow \text{DC.p} + \text{R.n};
0 \rightarrow \text{DC.p} + \text{R.n};
0 \rightarrow \text{DC.p} + \text{R.n};
0 \rightarrow \text{DC.p} + \text{R.n};
0 \rightarrow \text{DC.p} + \text{R.n};
0 \rightarrow \text{DC.p} + \text{R.n};
0 \rightarrow \text{DC.p} + \text{R.n};
0 \rightarrow \text{DC.p} + \text{R.n};
0 \rightarrow \text{DC.p} + \text{R.n};
0 \rightarrow \text{DC.p} + \text{R.n};
0 \rightarrow \text{DC.p} + \text{R.n};
```

Automatic transformation to ODE or DAE for simulation:

```
\frac{dx}{dt} = f(x, u, t), \quad \frac{du}{dt} = 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
- Modelica Association
  - www.modelica.org
- Books
  - Modeling and Simulation of Technical and Physical Systems with Modelica, Peter Fritzson, 2011
  - Introduction to Modelica, Michael Tiller

Summary

Multi-Domain Modeling

Visual Acausal Component Modeling

Typed

Declarative

Textual Language

Thanks for listening!