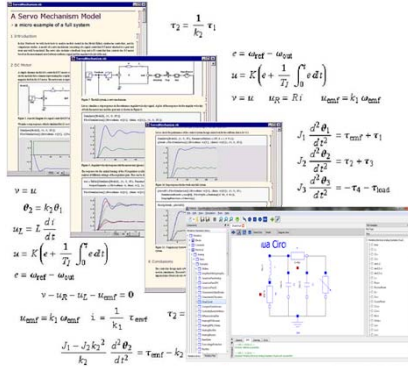


Introduction to Object-Oriented Modeling, Simulation and Control with Modelica



Tutorial, February 2012, at MODPROD by

Peter Fritzon

Linköping University, peter.fritzon@liu.se

Olena Rogovchenko

Linköping University, olena.rogovchenko@liu.se

Slides

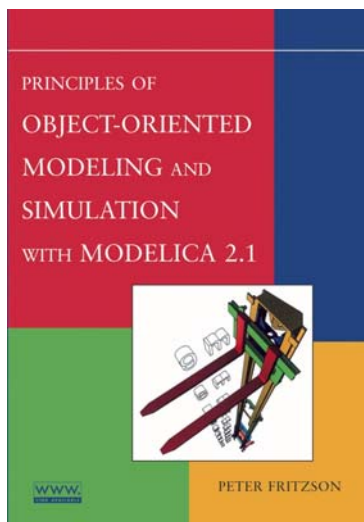
Based on book and lecture notes by Peter Fritzon
 Contributions 2004-2005 by Emma Larsdotter Nilsson, Peter Bunus
 Contributions 2006-2008 by Adrian Pop and Peter Fritzon
 Contributions 2009 by David Broman, Peter Fritzon, Jan Brugård,
 and Mohsen Torabzadeh-Tari
 Contributions 2010 by Peter Fritzon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



2012-02-07



Tutorial Based on Book, 2004 Download OpenModelica Software



Peter Fritzon

Principles of Object Oriented Modeling and Simulation with Modelica 2.1

Wiley-IEEE Press, 2004, 940 pages

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



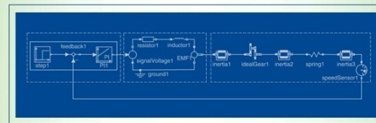
New Introductory Modelica Book

September 2011
232 pages

Wiley
IEEE Press

For Introductory
Short Courses on
Object Oriented
Mathematical Modeling

*Introduction to
Modeling and Simulation
of Technical and
Physical Systems
with Modelica*



PETER FRITZSON

WILEY

IEEE
IEEE PRESS

3 Copyright © Open Source Modelica Consortium

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 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 Elmquist, and MathCore
- Modelica Association: www.modelica.org
- OpenModelica: www.openmodelica.org

4 Copyright © Open Source Modelica Consortium

MODELICA

Outline

Part I

Introduction to Modelica and a demo example



Part II

Modelica environments



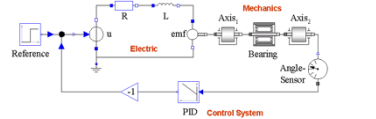
Part III

Modelica language concepts and textual modeling



Part IV

Graphical modeling and the Modelica standard library



Detailed Schedule

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 (30 minutes)*

- Hands-on exercises on textual modeling using the **OpenModelica** environment

10:45 – Coffee Break

11:00 - Modelica Discrete Events and Hybrid Properties

11:15 - *Exercises Part IIIb (10 minutes)*

- Hands-on exercises on textual modeling using the **OpenModelica** environment

11:25 – Part IV: Components, Connectors and Connections

- Modelica Libraries

11:45 - Graphical Modeling using OpenModelica

12:00 - *Exercises Part IV (30 minutes)* – *DCMotor etc.*

- Hands-on exercises on graphical modeling using **OpenModelica**

Software Installation - Windows

- Start the software installation
- Install OpenModelica-1.8.0.msi from the USB Stick

Software Installation – Linux (requires internet connection)

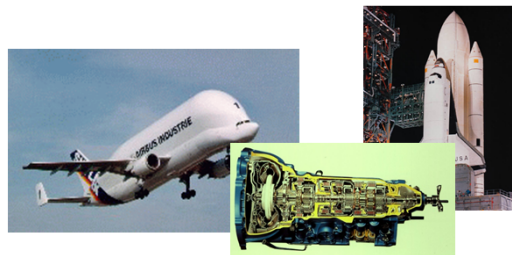
- Go to <https://openmodelica.org/index.php/download/download-linux> and follow the instructions.

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`

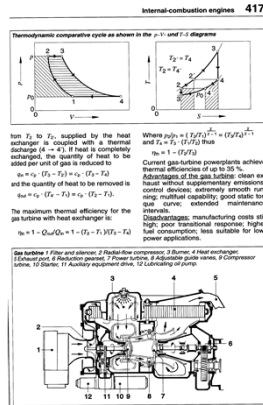
Part I

Introduction to Modelica and a demo example



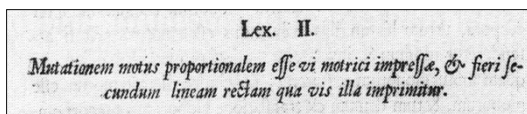
Modelica Background: Stored Knowledge

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.ze. — | — . 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”:

variable = expression

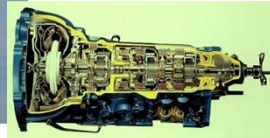
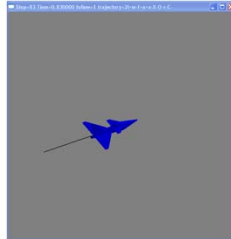
$v = \text{INTEG}(F) / m$

Programming languages usually do not allow equations!

What is Modelica?

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

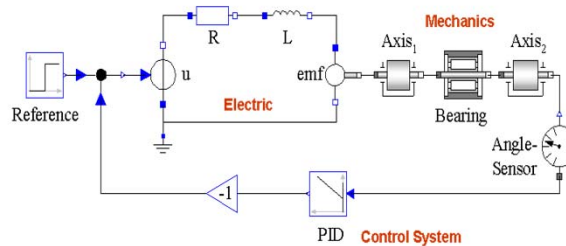
- Robotics
- Automotive
- Aircrafts
- Satellites
- Power plants
- Systems biology



MODELICA

What is Modelica?

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



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

What is Modelica?

A **language** for modeling of complex 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
- MathModelica by MathCore
- Dymola by Dassault systems / Dynasim
- SimulationX by ITI
- MapleSim by MapleSoft

Available at: www.modelica.org

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

Modelica Acausal Modeling

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*:

$$\mathbf{R} \cdot \mathbf{i} = \mathbf{v};$$

can be used in three ways:

$$\mathbf{i} := \mathbf{v}/\mathbf{R};$$

$$\mathbf{v} := \mathbf{R} \cdot \mathbf{i};$$

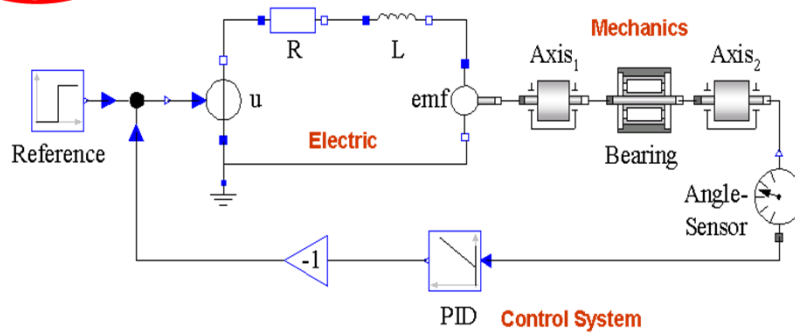
$$\mathbf{R} := \mathbf{v}/\mathbf{i};$$

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



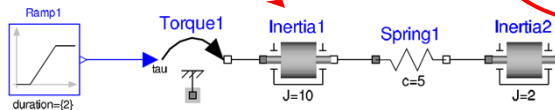
What is Special about Modelica?

Multi-Domain Modeling

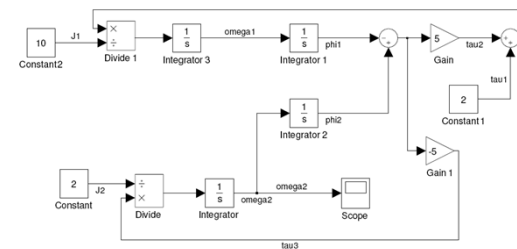
Visual Acausal Hierarchical Component Modeling

Keeps the physical structure

Acausal model (Modelica)



Causal block-based model (Simulink)

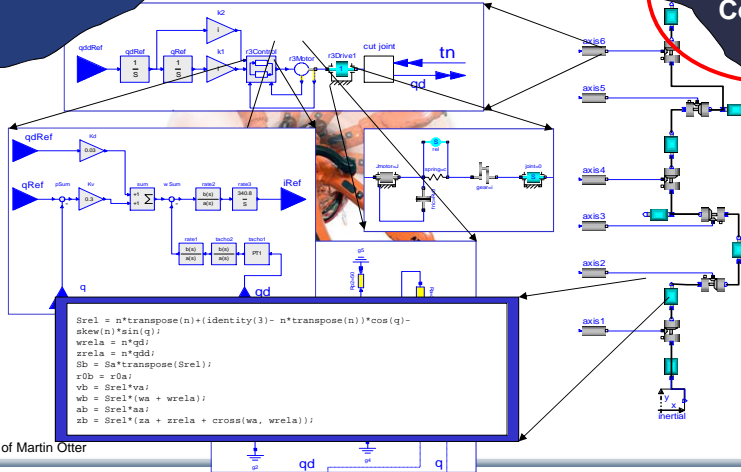


What is Special about Modelica?

Multi-Domain Modeling

Hierarchical system modeling

Visual Acausal Hierarchical Component Modeling



Courtesy of Martin Otter

21 Copyright © Open Source Modelica Consortium

MODELICA

What is Special about Modelica?

Multi-Domain Modeling

A textual *class-based* language
OO primary used for as a structuring concept

Visual Acausal Hierarchical Component Modeling

Behaviour described declaratively using

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

Variable declarations

Typed Declarative Equation-based Textual Language

```

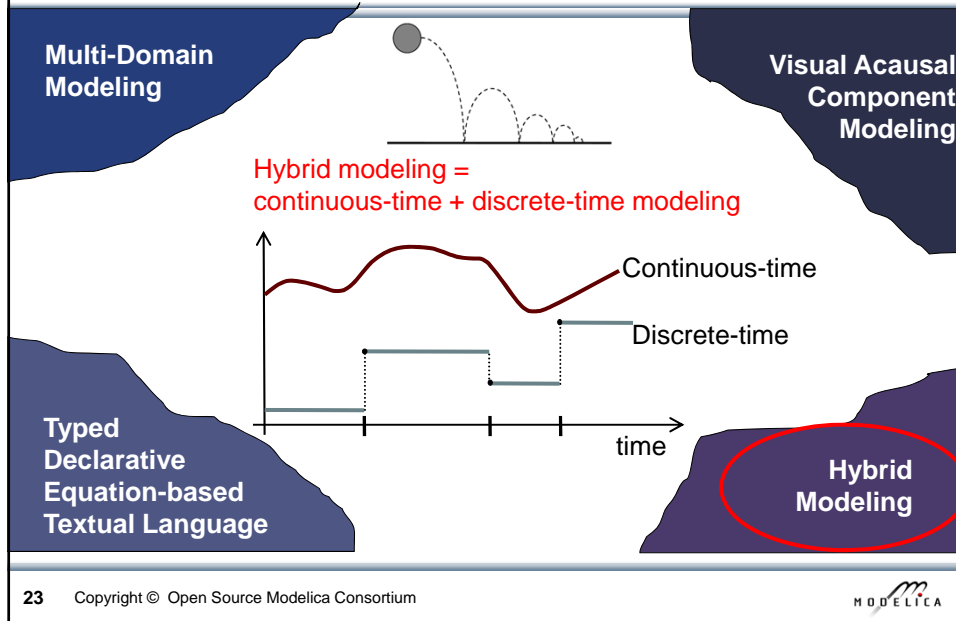
class VanDerPol "Van der Pol oscillator model"
  Real x(start = 1) "Descriptive string for x";
  Real y(start = 1) "y coordinate";
  parameter Real lambda = 0.3;
equation
  der(x) = y;
  der(y) = -x + lambda*(1 - x*x)*y;
end VanDerPol;
    
```

Differential equations

22 Copyright © Open Source Modelica Consortium

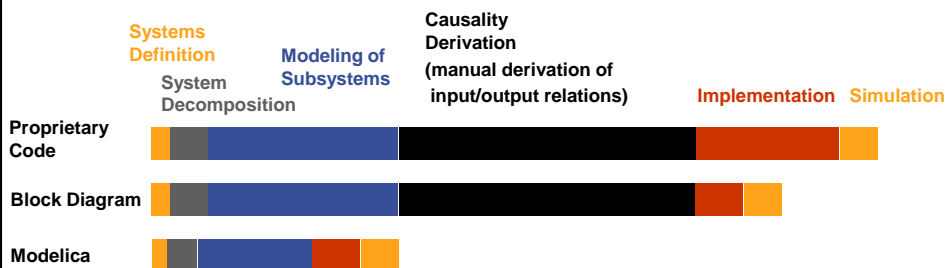
MODELICA

What is Special about Modelica?



Modelica – Faster Development, Lower Maintenance than with Traditional Tools

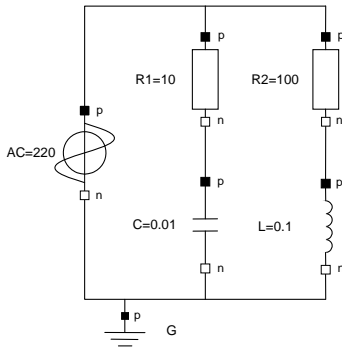
Block Diagram (e.g. Simulink, ...) or
Proprietary Code (e.g. Ada, Fortran, C,...)
vs Modelica



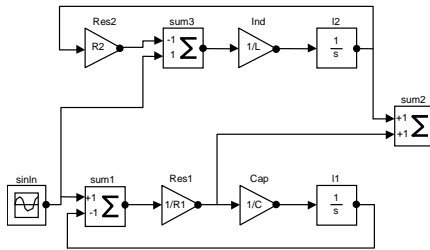
Modelica vs Simulink Block Oriented Modeling Simple Electrical Model

Modelica:
Physical model –
easy to understand

Keeps the
physical
structure



Simulink:
Signal-flow model – hard to
understand



25 Copyright © Open Source Modelica Consortium

MODELICA

Graphical Modeling - Using Drag and Drop Composition

26 Copyright © Open Source Modelica Consortium

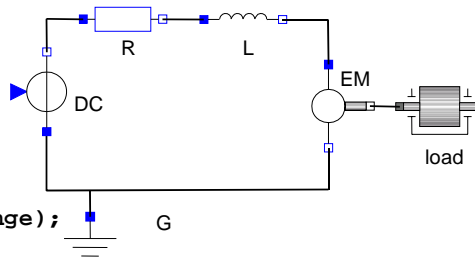
MODELICA

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
  
```



27 Copyright © Open Source Modelica Consortium

MODELICA

Corresponding DCMotor Model Equations

The following equations are automatically derived from the Modelica model:

0 == DC.p.i + R.n.i	EM.u == EM.p.v - EM.n.v	R.u == R.p.v - R.n.v
DC.p.v == R.n.v	0 == EM.p.i + EM.n.i	0 == R.p.i + R.n.i
	EM.i == EM.p.i	R.i == R.p.i
0 == R.p.i + L.n.i	EM.u == EM.k * EM.ω	R.u == R.R * R.i
R.p.v == L.n.v	EM.i == EM.M / EM.k	
	EM.J * EM.ω == EM.M - EM.b * EM.ω	L.u == L.p.v - L.n.v
0 == L.p.i + EM.n.i		0 == L.p.i + L.n.i
L.p.v == EM.n.v	DC.u == DC.p.v - DC.n.v	L.i == L.p.i
	0 == DC.p.i + DC.n.i	L.u == L.L * L.i'
0 == EM.p.i + DC.n.i	DC.i == DC.p.i	
EM.p.v == DC.n.v	DC.u == DC.Amp * Sin[2 * π * DC.f * t]	
0 == DC.n.i + G.p.i		
DC.n.v == G.p.v	(load component not included)	

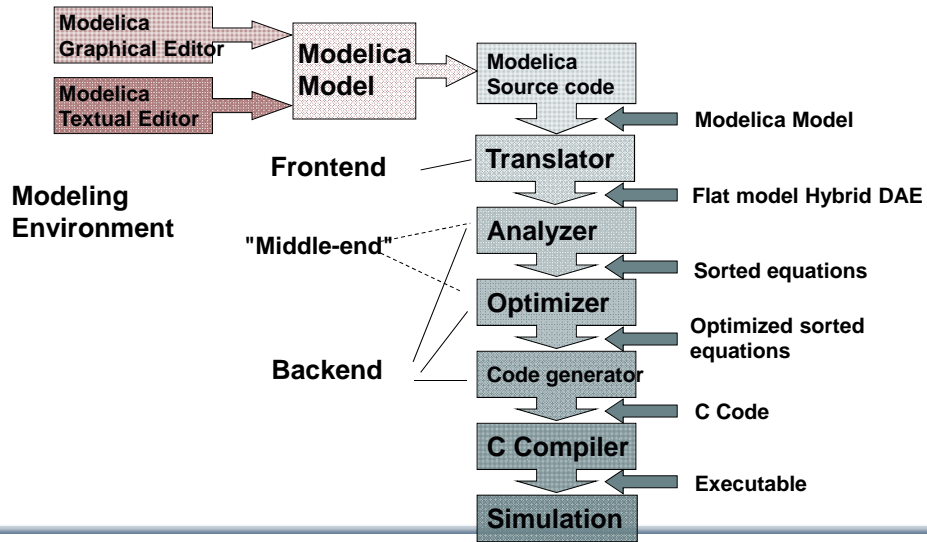
Automatic transformation to ODE or DAE for simulation:

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

28 Copyright © Open Source Modelica Consortium

MODELICA

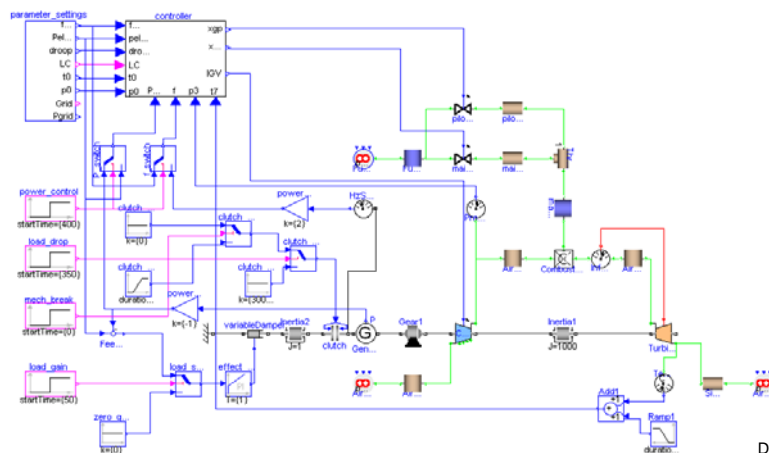
Model Translation Process to Hybrid DAE to Code



29 Copyright © Open Source Modelica Consortium



Modelica in Power Generation GTX Gas Turbine Power Cutoff Mechanism



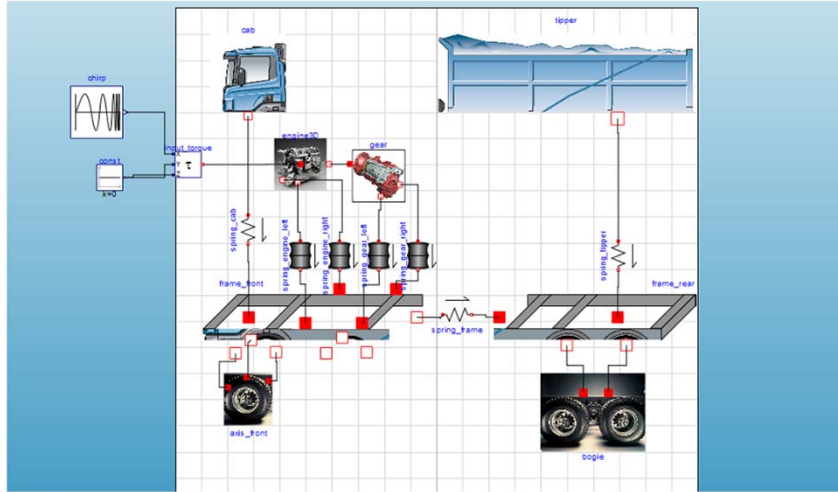
Developed by MathCore for Siemens

Courtesy of Siemens Industrial Turbomachinery AB

30 Copyright © Open Source Modelica Consortium



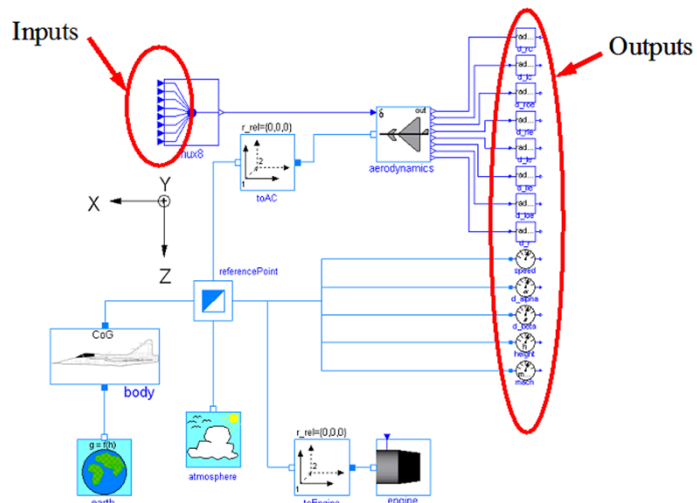
Modelica in Automotive Industry



31 Copyright © Open Source Modelica Consortium



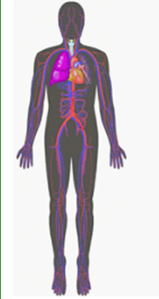
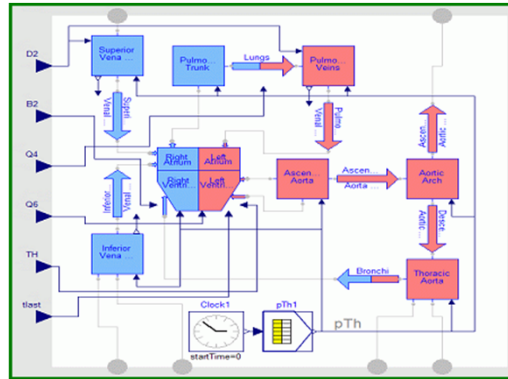
Modelica in Avionics



32 Copyright © Open Source Modelica Consortium



Modelica in Biomechanics



33 Copyright © Open Source Modelica Consortium

MODELICA

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 Martin Otter, DLR, Oberpfaffenhofen, Germany

34 Copyright © Open Source Modelica Consortium

MODELICA

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 expected May 2012
- 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
- Coming: The 9th International Modelica conference Sept 3-5, 2012 in Munich, Germany

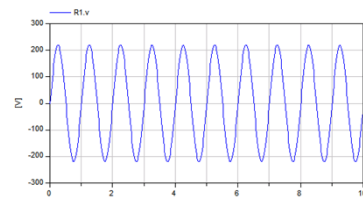
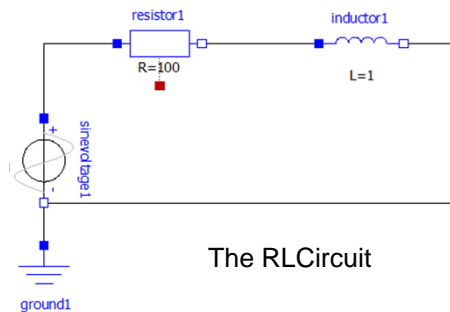
Exercises Part I Hands-on graphical modeling (15 minutes)

37 Copyright © Open Source Modelica Consortium



Exercises Part I – Basic Graphical Modeling

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



38 Copyright © Open Source Modelica Consortium




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**.

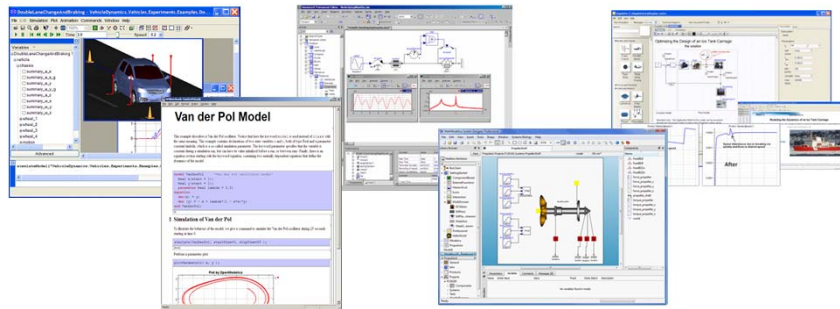
- Under the **Modelica Library**:
 - Contains The standard Modelica library components
 - The **Modelica files** contains the list of models you have created.

Exercises Part I – OMEdit Instructions (Part II)

- For the RLCircuit model, browse the Modelica standard library and add the following component models:
 - Add `Ground`, `Inductor` and `Resistor` component models from `Modelica.Electrical.Analog.Basic` package.
 - Add `SineVoltage` component model from `Modelica.Electrical.Analog.Sources` package.
- Make the corresponding connections between the component models as shown in slide 37.
- 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

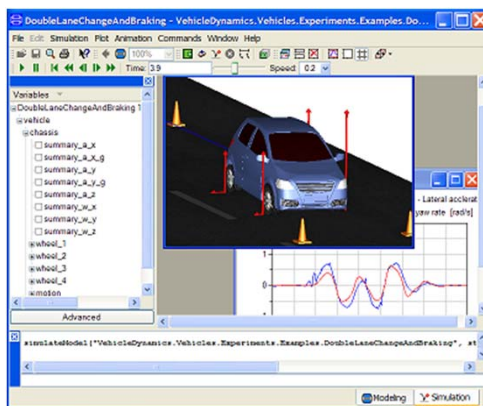
Modelica environments and OpenModelica



41 Copyright © Open Source Modelica Consortium



Dymola

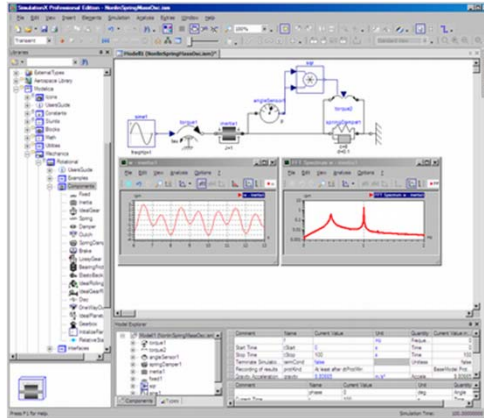


- Dynasim (Dassault Systemes)
- Sweden
- First Modelica tool on the market
- Main focus on automotive industry
- www.dynasim.com

42 Copyright © Open Source Modelica Consortium



Simulation X

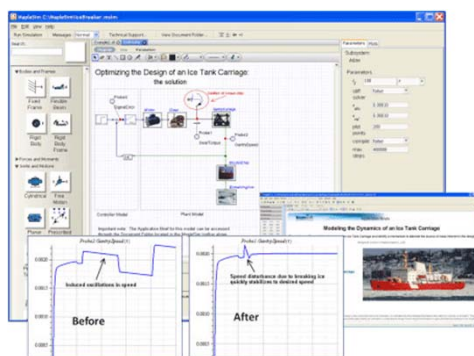


- ITI
- Germany
- Mechatronic systems
- www.simulationx.com

43 Copyright © Open Source Modelica Consortium



MapleSim

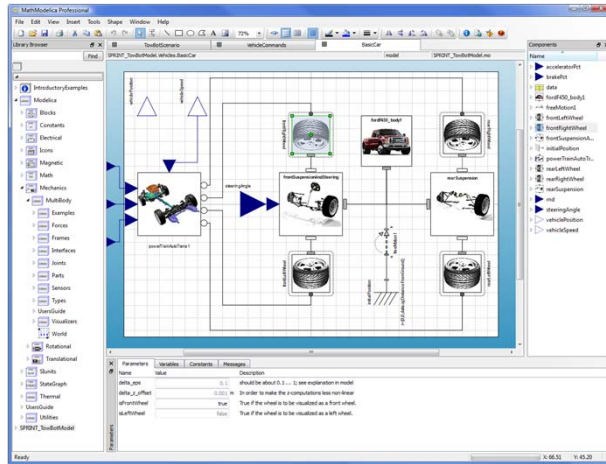


- Maplesoft
- Canada
- Recent Modelica tool on the market
- Integrated with Maple
- www.maplesoft.com

44 Copyright © Open Source Modelica Consortium

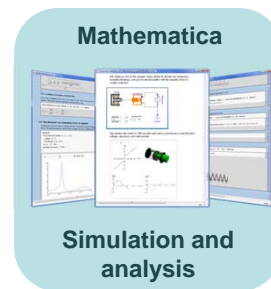


MathModelica – MathCore / Wolfram Research



Car model graphical view

- Wolfram Research
- USA, Sweden
- General purpose
- Mathematica integration
- www.wolfram.com
- www.mathcore.com



Courtesy
Wolfram
Research

45 Copyright © Open Source Modelica Consortium

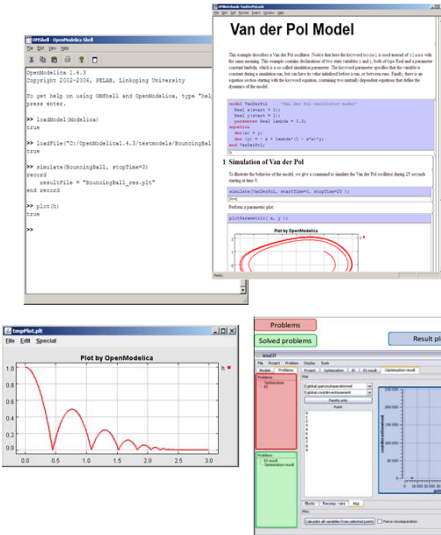


The OpenModelica Environment
www.OpenModelica.org

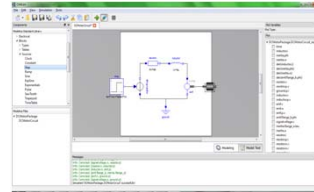
46 Copyright © Open Source Modelica Consortium



OpenModelica (Part I)



- OpenModelica
- Open Source Modelica Consortium (OSMC)
- Sweden and other countries
- Open source
- www.openmodelica.org



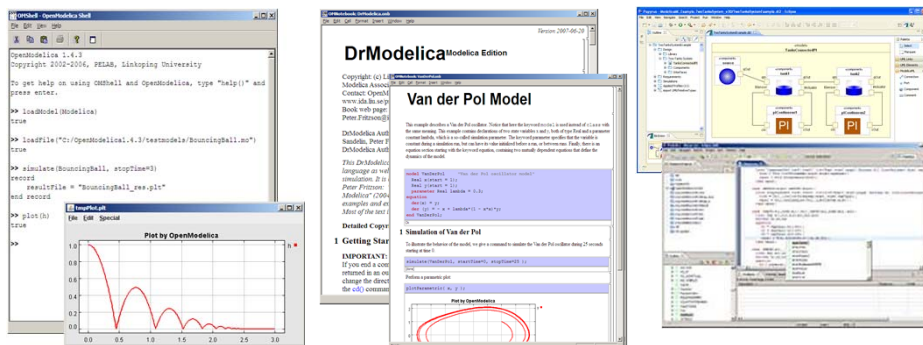
- OMEdit, graphical editor
- OMOptim, optimization subsystem

47 Copyright © Open Source Modelica Consortium



OpenModelica (Part II)

- Advanced Interactive Modelica compiler (OMC)
 - Supports most of the Modelica Language
- Basic environment for creating models
 - OMShell – an interactive command handler
 - OMNotebook – a literate programming notebook
 - MDT – an advanced textual environment in Eclipse
 - ModelicaML UML Profile
 - MetaModelica extension
 - ParModelica extension



48 Copyright © Open Source Modelica Consortium



OSMC – Open Source Modelica Consortium

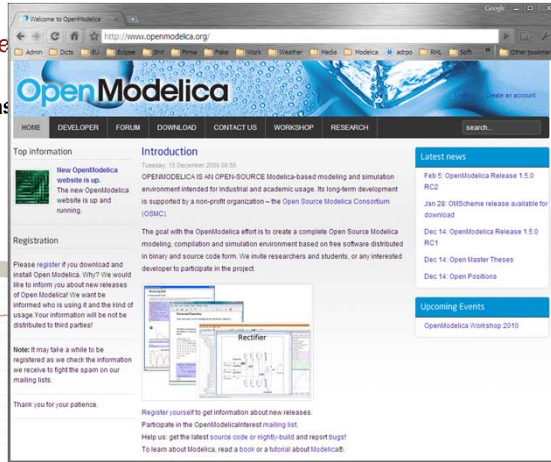
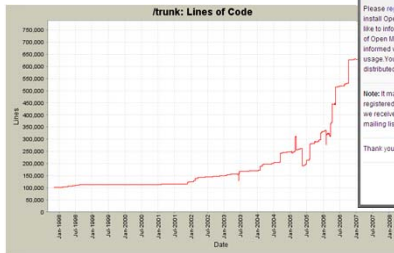
40 organizational members February 2012

Founded Dec 4, 2007

Open-source community services

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

Code Statistics



49 Copyright © Open Source Modelica Consortium



OSMC 40 Organizational Members, Feb 2012 (initially 7 members, 2007)

Companies and Institutes (22 members)

- ABB Corporate Research, Sweden
- Bosch Rexroth AG, Germany
- Siemens PLM, California, USA
- Siemens Turbo Machinery AB, Sweden
- CDAC Centre for Advanced Computing, Kerala, India
- Creative Connections, Prague, Czech Republic
- DHI, Aarhus, Denmark
- Evonik, Dehli, India
- Equa Simulation AB, Sweden
- Fraunhofer FIRST, Berlin, Germany
- Frontway AB, Sweden
- IFP, Paris, France
- InterCAX, Atlanta, USA
- ISID Dentsu, Tokyo, Japan
- MathCore Engineering/Wolfram, Sweden
- Maplesoft, Canada
- TLK Thermo, Germany
- Sozhou Tongyuan Software and Control, China
- Vi-grade, Italy
- VTI, Linköping, Sweden
- VTT, Finland
- XRG Simulation, Germany

Universities (18 members)

- Linköping University, Sweden
- TU Berlin, Institute of UEBC, Germany
- FH Bielefeld, Bielefeld, Germany
- TU Braunschweig, Institute of Thermodynamics, Germany
- TU Dortmund, Proc. Dynamics, Germany
- Technical University Dresden, Germany
- Université Laval, modelEAU, Canada
- Georgia Institute of Technology, USA
- Ghent University, Belgium
- Griffith University, Australia
- Hamburg Univ. Technology/TuTech, Institute of Thermo-Fluid, Germany
- University of Ljubljana, Slovenia
- University of Maryland, Inst. Systems Engineering, USA
- University of Maryland, CEE, USA
- Politecnico di Milano, Italy
- Ecoles des Mines, ParisTech, CEP, France
- Mälardalen University, Sweden
- Telemark University College, Norway

50 Copyright © Open Source Modelica Consortium



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)

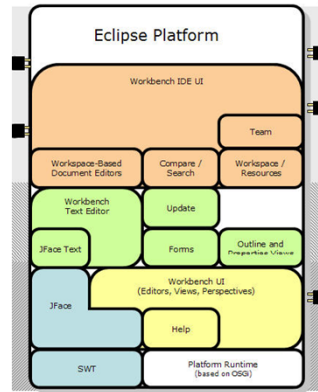
51 Copyright © Open Source Modelica Consortium

OMnotebook Interactive Electronic Notebook Here Used for Teaching Control Theory

52 Copyright © Open Source Modelica Consortium

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)



53 Copyright © Open Source Modelica Consortium



OpenModelica MDT: Code Outline and Hovering Info

Identifier Info on Hovering

Code Outline for easy navigation within Modelica files

The OpenModelica MDT Debugger (Eclipse-based) Using Japanese Characters

The screenshot displays the Eclipse IDE with the OpenModelica MDT Debugger. The top toolbar includes 'Debug' and 'Modelica'. The 'Debug' window shows the 'MDT GDB (Modelica Development Tooling (MDT) GDB)' interface. The 'Variables' window lists two variables: 'キャン・ザー・デバガー・シー・ミー' (Real, 1.5) and 'イエス・イット・キャン' (Real, -4.836697827222). The 'Breakpoints' window is empty. The source code editor shows a function definition in Japanese: 'function オープンモーターリッカー ロックス; input Real キャン・ザー・デバガー・シー・ミー; output Real イエス・イット・キャン; algorithm イエス・イット・キャン := sin(キャン・ザー・デバガー・シー・ミー); end オープンモーターリッカー ロックス;'. The console shows the execution of the main function: 'MDT GDB (Modelica Development Tooling (MDT) GDB) C:\OpenModelica\trunk\testsuite\bootstrapping\main.exe'.

55 Copy

Interactive Simulation with OpenModelica

The diagram illustrates the interactive simulation workflow in OpenModelica. It shows four main components:

- Simulation Control:** A window titled 'Simulation Center' with input fields for 'Source.flowLevel' (0.02), 'tank1.area' (1.0), and 'Source.flowLevel' (1.3). It includes 'Start', 'Pause', and 'Stop' buttons, a 'Simulation Time' field (00:01:56:220), and a status bar indicating 'Simulation is running...'.
- Plot View:** A window showing a line graph with multiple data series over time.
- Requirements Evaluation View in ModelicaML:** A window titled 'Requirements View' showing a table of requirements with columns for 'Req ID', 'Name', 'Status', and 'Start Time'. It includes a 'Show violations (2)' button and a 'Details' button.
- Domain-Specific Visualization View:** A window titled 'FormulatorView' showing a 3D visualization of two tanks, 'Tank 1' and 'Tank 2', with various levels and flow indicators.

Arrows indicate the flow of data and control between these components: from Simulation Control to Plot View, from Simulation Control to Requirements Evaluation View, and from Simulation Control to Domain-Specific Visualization View.

OMOptim – Optimization (1)

Model structure

Model Variables

Optimized parameters

Optimized Objectives

Name	Value	Description
global.sourceeadeville.h	1,18294e+06	[Mg]
global.sourceeadeville.RowPort.p	100000	
global.sourceeadeville.ColdB.h	1,413474e+06	[Mg]
global.sourceeadeville.ColdB.RowPort.p	100000	
global.sourceeadeville.ColdB.debit	12,78	[kg/s]
global.sourceeadeville.ECS.h	1,354954e+06	[Mg]
global.sourceeadeville.ECS.RowPort.p	100000	
global.sourceeadeville.ECS.eta	1	
global.sourceeadeville.ECS.debit1	0	
global.sourceeadeville.ECS.debit	1	[kg/s]
global.sourceeadeville.B.h	1,354954e+06	[Mg]
global.sourceeadeville.B.RowPort.p	100000	
global.sourceeadeville.B.eta	1	
global.sourceeadeville.B.debit	1,22612	[kg/s]
global.sourceeadeville.A.h	1,354954e+06	[Mg]
global.sourceeadeville.A.RowPort.p	100000	
global.sourceeadeville.A.eta	1	
global.sourceeadeville.A.debit	0,601234	[kg/s]
global.scenarioEadeville.debit	0,340001	[kg/s]
global.scenarioEadeville.p	0	

Name	Description	Opt. Minimum
global.sourceeadeville.debit	[kg/s]	0
global.sourceeadeville.debit	[kg/s]	0
global.scenarioPACA.MySpecPomp		0
global.scenarioPACA.MySpecPomp		0

Name	Description	Direction	Value
global.gaincouteoperationnel		Maximize	0
global.coutdinvestissement		Minimize	0

Problems

OMOptim – Optimization (2)

Solved problems

Result plot

Export result data .csv

Plot

X: global.gaincouteoperationnel

Y: global.coutdinvestissement

Pareto only

Point

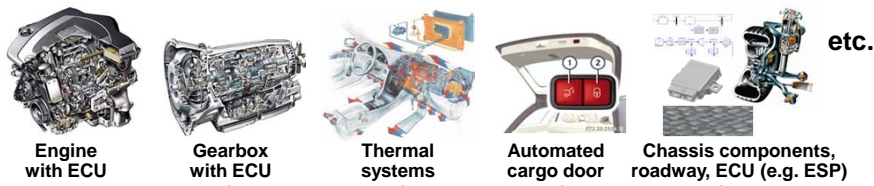
Point	global.gaincouteoperationnel	global.coutdinvestissement
0	0	0
1	10000	20000
2	20000	40000
3	30000	60000
4	40000	80000
5	50000	100000
6	60000	120000
7	70000	140000
8	80000	160000
9	80000	200000

Export...

General Tool Interoperability & Model Exchange Functional Mock-up Interface (FMI)

The FMI development is part of the MODELISAR 29-partner project

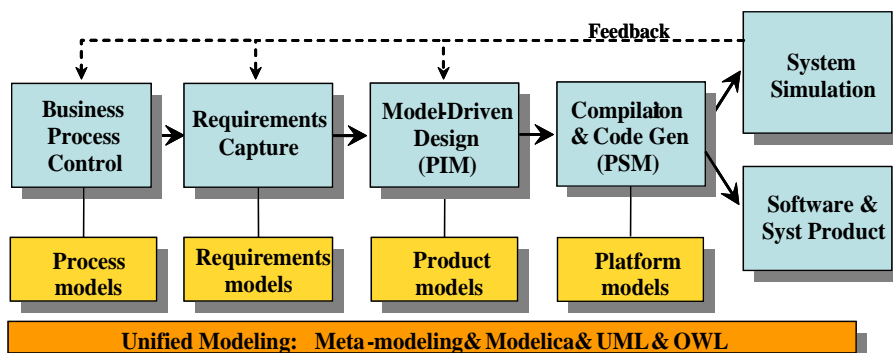
- FMI development initiated by **Daimler**
- Improved Software/Model/Hardware-in-the-Loop Simulation, of **physical** models and of **AUTOSAR** controller models from **different vendors** for automotive applications with **different levels of detail**.
- **Open Standard**
- **14 automotive use cases** for evaluation
- **> 10 tool vendors** are supporting it



functional mockup interface for model exchange and tool coupling

courtesy Daimler

OPENPROD – Large 28-partner European Project, 2009-2012 Vision of Cyber-Physical Model-Based Product Development



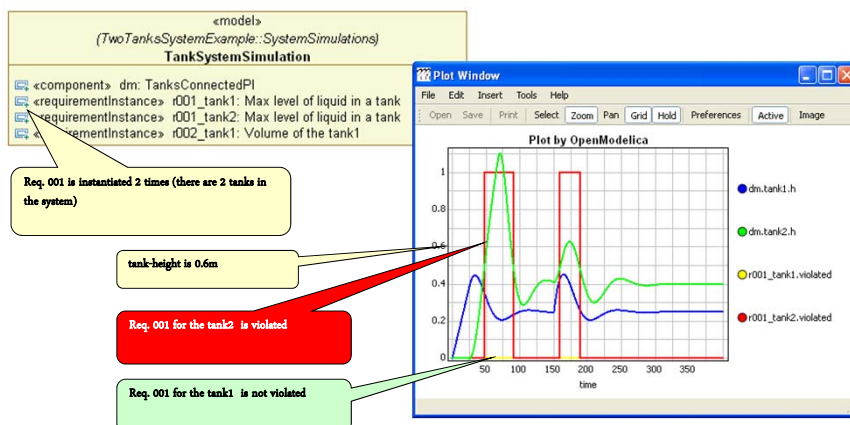
OPENPROD Vision of unified modeling framework for model-driven product development from platform independent models (PIM) to platform specific models (PSM)

Current work based on Eclipse, UML/SysML, OpenModelica

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

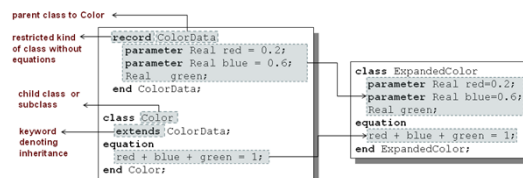


OpenModelica – Recent Developments and Plans

- January 2012. OpenModelica 1.8.1 release with operator overloading, faster compilation, ModelicaML with valuebindings
- 2012. Continued high priority on better support for the Modelica standard library.
- Spring 2012. Support for larger models and improved simulation.
- February 2012. Shifting to bootstrapped OpenModelica compiler for development.
- March 2011. Subset Fluid library flattening and simulating
- March 2012. Thermopower library simulating
- March 2011. Further improved support for MultiBody simulation.
- April 2011. Most of Fluid library flattening
- April-May 2011. Most of Media and Fluid libraries simulating
- May-June 2012. Integrated Modelica debugger.

Part III

Modelica language concepts and textual modeling



Typed
Declarative
Equation-based
Textual Language

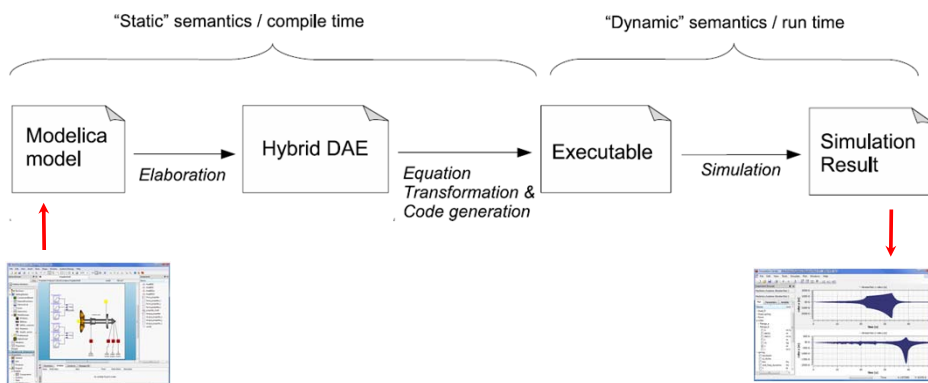
Hybrid
Modeling

Acausal Modeling

The order of computations is not decided at modeling time

	Acausal	Causal
Visual Component Level		
Equation Level	<p>A resistor <i>equation</i>:</p> $R \cdot i = v;$	<p>Causal possibilities:</p> $i := v/R;$ $v := R \cdot i;$ $R := v/i;$

Typical Simulation Process



Simple model - Hello World!

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

Continuous-time
variable
Parameter, constant
during simulation

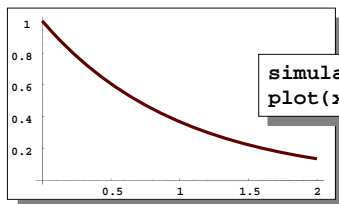
Name of model

Initial condition

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

Simulation in OpenModelica environment

Differential equation



```
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

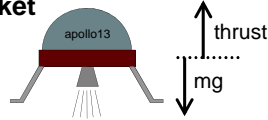
- constant**

- parameter**

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

A Simple Rocket Model

Rocket



$$acceleration = \frac{thrust - mass \cdot gravity}{mass}$$

$$mass' = -massLossRate \cdot abs(thrust)$$

$$altitude' = velocity$$

$$velocity' = acceleration$$

```

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

```

new model ←

parameters (changeable before the simulation) ←

floating point type ←

differentiation with regards to time ←

→ declaration comment

→ start value

→ name + default value

→ mathematical equation (acausal)

Celestial Body Class

A class declaration creates a *type name* in 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

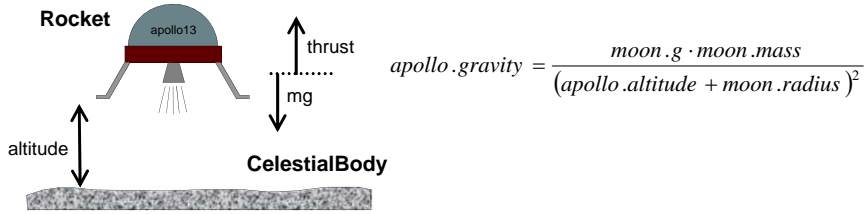
```

...
CelestialBody moon;
...

```

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;

```

only access
inside the class

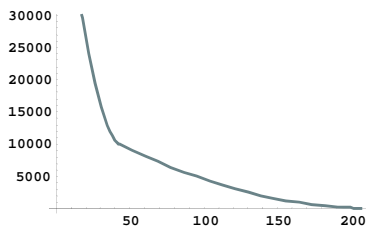
access by dot
notation outside
the class

Simulation of Moon Landing

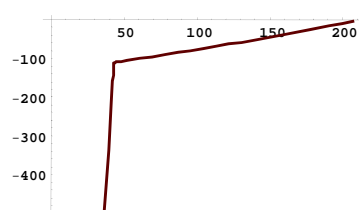
```

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

```
function sum
  input Real arg1;
  input Real arg2;
  output Real result;
algorithm
  result := arg1+arg2;
end sum;
```

Function Call – Example Function with for-loop

Example Modelica function call:

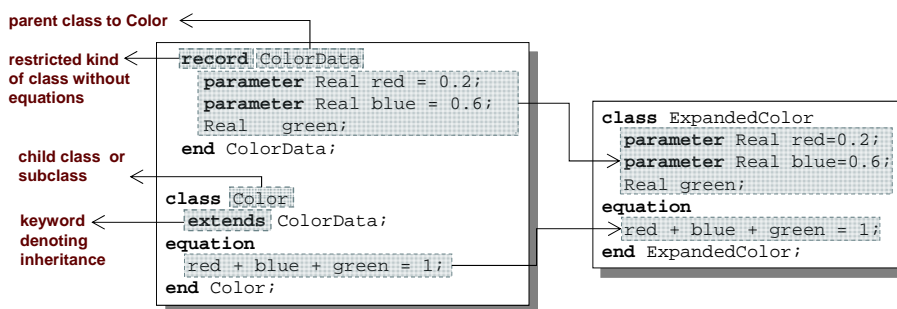
```
...
p = polynomialEvaluator({1,2,3,4},21)
```

{1,2,3,4} becomes the value of the coefficient vector A, and 21 becomes the value of the formal parameter 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;
```

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

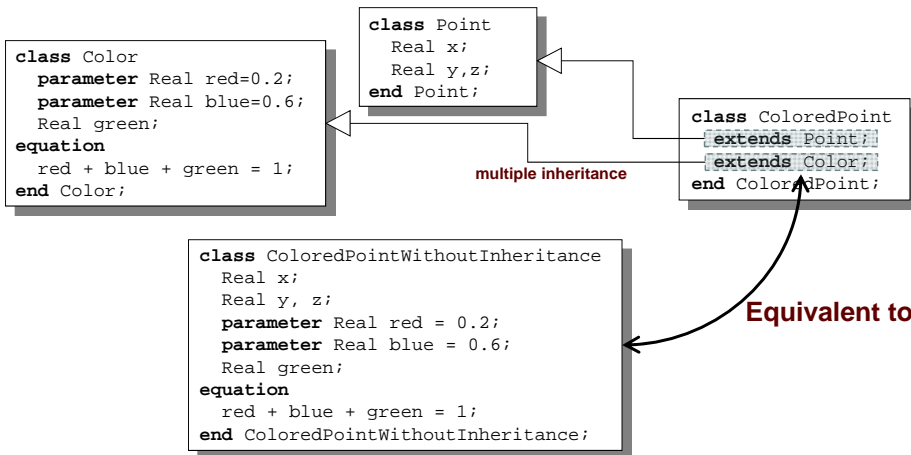
Inheritance



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

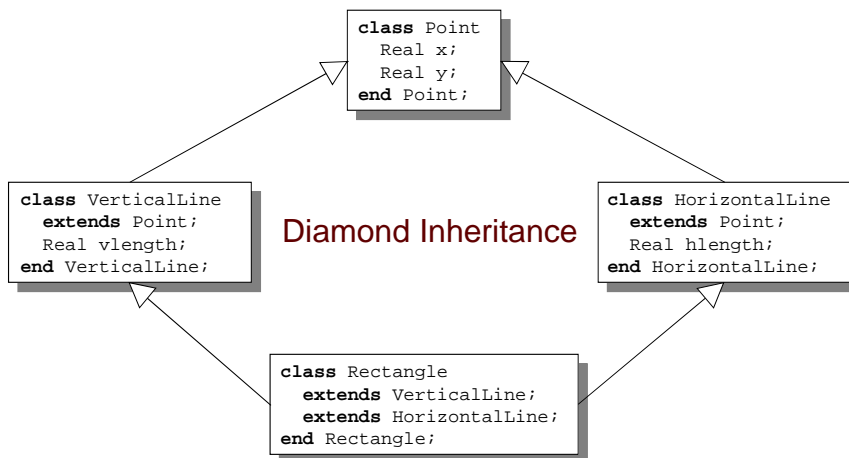
Multiple Inheritance

Multiple Inheritance is fine – inheriting both geometry and color



Multiple Inheritance cont'

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



Simple Class Definition

- Simple Class Definition
 - Shorthand Case of Inheritance
- Example:
- Often used for introducing new names of types:

```
class SameColor = Color;
```

Equivalent to:

```
class SameColor
  extends Color;
end SameColor;
```

inheritance ←

```
type Resistor = Real;
```

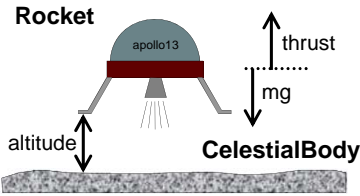
```
connector MyPin = Pin;
```

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 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)

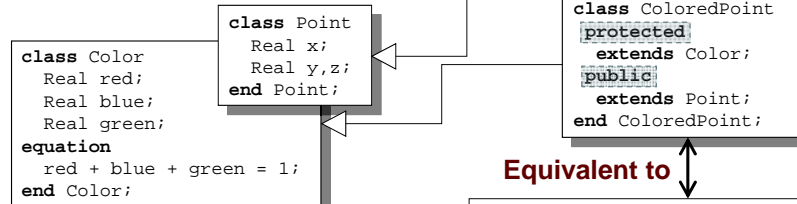
```

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;
    
```

inherited parameters

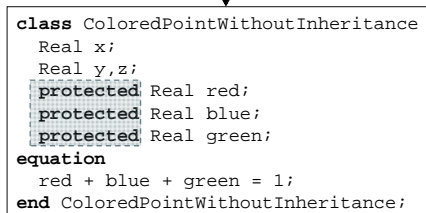
Inheritance of Protected Elements

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 II (30 minutes)

Exercises Part II

- 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.

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

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

- Locate the VanDerPol model in DrModelica (link from Section 2.1), using OMNotebook!
- **Exercise 2.2:** Simulate and plot VanDerPol. Do a slight change in the model, re-simulate and re-plot.

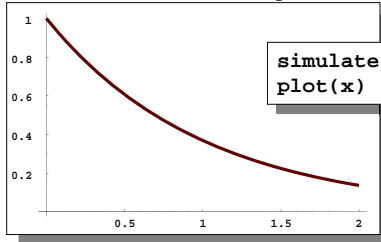
Exercise 2.1 – Hello World!

A Modelica “Hello World” model

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

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

Simulation in OpenModelica environment

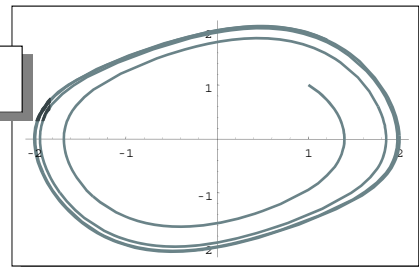


```
simulate(HelloWorld, stopTime = 2)  
plot(x)
```

Exercise 2.2 – Van der Pol Oscillator

```
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)
```



Exercise 2.3 – DAE Example

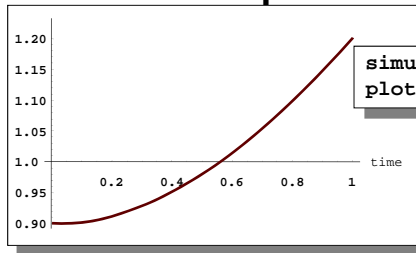
Include algebraic equation

Algebraic equations contain no derivatives

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

```
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;
```

Simulation in OpenModelica environment



```
simulate(DAEexample, stopTime = 1)
plot(x)
```

Exercise 2.4 – Model the system below

- Model this Simple System of Equations in Modelica

$$\dot{x} = 2 * x * y - 3 * x$$

$$\dot{y} = 5 * y - 7 * x * y$$

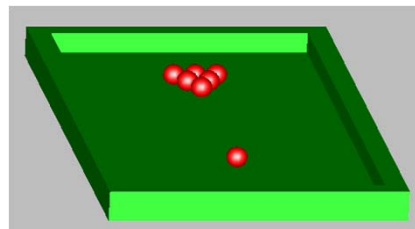
$$x(0) = 2$$

$$y(0) = 3$$

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`.

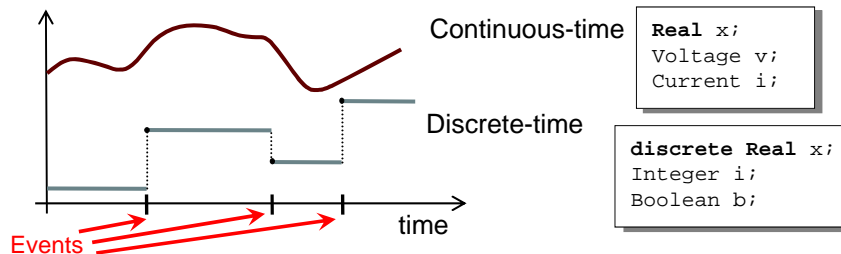
Discrete Events and Hybrid Systems



Picture: Courtesy Hilding Elmqvist

Hybrid Modeling

Hybrid modeling = continuous-time + discrete-time modeling



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

Event creation – if

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

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

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

False if $s < 0$

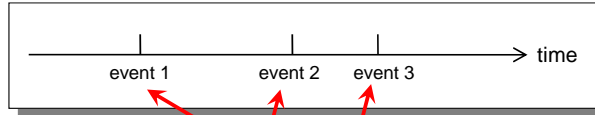
If-equation choosing equation for v

If-expression

Event creation – when

when-equations

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



Equations only active at event times

Time event

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

Only dependent on time, can be scheduled in advance

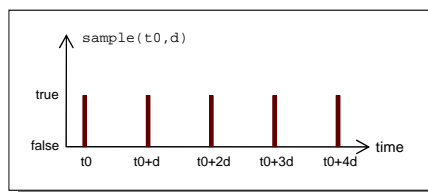
State event

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

Related to a state. Check for zero-crossing

Generating Repeated Events

The call `sample(t0,d)` returns true and triggers events at times t_0+i*d , where $i=0,1,\dots$

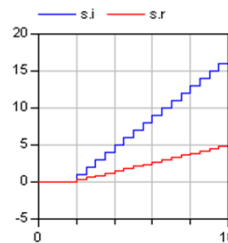


Variables need to be discrete

```
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;
```

Creates an event after 2 s, then each 0.5 s

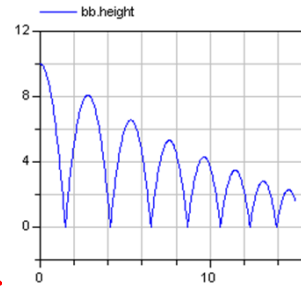
pre(...) takes the previous value before the event.



Reinit - discontinuous changes

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

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

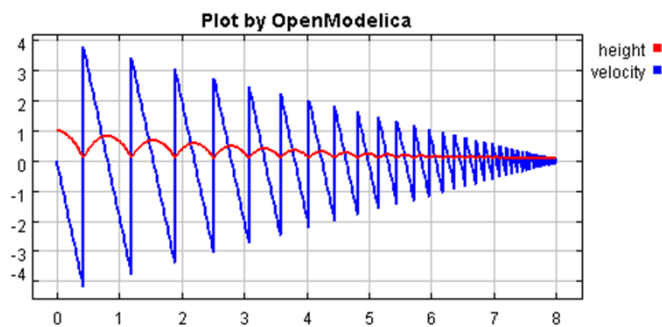


Reinit "assigns"
continuous-time variable
velocity a new value

Initial conditions

Exercise 2.6 – BouncingBall

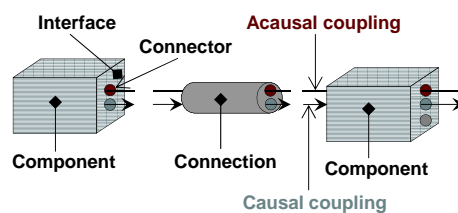
- Locate the BouncingBall model in one of the hybrid modeling sections of DrModelica (the When-Equations link in Section 2.9), run it, change it slightly, and re-run it.



Part IV

Components, Connectors and Connections – Modelica Libraries and Graphical Modeling

Software Component Model



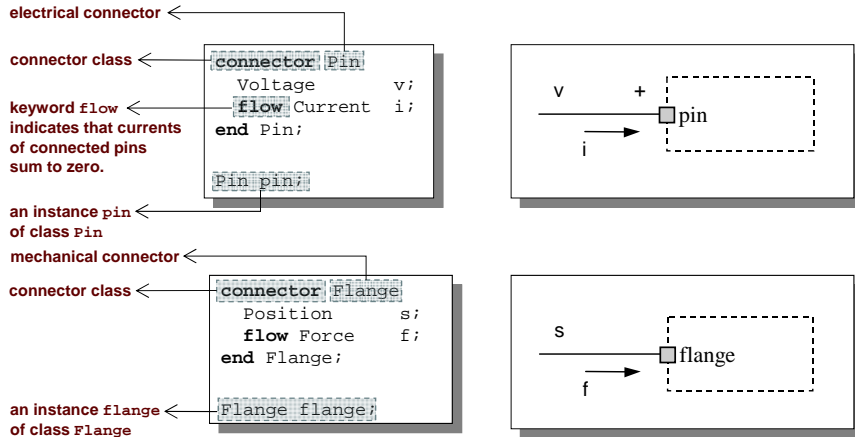
A component class should be defined *independently of the environment*, very essential for *reusability*

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*



The `flow` prefix

Two kinds of variables in connectors:

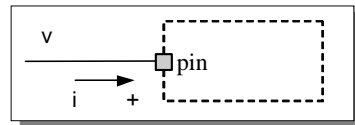
- *Non-flow variables* potential or energy level
- *Flow variables* represent some kind of flow

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

positive flow direction:



Physical Connector

- Classes Based on Energy Flow

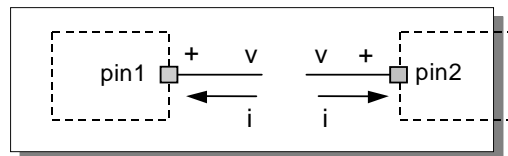
Domain Type	Potential	Flow	Carrier	Modelica Library
Electrical	Voltage	Current	Charge	Electrical. Analog
Translational	Position	Force	Linear momentum	Mechanical. Translational
Rotational	Angle	Torque	Angular momentum	Mechanical. Rotational
Magnetic	Magnetic potential	Magnetic flux rate	Magnetic flux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

connect-equations

Connections between connectors are realized as *equations* in Modelica

```
connect(connector1,connector2)
```

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:

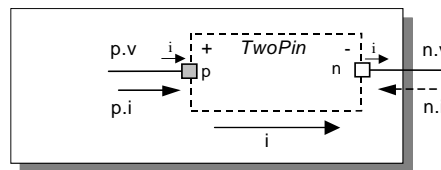
$$v_1 = v_2 = v_3 = \dots v_n$$

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

$$i_1 + i_2 + \dots (-i_k) + \dots i_n = 0$$

Common Component Structure

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



partial class
(cannot be
instantiated)

positive pin
negative pin

```
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
```

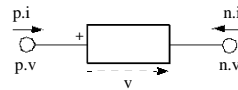
```
connector Pin
  Voltage v;
  flow Current i;
end Pin;
```

electrical connector class

Electrical Components

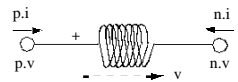
```

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



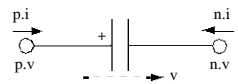
```

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



```

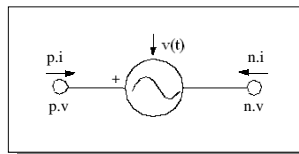
model Capacitor "Ideal electrical capacitor"
  extends TwoPin;
  parameter Real C ;
  equation
    i=C*der(v);
  end Capacitor;
  
```



Electrical Components cont'

```

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

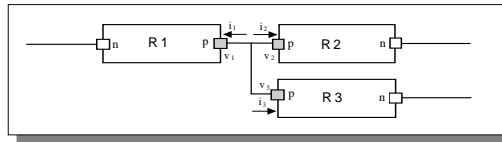


```

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



Resistor Circuit



```
model ResistorCircuit
  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;
```

Corresponds to

```
R1.p.v = R2.p.v;
R1.p.v = R3.p.v;
R1.p.i + R2.p.i + R3.p.i = 0;
```

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>

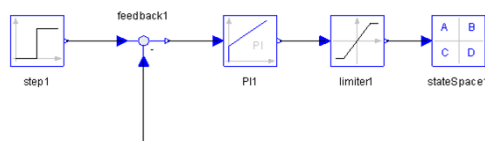
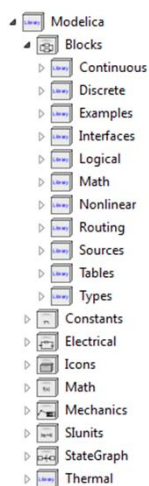
Modelica Standard Library cont'

The Modelica Standard Library contains components from various application areas, including the following sublibraries:

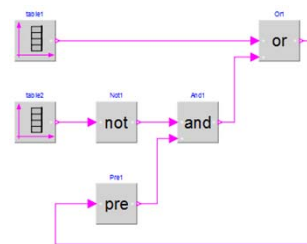
- Blocks Library for basic input/output control blocks
- Constants Mathematical constants and constants of nature
- Electrical Library for electrical models
- Icons Icon definitions
- Fluid 1-dim Flow in networks of vessels, pipes, fluid machines, valves, etc.
- Math Mathematical functions
- Magnetic Magnetic.Fluxtubes – for magnetic applications
- Mechanics Library for mechanical systems
- Media Media models for liquids and gases
- Slunits 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



Analog



Digital

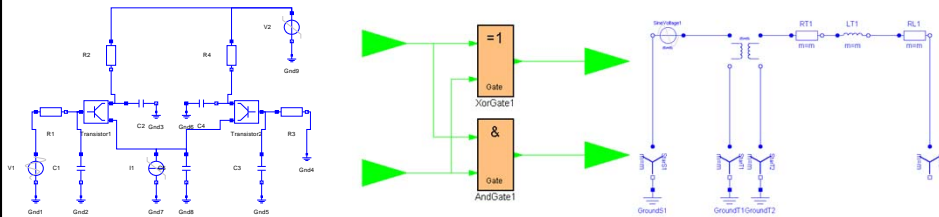


Machines



MultiPhase

Examples:



113 Copyright © Open Source Modelica Consortium

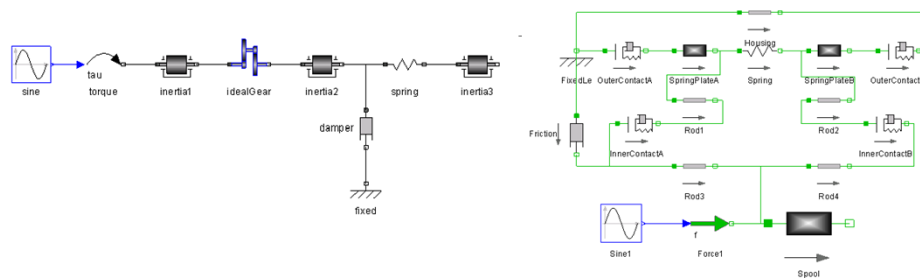
MODELICA

Modelica.Mechanics

Package containing components for mechanical systems

Subpackages:

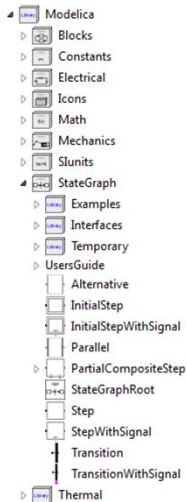
- Rotational 1-dimensional rotational mechanical components
- Translational 1-dimensional translational mechanical components
- MultiBody 3-dimensional mechanical components



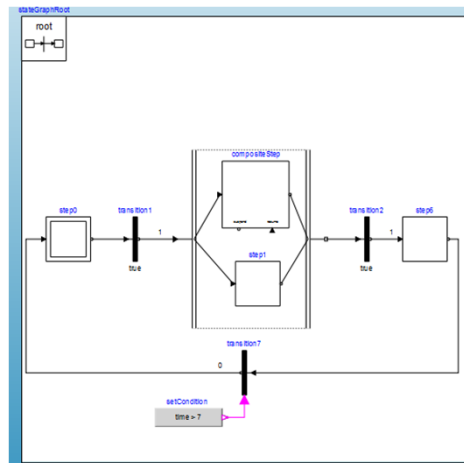
114 Copyright © Open Source Modelica Consortium

MODELICA

Modelica.Stategraph



Hierarchical state machines (similar to Statecharts)



115 Copyright © Open Source Modelica Consortium



Other Free Libraries

- WasteWater Wastewater treatment plants, 2003
- ATPlus Building simulation and control (fuzzy control included), 2004
- 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
- External.Media Library External fluid property computation, 2008
- VirtualLabBuilder Implementation of virtual labs, 2007
- SPOT Power systems in transient and steady-state mode, 2007
- ...

116 Copyright © Open Source Modelica Consortium

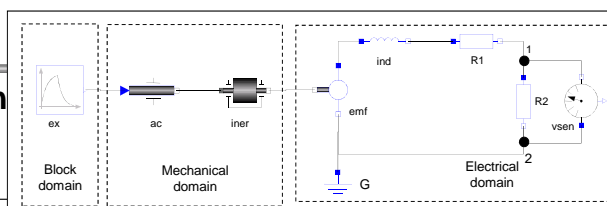


Some Commercial Libraries

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

Connecting Components from Multiple Domains

- Block domain
- Mechanical domain
- Electrical domain



model Generator

```

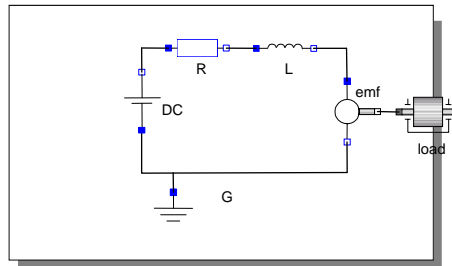
Modelica.Mechanics.Rotational.Accelerate ac;
Modelica.Mechanics.Rotational.Inertia iner;
Modelica.Electrical.Analog.Basic.EMF emf(k=-1);
Modelica.Electrical.Analog.Basic.Inductor ind(L=0.1);
Modelica.Electrical.Analog.Basic.Resistor R1,R2;
Modelica.Electrical.Analog.Basic.Ground G;
Modelica.Electrical.Analog.Sensors.VoltageSensor vsens;
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;

```

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;
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;
```



Exercises Part IV Graphical Modeling Exercises

using
OpenModelica

Graphical Modeling - Using Drag and Drop Composition

The screenshot displays the OMEdit - OpenModelica Connection Editor interface. The main workspace shows a circuit diagram for a DC motor model. The circuit includes a voltage source labeled 'signalVoltage1' connected to a resistor 'resistor1' (with $R = \%R$), followed by an inductor 'inductor1' (with $L = \%L$). The circuit is connected to a ground 'ground1' and an electromechanical component 'emf1' (with $k = \%k$), which is further connected to an inertia 'inertia1' (with $J = \%J$). A step function 'step1' (with $\text{startTime} = \% \text{startTim}$) is connected to the voltage source. The left sidebar shows the 'Modelica Standard Library' with various components like 'CombiTimeTable', 'Constant', 'Exponential', 'ExpSine', etc. The bottom status bar shows '121 Copyright © Open Source Modelica Consortium' and the Modelica logo.

Graphical Modeling Animation – DCMotor

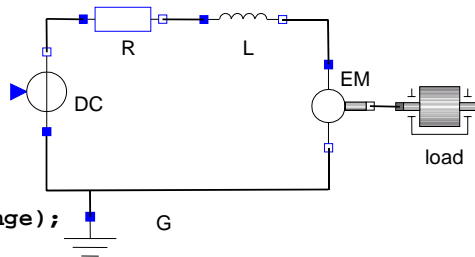
The screenshot shows the OMEdit - OpenModelica Connection Editor interface with an empty workspace. The left sidebar displays the 'Modelica Standard Library' and the 'Modelica' folder. The bottom status bar shows 'OMEdit, Version: 1.6.0' and 'OpenModelica, Version: "1.6.0"'. The interface is ready for a new model to be created or an existing one to be loaded.

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
  
```



123 Copyright © Open Source Modelica Consortium

MODELICA

Corresponding DCMotor Model Equations

The following equations are automatically derived from the Modelica model:

0 == DC.p.i + R.n.i	EM.u == EM.p.v - EM.n.v	R.u == R.p.v - R.n.v
DC.p.v == R.n.v	0 == EM.p.i + EM.n.i	0 == R.p.i + R.n.i
	EM.i == EM.p.i	R.i == R.p.i
0 == R.p.i + L.n.i	EM.u == EM.k * EM.ω	R.u == R.R * R.i
R.p.v == L.n.v	EM.i == EM.M / EM.k	
	EM.J * EM.ω == EM.M - EM.b * EM.ω	L.u == L.p.v - L.n.v
0 == L.p.i + EM.n.i		0 == L.p.i + L.n.i
L.p.v == EM.n.v	DC.u == DC.p.v - DC.n.v	L.i == L.p.i
	0 == DC.p.i + DC.n.i	L.u == L.L * L.i'
0 == EM.p.i + DC.n.i	DC.i == DC.p.i	
EM.p.v == DC.n.v	DC.u == DC.Amp * Sin[2 * π * DC.f * t]	
0 == DC.n.i + G.p.i		(load component not included)
DC.n.v == G.p.v		

Automatic transformation to ODE or DAE for simulation:

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

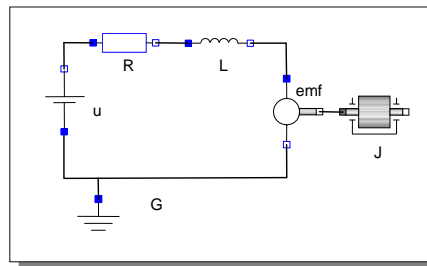
124 Copyright © Open Source Modelica Consortium

MODELICA

Exercise 3.1

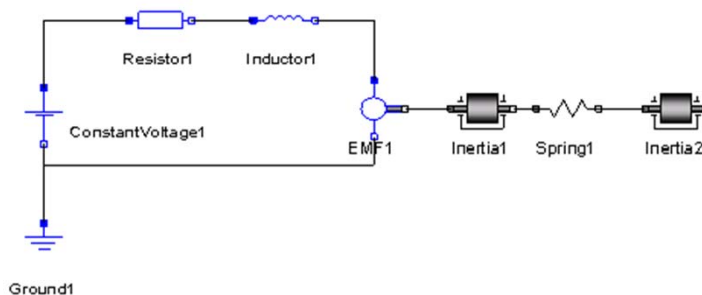
- Draw the `DCMotor` model using the graphic connection editor using models from the following Modelica libraries:
`Mechanics.Rotational.Components`,
`Electrical.Analog.Basic`,
`Electrical.Analog.Sources`

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



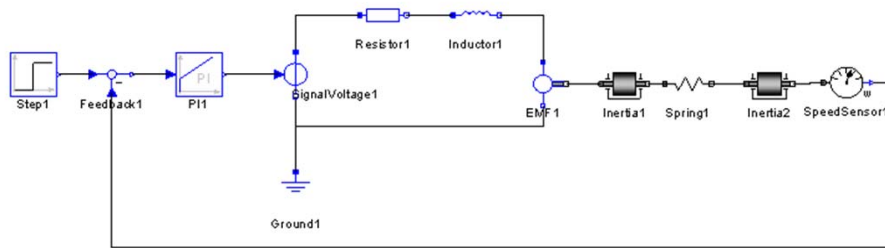
Exercise 3.2

- 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

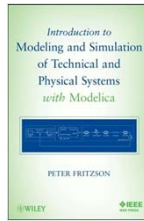
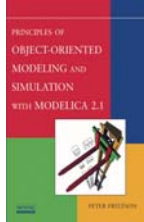
- If there is enough time: Add a PI controller to the system and try to control the rotational speed of the outgoing shaft. Verify the result using a step signal for input. Tune the PI controller by changing its parameters in OMEdit.



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:\OpenModelica1.6.0\share\omnotebook\drcontrol\DrControl.onb

Learn more...



- OpenModelica
 - www.openmodelica.org
- Modelica Association
 - www.modelica.org
- Books
 - Principles of Object Oriented Modeling and Simulation with Modelica 2.1, Peter Fritzson
<http://eu.wiley.com/WileyCDA/WileyTitle/productCd-0471471631.html>
 - Modeling and Simulation of Technical and Physical Systems with Modelica. Peter Fritzson.
<http://eu.wiley.com/WileyCDA/WileyTitle/productCd-111801068X.html>
 - Introduction to Modelica, Michael Tiller

Summary

Multi-Domain
Modeling

Visual Acausal
Component
Modeling

Typed
Declarative
Textual Language

Hybrid
Modeling

Thanks for listening!