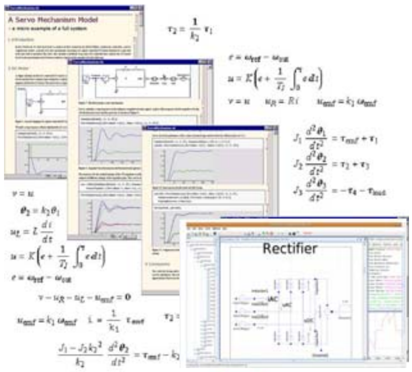
Introduction to Object-Oriented Modeling and Simulation with Modelica



Tutorial for MODPROD Workshop 2010

by

Peter Fritzson
Linköping University, petfr@ida.liu.se

Mohsen Torabzadeh-Tari
Linköping University, mohto@ida.liu.se

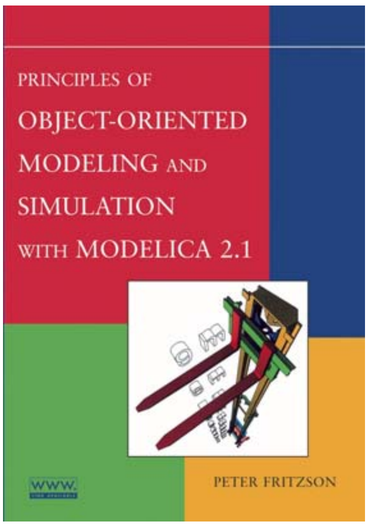
Slides
Based on book and lecture notes by Peter Fritzson
Contributions 2004-2005 by Emma Larsdotter Nilsson, Peter Bunus
Contributions 2006-2008 by Adrian Pop and Peter Fritzson
Contributions 2009 by David Broman, Peter Fritzson, Jan Brugård, and Mohsen Torabzadeh-Tari

Linköpings universitet

2010-02-09 MODPROD'2010

MODELICA

Tutorial Based on Book, 2004



Peter Fritzson
Principles of Object Oriented Modeling and Simulation with Modelica 2.1

Wiley-IEEE Press

940 pages

MODELICA

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 for contributions to the layout of these slides, and to Peter Bunus, David Broman, Jan Brugård for contributions.
- 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, and MathCore
- Modelica Association: www.modelica.org
- OpenModelica: www.openmodelica.org

Outline

Part I

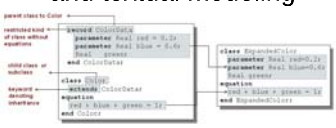
Introduction to Modelica and a demo example

Part II

Modelica environments

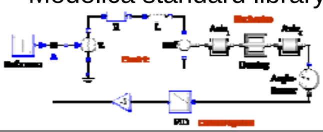
Part III

Modelica language concepts and textual modeling



Part IV

Graphical modeling and the Modelica standard library



Detailed Schedule

16:00 - Introduction to Modeling and Simulation

- Start installation of **OpenModelica** and **simForge** graphic editor

16:05 - Modelica – The Next Generation Modeling Language

16:15 - *Exercises Part I (15 minutes)*

- Short hands-on exercise on graphical modeling using **simForge** – RL Circuit

16:30 – Part II: Modelica Environments and the OpenModelica Environment

16:40 – Part III: Modelica Textual Modeling

17:00 - *Exercises Part IIIa (30 minutes)*

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

17:30 - Modelica Discrete Events and Hybrid Properties

17:45 - *Exercises Part IIIb (10 minutes)*

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

17:55 – Part IV: Components, Connectors and Connections
- Modelica Libraries

18:15 - Graphical Modeling using simForge and OpenModelica

18:15 - *Exercises Part IV (45 minutes) – DCMotor etc.*

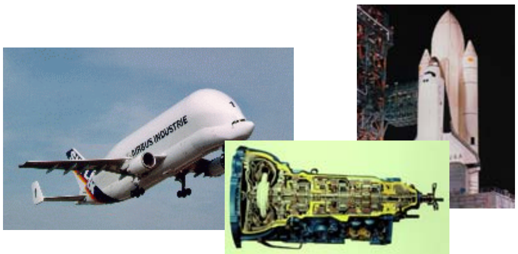
- Hands-on exercises on graphical modeling using **simForge** and OpenModelica

Software Installation

- Start the software installation
- Install OpenModelica-1.5.msi or OpenModelica-1.4.5.msi, and simForge (e.g. SimForge-0.8.4.1.jar) from the USB Stick
- (If you have a Mac or Linux computer, install OpenModelica-1.4.5)

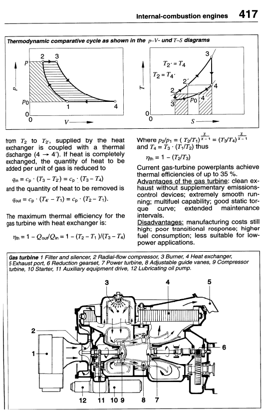
Part I

Introduction to Modelica and
a demo example

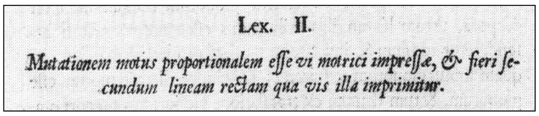


Modelica Background: Stored Knowledge

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

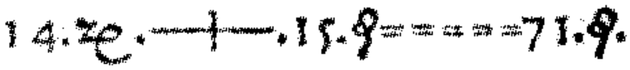


Lex. II.

Mutationem motus proportionalem esse vi motrici impressæ, & fieri secundum lineam rectam qua vis illa imprimitur.

Modelica Background: The Form – Equations

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



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":

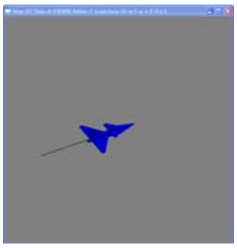
```
variable = expression
v = INTEG(F)/m
```

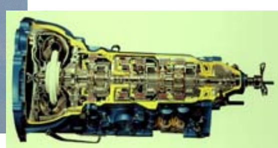
Programming languages usually do not allow equations!

What is Modelica?

A language for modeling of complex physical systems

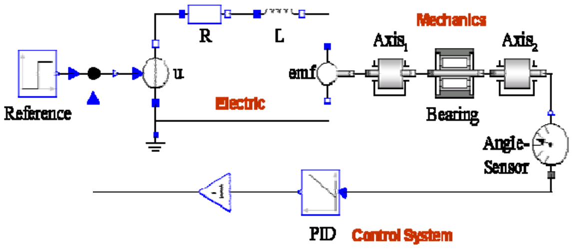
- Robotics
- Automotive
- Aircrafts
- Satellites
- Power plants
- Systems biology





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:

Available at: www.modelica.org

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

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*:

R*i = v;

can be used in three ways:

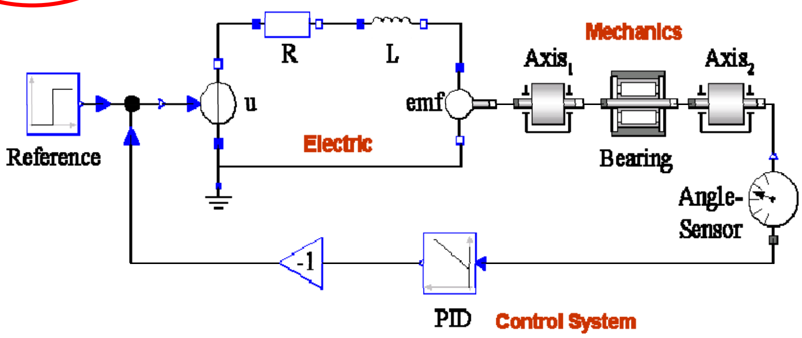
i := v/R;
v := R*i;
R := v/i;

What is Special about Modelica?

- Multi-Domain Modeling
- Visual acausal hierarchical component modeling
- Typed declarative equation-based textual language
- Hybrid modeling and simulation

What is Special about Modelica?

Multi-Domain Modeling



What is Special about Modelica?

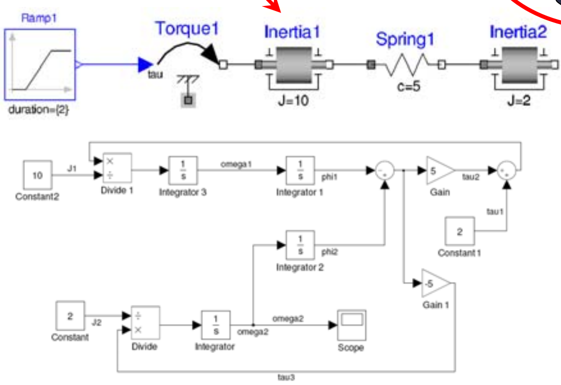
Multi-Domain Modeling

Visual Acausal Hierarchical Component Modeling

Keeps the physical structure

Acausal model (Modelica)

Causal block-based model (Simulink)

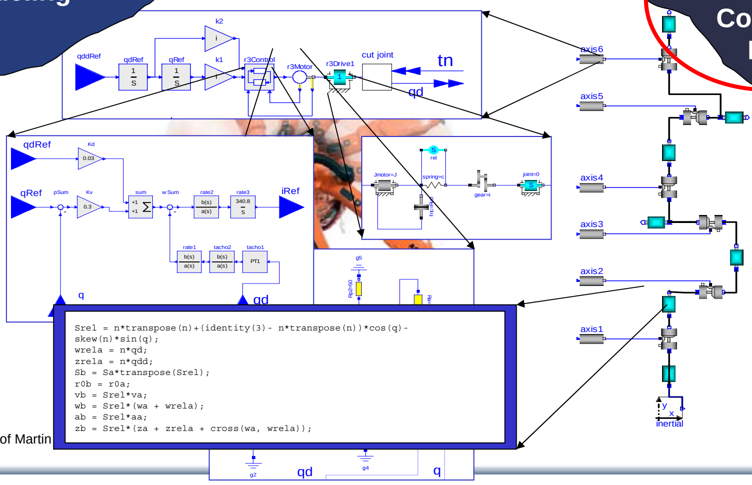


What is Special about Modelica?

Multi-Domain Modeling

Visual Acausal Hierarchical Component Modeling

Hierarchical system modeling



Courtesy of Martin Otter

What is Special about Modelica?

Multi-Domain Modeling

Visual Acausal Hierarchical Component Modeling

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

Behaviour described declaratively using

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

Variable declarations

```
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;
```

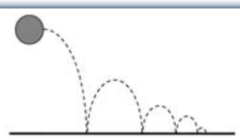
Differential equations

Typed Declarative Equation-based Textual Language

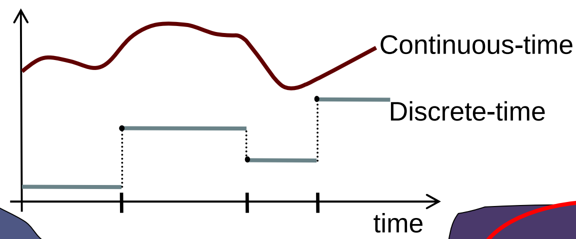
What is Special about Modelica?

Multi-Domain Modeling

Visual Acausal Component Modeling



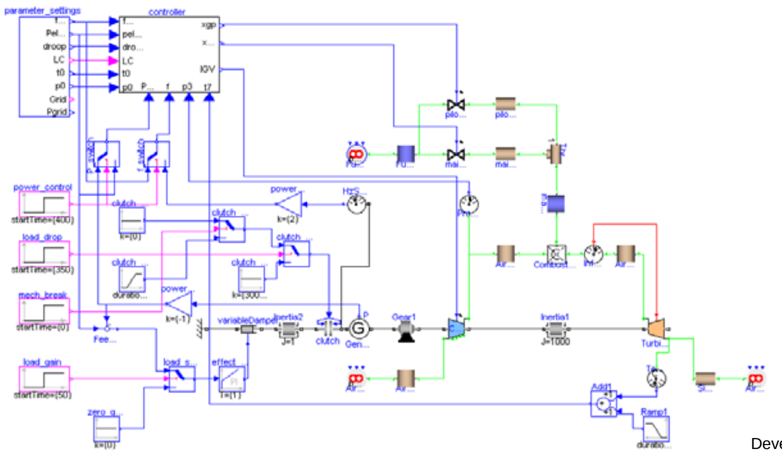
Hybrid modeling =
continuous-time + discrete-time modeling



Typed Declarative Equation-based Textual Language

Hybrid Modeling

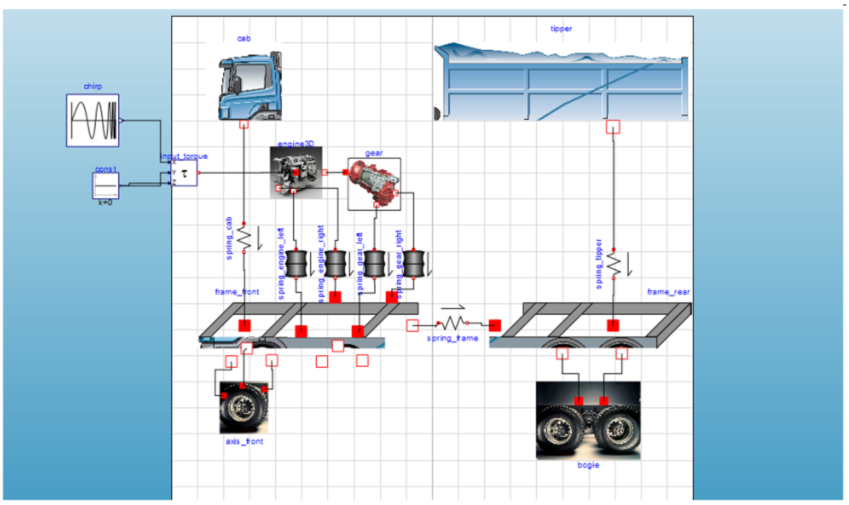
Modelica in Power Generation
GTX Gas Turbine Power Cutoff Mechanism



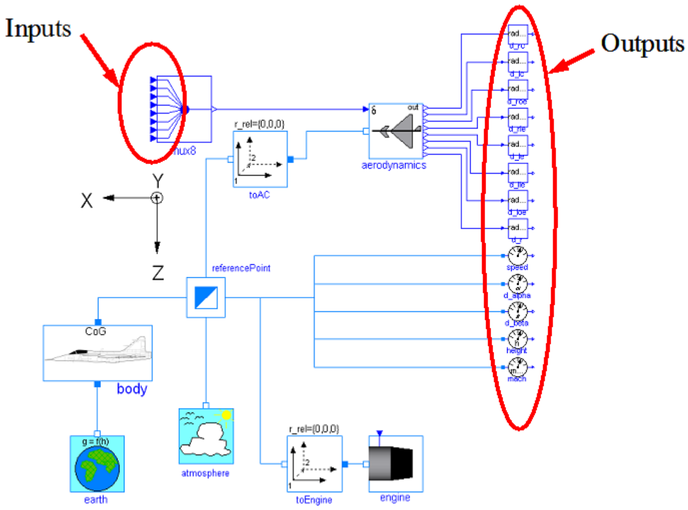
Developed
by MathCore
for Siemens

Courtesy of Siemens Industrial Turbomachinery AB

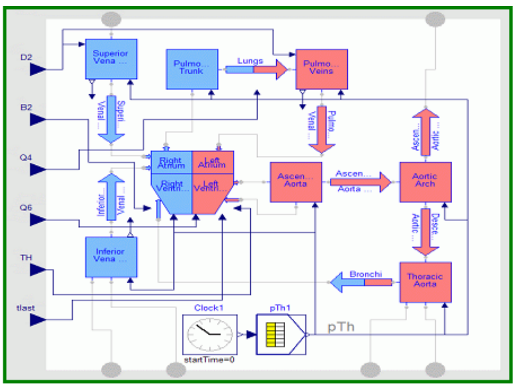
Modelica in Automotive Industry

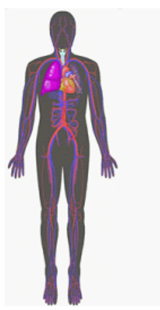


Modelica in Avionics



Modelica in Biomechanics





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
- Modelica Association established 2000
  - Open, non-profit organization

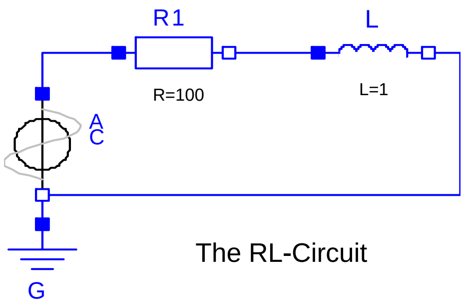
Modelica Conferences

- 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

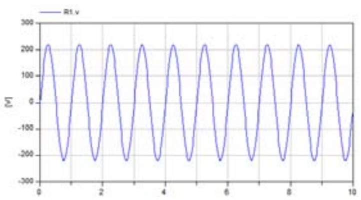
Exercises Part I
Hands-on graphical modeling
(15 minutes)

Exercises Part I – Basic Graphical Modeling

- (*See instructions on next two pages*)
- Start the simForge editor
- Draw the RL-Circuit
- Simulate



The RL-Circuit



Simulation

Exercises Part I – simForge Instructions Page 1

- Start simForge, (e.g. SimForge-0.8.4.1.jar).
- Under Tools pulldown menu, check if correct paths to OpenModelica.
- Go to **File** menu and choose **New Project**.
- Write *RL_Circuit* and click on the **Browse** button for choosing the destination folder.
- Press **OK.**
- In the navigation bar in the left, there should be three items, **Modelica, IEC61131-3** and **Simulation result**. Double-click on the **Modelica.**

- Under the **Modelica :**
  - The standard Modelica library components are listed in the **Used external package.**
  - The **Modelica classes** and **Modelica files** are the places where your models will end up under. The first folder is for the graphical models and the latter is for the texual form.

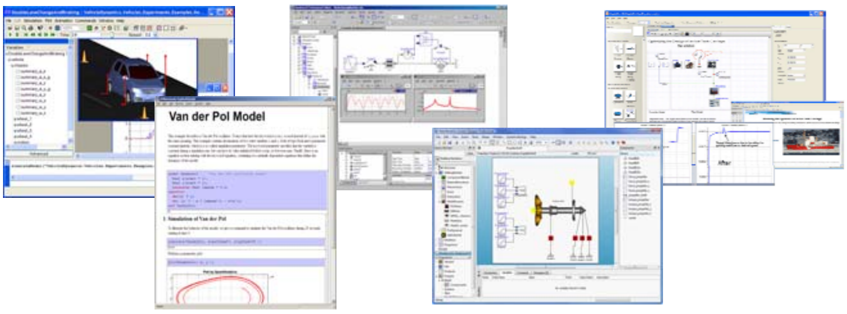
Exercises Part I – simForge Instructions Page 2

- Go to **File** menu and choose **New File.** Write *RL_circuit* and press **OK.**
- In the **Add Class** pop-up dialog box change the **Type** from **package** to **class** and press **OK.**
- Double click on the *RL_circuit* under the **Modelica classes** and the graphical window will appear.
- **Drag and Drop** components from the standard Modelica library to your model.
- For connecting components, move the cursor to the target pin and press shift+click ***once*** and just move the cursor with the mouse to the destination pin and click. Intermediate clicks for chaning line direction.
- Start the simulation with **simulation** button.
- In the simulation pop-up you can leave out some fields like the **Stop time**, which will result in a default value of 1 sec. will be used.
- The result will appear under the **Simulation result**, left bottom**.**
- To plot, click on the model (and its parts) under simulation result and tick the variables you would like to have plotted.

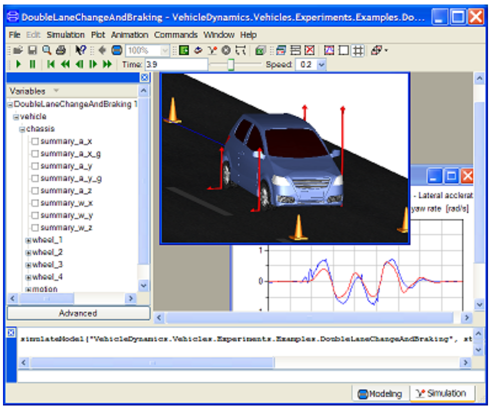
- Under the **Edit** menu -> **Advanced properties** you can tick the **visible legend** bar.

Part II

Modelica environments and OpenModelica

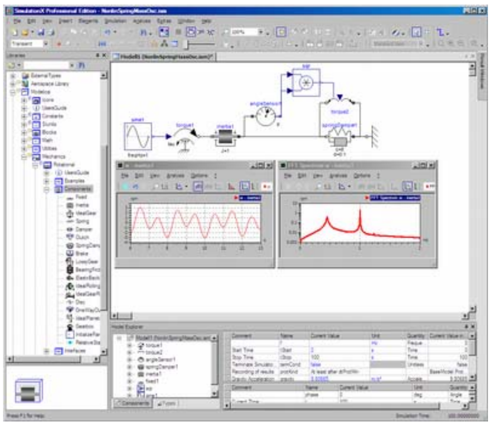


Dymola



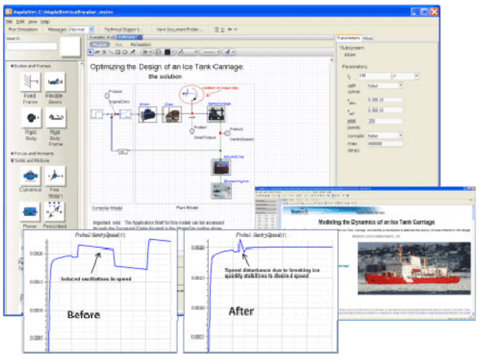
- Dynasim (Dassault Systemes)
- Sweden
- First Modelica tool on the market
- Main focus on automotive industry
- www.dynasim.com

Simulation X



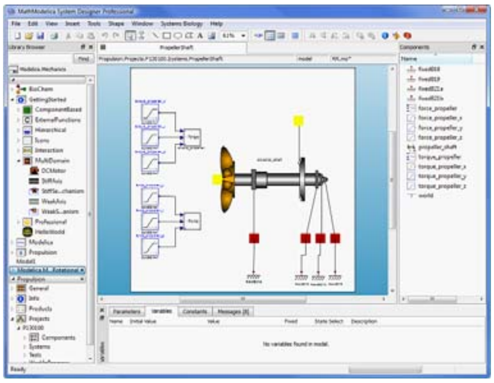
- ITI
- Germany
- Mechatronic systems
- www.simulationx.com

MapleSim



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

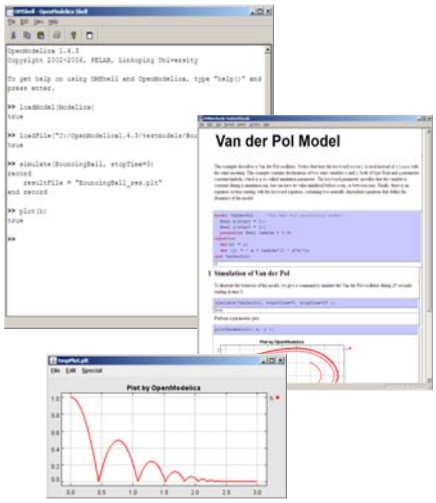
MathModelica



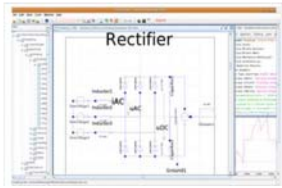
- MathCore
- Sweden
- Released 2006
- General purpose
- Mathematica connection
- www.mathcore.com

The OpenModelica Environment
www.OpenModelica.org

OpenModelica and simForge



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

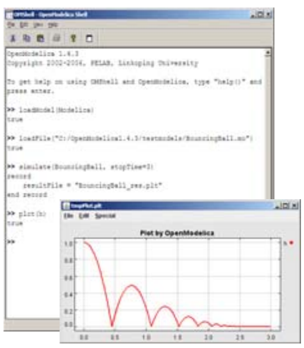


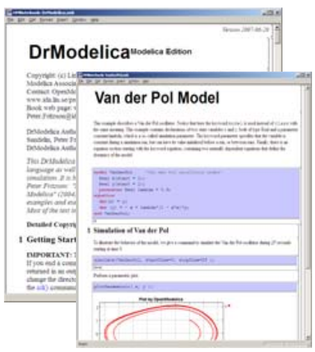
- Graphical editor simForge
- Politecnico di Milano, Italy
- Runs together with OpenModelica
- Open source

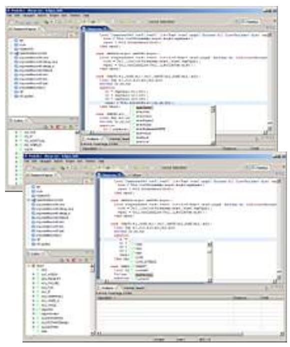
OpenModelica

- 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







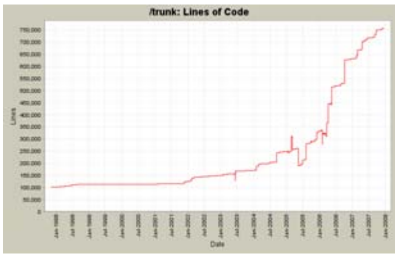
Open Source Modelica Consortium

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



Industrial members (14)

- **ABB Corporate Research**
- **Bosch-Rexroth AG, Germany**
- **Siemens Turbo Machinery AB**
- **Creative Connections, Prague**
- **Equa Simulation AB, Sweden**
- **IFP, Paris, France**
- **InterCAX, Atlanta, USA**
- **MostforWater, Belgium**
- **MathCore Engineering AB**
- **MapleSoft, Canada**
- **TLK Thermo, Germany**
- **VI-grade, Italy**
- **VTT, Finland**
- **XRG Simulation AB, Germany**

University members (11)

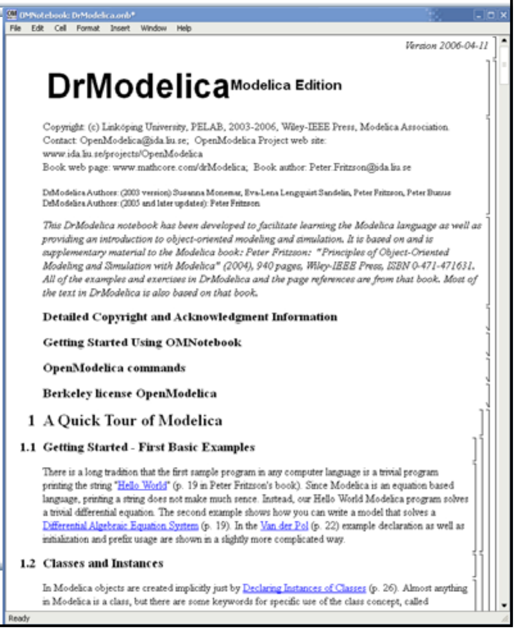
- **Linköping University, Sweden**
- **Hamburg University of Technology/TuTech, Germany**
- **Technical University of Braunschweig, Germany**
- **Université Laval, the modelEAU group, Canada**
- **Griffith University, Australia**
- **University of Queensland, Australia**
- **Politecnico di Milano, Italy**
- **Mälardalen University, Sweden**
- **Technical University Dortmund, Germany**
- **Technical University Dresden, Germany**
- **Telemark University College, Norway**

OMNotebook Electronic Notebook with DrModelica

- Primarily for teaching
- Interactive electronic book
- Platform independent

Commands:

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



File Edit Cell Format Insert Window Help

Version 2006-04-11

DrModelica Modelica Edition

Copyright: (c) Linköping University, PELAB, 2003-2006, Wiley-IEEE Press, Modelica Association.
Contact: OpenModelica@ida.liu.se; OpenModelica Project web site: www.ida.liu.se/projects/OpenModelica
Book web page: www.mathcore.com/drModelica; Book author: Peter.Fritzson@ida.liu.se

DrModelica Authors: (2003 version) Susanna Monemar, Eva-Lena Lengquist Sandelin, Peter Fritzson, Peter Bunus
DrModelica Authors: (2005 and later updates): Peter Fritzson

This DrModelica notebook has been developed to facilitate learning the Modelica language as well as providing an introduction to object-oriented modeling and simulation. It is based on and is supplementary material to the Modelica book: Peter Fritzson: "Principles of Object-Oriented Modeling and Simulation with Modelica" (2004), 940 pages, Wiley-IEEE Press, ISBN 0-471-471631. All of the examples and exercises in DrModelica and the page references are from that book. Most of the text in DrModelica is also based on that book.

Detailed Copyright and Acknowledgment Information

Getting Started Using OMNotebook

OpenModelica commands

Berkeley license OpenModelica

1 A Quick Tour of Modelica

1.1 Getting Started - First Basic Examples

There is a long tradition that the first sample program in any computer language is a trivial program printing the string "Hello World" (p. 19 in Peter Fritzson's book). Since Modelica is an equation based language, printing a string does not make much sence. Instead, our Hello World Modelica program solves a trivial differential equation. The second example shows how you can write a model that solves a Differential Algebraic Equation System (p. 19). In the Van der Pol (p. 22) example declaration as well as initialization and prefix usage are shown in a slightly more complicated way.

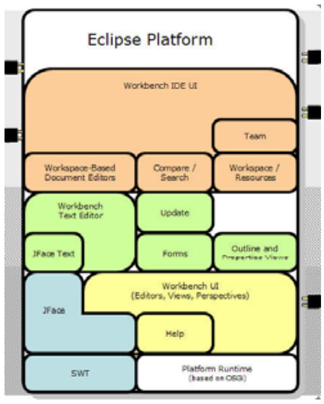
1.2 Classes and Instances

In Modelica objects are created implicitly just by Declaring Instances of Classes (p. 26). Almost anything in Modelica is a class, but there are some keywords for specific use of the class concept, called

Ready

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)

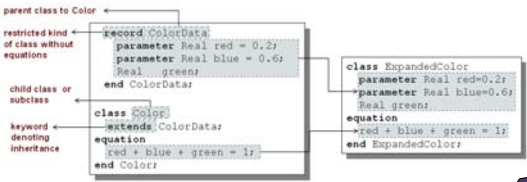


OpenModelica – Recent Developments

- Dec 2008. OSMC Board decides to focus on improving the OpenModelica compiler for Modelica libraries during 2009
- Dec 2008. MathCore contributes 1 man-year worth of source code for the flattening frontend.
- Jan-Sept 2009. Development mostly on the compiler frontend
- Sept 2009. OpenModelica release 1.5, containing approx 2 man-years development compared to version 1.4.5. (Beta release available today).

Part III

Modelica language concepts
and textual modeling

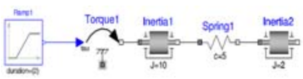
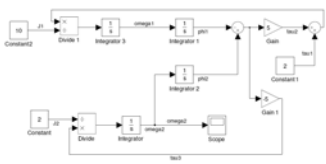


Typed Declarative Equation-based Textual Language

Hybrid Modeling

Acausal Modeling

The order of computations is not decided at modeling time

	Acausal	**Causal**
Visual Component Level		

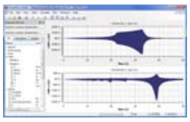
Equation Level	A resistor *equation*:
	`R*i = v;`
		Causal possibilities:
		`i := v/R;`
		`v := R*i;`
		`R := v/i;`

Typical Simulation Process

"Static" semantics / compile time

"Dynamic" semantics / run time

Modelica model → *Elaboration* → Hybrid DAE → *Equation Transformation & Code generation* → Executable → *Simulation* → Simulation Result



Simple model - Hello World!

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

Name of model

Initial condition

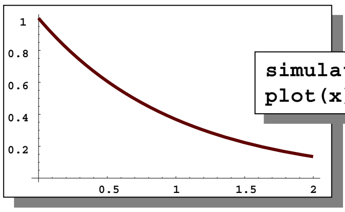
Continuous-time variable

Parameter, constant during simulation

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

Differential equation

Simulation in OpenModelica environment




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


Modelica Variables and Constants

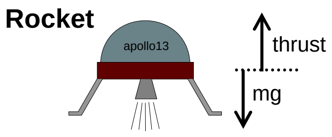
- Built-in primitive data types

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

- Parameters are constant during simulation
- Two types of constants in Modelica
  - **constant**
  - **parameter**

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


A Simple Rocket Model



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

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

$$altitude' = velocity$$

$$velocity' = acceleration$$

```
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

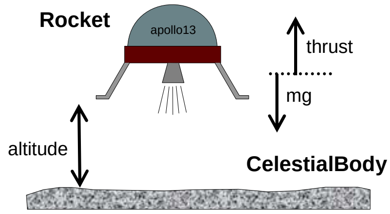
```
class CelestialBody
  constant  Real    g = 6.672e-11;
  parameter Real    radius;
  parameter String  name;
  parameter Real    mass;
end CelestialBody;
```

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



$$apollo.gravity = \frac{moon.g \cdot moon.mass}{(apollo.altitude + moon.radius)^2}$$

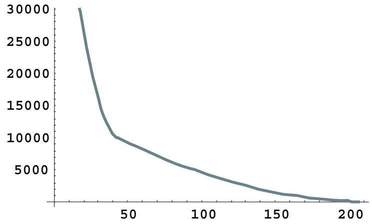
only access inside the class

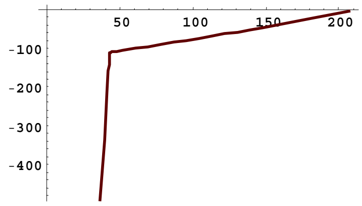
access by dot notation outside the class

```
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;
```


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

Restricted Class Keywords

- Classes can also be declared with other keywords, e.g.: `model`, `record`, `block`, `connector`, `function`, ...
- Classes declared with such keywords have restrictions
- Restrictions apply to the contents of restricted classes
- After Modelica 3.0 the `class` keyword means the same as `model`

- Example: 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 special kind of restricted class with some 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

parent class to Color

restricted kind of class without equations

child class or subclass

keyword denoting inheritance

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

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

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

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

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

```
class Color
  parameter Real red=0.2;
  parameter Real blue=0.6;
  Real green;
equation
  red + blue + green = 1;
end Color;
```

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

multiple inheritance

```
class ColoredPointWithoutInheritance
  Real x;
  Real y, z;
  parameter Real red = 0.2;
  parameter Real blue = 0.6;
  Real green;
equation
  red + blue + green = 1;
end ColoredPointWithoutInheritance;
```

Equivalent to

Multiple Inheritance cont'

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

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

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

Diamond Inheritance

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

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


Simple Class Definition

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

```
class SameColor = Color;
```

Equivalent to:

inheritance

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

- Often used for introducing new names of types:

```
type Resistor = Real;
```

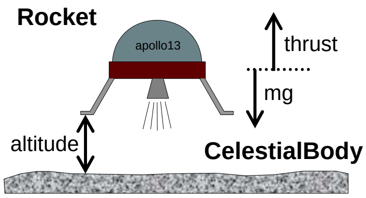
```
connector MyPin = Pin;
```


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)

inherited parameters

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


Inheritance of Protected Elements

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

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

```
class Color
  Real red;
  Real blue;
  Real green;
equation
  red + blue + green = 1;
end Color;
```

```
class ColoredPoint
 protected
  extends Color;
 public
  extends Point;
end ColoredPoint;
```

Equivalent to

```
class ColoredPointWithoutInheritance
  Real x;
  Real y,z;
  protected Real red;
  protected Real blue;
  protected Real green;
equation
  red + blue + green = 1;
end ColoredPointWithoutInheritance;
```

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
- Open Exercises-ModelicaTutorial.pdf (also available in printed handouts)

Exercises 2.1 and 2.2

- 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.

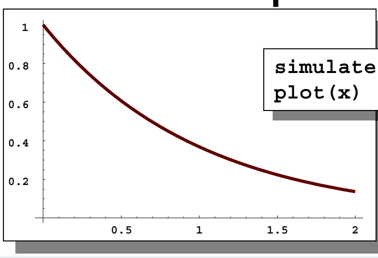
Exercise 2.1 – Hello World!

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

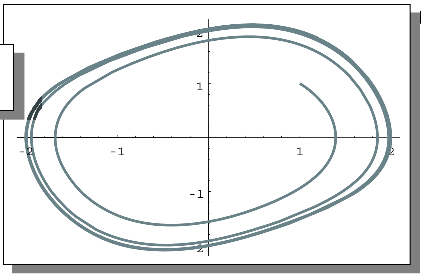



```
simulate(HelloWorld, stopTime = 2)
plot(x)
```


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

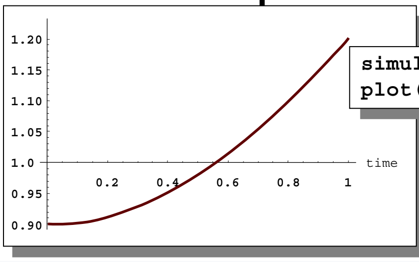
Include algebraic equation

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

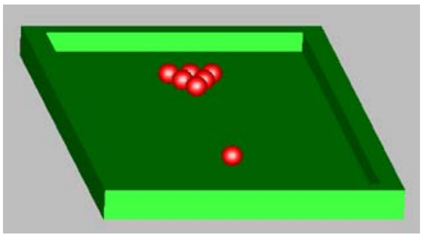
$$\dot{x} = 2 * x * y - 3 * x$$
$$\dot{y} = 5 * y - 7 * x * y$$
$$x(0) = 2$$
$$y(0) = 3$$

Exercise 2.5 – Functions (if you have time)

- 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`**.

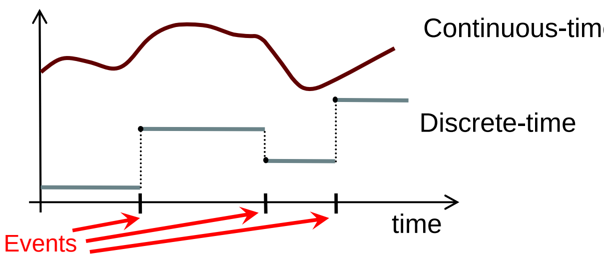
Discrete Events and Hybrid Systems



Picture: Courtesy Hilding Elmqvist

Hybrid Modeling

Hybrid modeling = continuous-time + discrete-time modeling




```
Real x;
Voltage v;
Current i;
```

```
discrete Real x;
Integer i;
Boolean b;
```

- 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

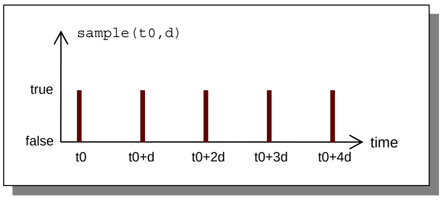
State event

```
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 `t0+i*d`, where `i=0,1, …`

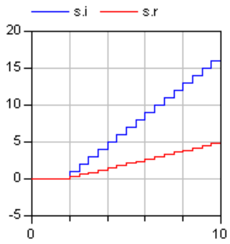


Variables need to be discrete

```
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

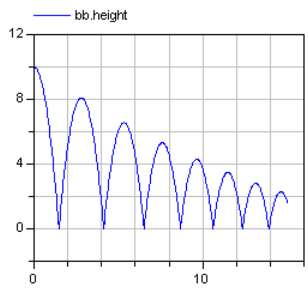
pre(...) takes the previous value before the event.



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;
```

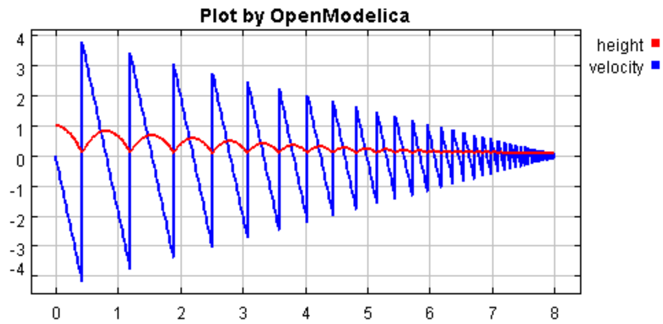


Initial conditions

Reinit ”assigns” continuous-time variable `velocity` a new value

Exercise 2.6 – BouncingBall

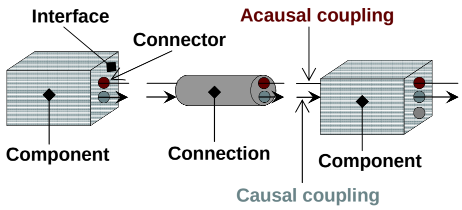
- Locate the BouncingBall model in one of the hybrid modeling sections of DrModelica (the When-Equations link in Section 2.9), run it, change it slightly, and re-run it.



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*

electrical connector

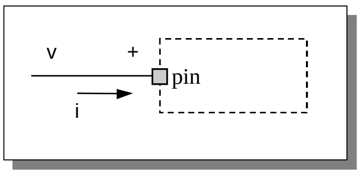
connector class

keyword `flow` indicates that currents of connected pins sum to zero.

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

Pin pin;
```

an instance `pin` of class `Pin`



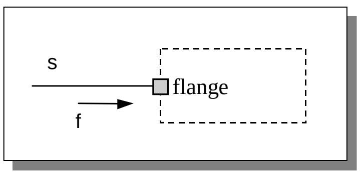
mechanical connector

connector class

```
connector Flange
  Position      s;
  flow Force    f;
end Flange;

Flange flange;
```

an instance `flange` of class `Flange`



The `flow` prefix

Two kinds of variables in connectors:

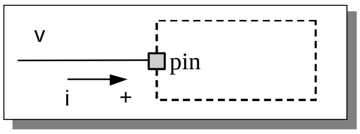
- *Non-flow variables potential* or energy level
- *Flow variables* represent some kind of flow

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

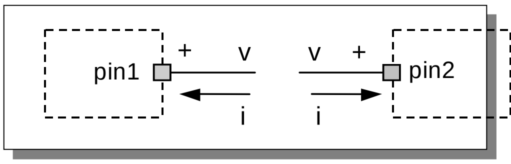
Domain Type	Potential	Flow	Carrier	Modelica Library
Electrical	Voltage	Current	Charge	`Electrical.Analog`
Translational	Position	Force	Linear momentum	`Mechanical.Translational`
Rotational	Angle	Torque	Angular momentum	`Mechanical.Rotational`
Magnetic	Magnetic potential	Magnetic flux rate	Magnetic flux	
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:

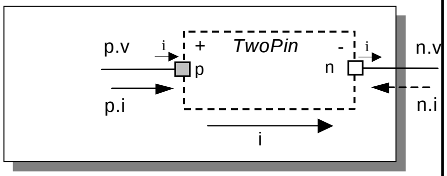
$$v_1 = v_2 = v_3 = \ldots v_n$$

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

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

Common Component Structure

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



partial 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
```

electrical connector class

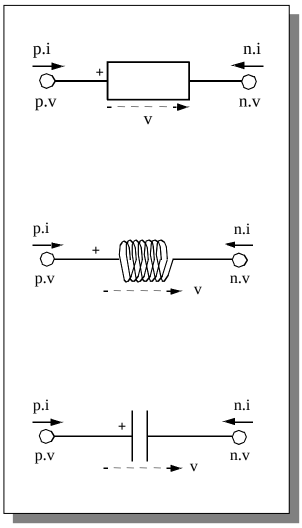
```
connector Pin
  Voltage      v;
  flow Current i;
end Pin;
```


Electrical Components

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

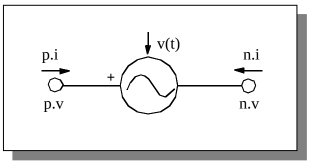
```
model Inductor "Ideal electrical inductor"
  extends TwoPin;
  parameter Real L "Inductance";
equation
  L*der(i) = v;
end Inductor;
```

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

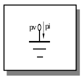


Electrical Components cont’

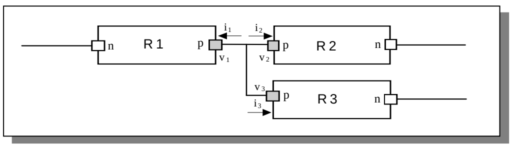
```
model Source
   extends TwoPin;
   parameter Real A,w;
equation
   v = A*sin(w*time);
end Resistor;
```




```
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

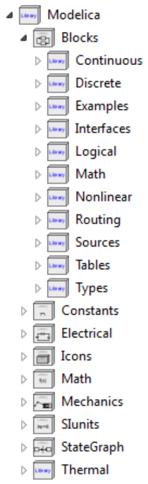
- *Modelica Standard Library* (called Modelica) is a standardized predefined package developed by Modelica Association
- It can be used freely for both commercial and noncommercial purposes under the conditions of *The Modelica License*.
- Modelica libraries are available online including documentation and source code from http://www.modelica.org/library/library.html

Modelica Standard Library cont'

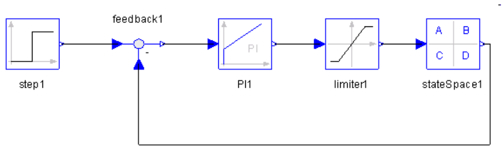
The Modelica Standard Library contains components from various application areas, including the following sublibraries:

• 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
• SIunits	Type definitions based on SI units according to ISO 31-1992
• Stategraph	Hierarchical state machines (analogous to Statecharts)
• Thermal	Components for thermal systems
• Utilities	Utility functions especially for scripting

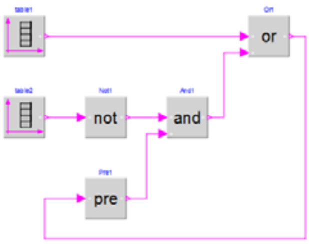
Modelica.Blocks



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



Examples:

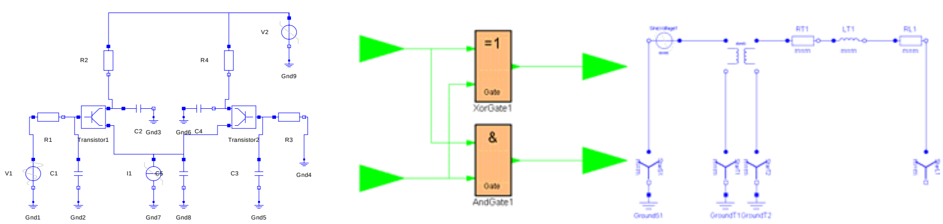


Modelica.Electrical

Electrical components for building analog, digital, and multiphase circuits

Analog   Digital   Machines   MultiPhase

Examples:

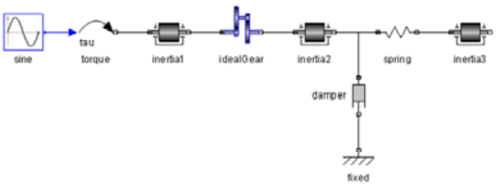


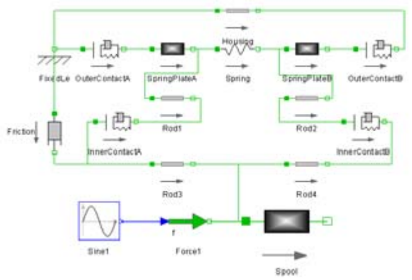
Modelica.Mechanics

Package containing components for mechanical systems

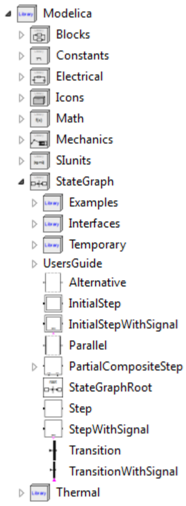
Subpackages:

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

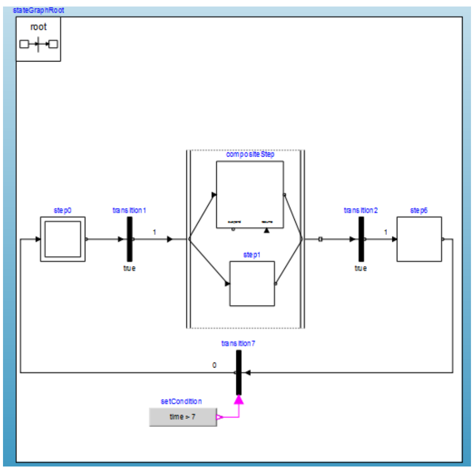




Modelica.Stategraph



Hierarchical state machines (similar to Statecharts)



Other Free Libraries

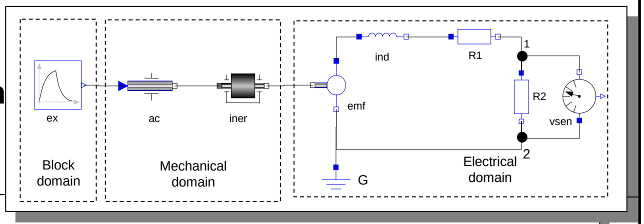
• WasteWater	Wastewater treatment plants, 2003
• ATPlus	Building simulation and control (fuzzy control included), 2005
• MotorCycleDymanics	Dynamics and control of motorcycles, 2009
• NeuralNetwork	Neural network mathematical models, 2006
• VehicleDynamics	Dynamics of vehicle chassis (obsolete), 2003
• 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
• ...	

Some Commercial Libraries

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

Connecting Components from Multiple Domains

- Block domain
- Mechanical domain
- Electrical domain

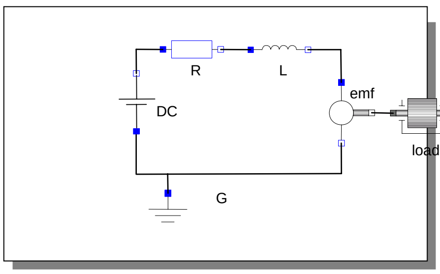



```
model Generator
  Modelica.Mechanics.Rotational.Accelerate ac;
  Modelica.Mechanics.Rotational.Inertia iner;
  Modelica.Electrical.Analog.Basic.EMF emf(k=-1);
  Modelica.Electrical.Analog.Basic.Inductor ind(L=0.1);
  Modelica.Electrical.Analog.Basic.Resistor R1,R2;
  Modelica.Electrical.Analog.Basic.Ground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

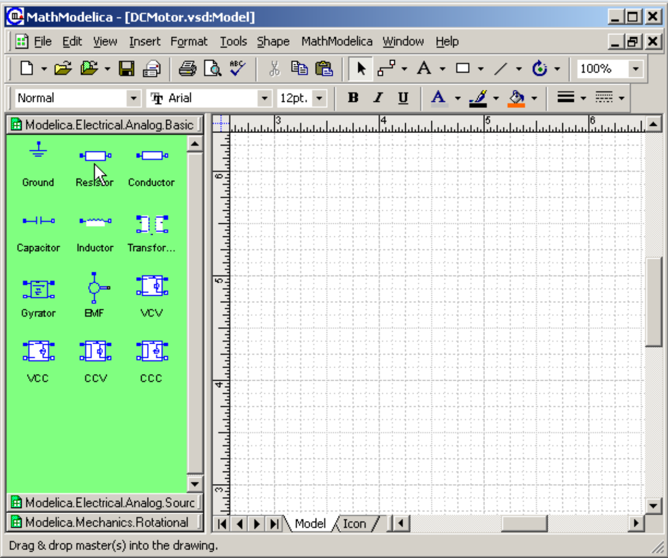
```
model DCMotor
   Resistor R(R=100);
   Inductor L(L=100);
   VsourceDC DC(f=10);
   Ground G;
   EMF emf(k=10,J=10, b=2);
   Inertia load;
equation
   connect(DC.p,R.n);
   connect(R.p,L.n);
   connect(L.p, emf.n);
   connect(emf.p, DC.n);
   connect(DC.n,G.p);
   connect(emf.flange,load.flange);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



Exercises Part IV
Graphical Modeling Exercises

using
simForge and OpenModelica

Graphical Modeling - Using Drag and Drop Composition

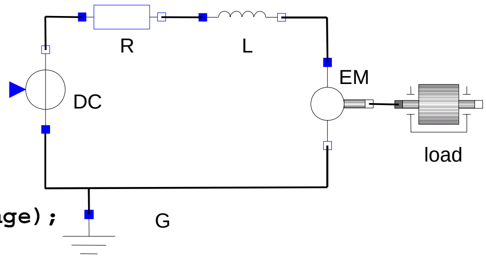


Courtesy
MathCore
Engineering AB

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 == 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)

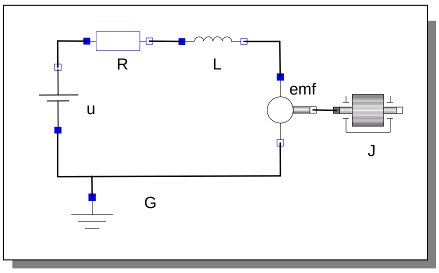
Automatic transformation to ODE or DAE for simulation:

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

Exercise 3.1

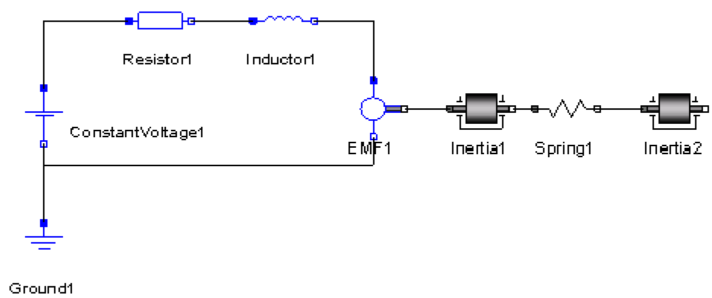
- Draw the `DCMotor` model using the graphic connection editor using models from the following Modelica libraries:
  `Mechanics.Rotational,`
  `Electrical.Analog.Basic,`
  `Electrical.Analog.Sources`

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



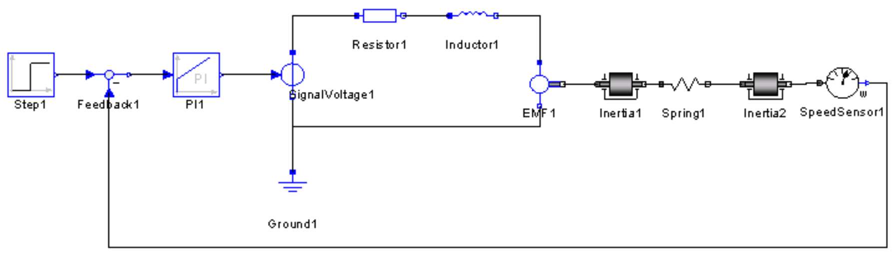
Exercise 3.2

- 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 simForge.

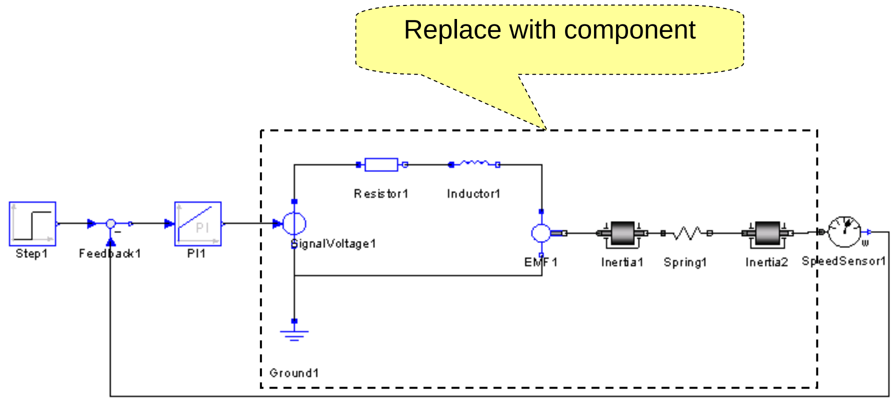


Live example – Graphical Modeling

- Building a component with icon

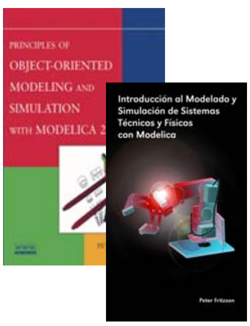
Exercise 3.4

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



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
  - Introducción al Modelado y Simulación de Sistemas Técnicos y Físicos con Modelica, Peter Fritzson
  - Introduction to Modelica, Michael Tiller



Summary

Multi-Domain Modeling

Visual Acausal Component Modeling

Typed Declarative Textual Language

Hybrid Modeling

Thanks for listening!

