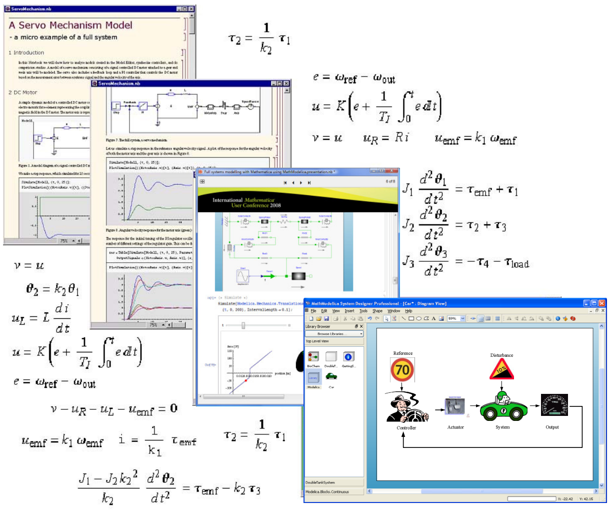
Modelica Tutorial – Modeling and Simulation with OpenModelica and MathModelica


A Servo Mechanism Model
- a micro example of a full system
1 Introduction
2 DC Motor
$\tau_2 = \frac{1}{k_2}\tau_1$
$e = \omega_{ref} - \omega_{out}$
$u = K\left(e + \frac{1}{T_I}\int_0^t e\,dt\right)$
$v = u \quad u_R = R\,i \quad u_{emf} = k_1\,\omega_{emf}$
$J_1 \frac{d^2\theta_1}{dt^2} = \tau_{emf} + \tau_1$
$J_2 \frac{d^2\theta_2}{dt^2} = \tau_2 + \tau_3$
$J_3 \frac{d^2\theta_3}{dt^2} = -\tau_4 - \tau_{load}$
$v = u$
$\theta_2 = k_2\theta_1$
$u_L = L\frac{di}{dt}$
$u = K\left(e + \frac{1}{T_I}\int_0^t e\,dt\right)$
$e = \omega_{ref} - \omega_{out}$
$v - u_R - u_L - u_{emf} = 0$
$u_{emf} = k_1\,\omega_{emf} \quad i = \frac{1}{k_1}\tau_{emf} \quad \tau_2 = \frac{1}{k_2}\tau_1$
$\frac{J_1 - J_2 k_2^2}{k_2}\frac{d^2\theta_2}{dt^2} = \tau_{emf} - k_2\tau_3$
Reference
Disturbance
70
Controller
Actuator
System
Output

Jan Brugård

MathCore Engineering AB
jan.brugard@mathcore.com

David Broman

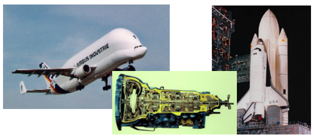
Linköping University
Department of Computer and Information Science
davbr@ida.liu.se

Agenda

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

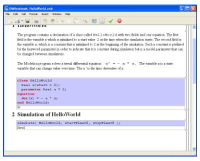
Part I

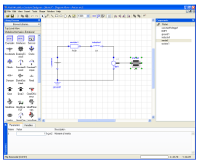
Introduction to Modelica and a demo example



Part II

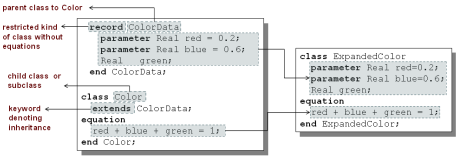
Modelica environments





Part III

Modelica language concepts and textual modeling

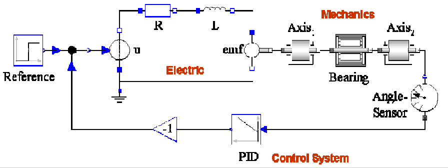

parent class to Color
restricted kind of class without equations
child class or subclass
keyword denoting inheritance

```
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;
```

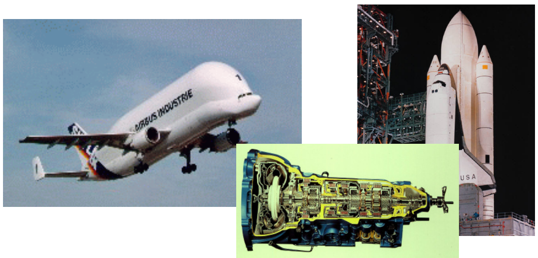
Part IV

Graphical modeling and the Modelica standard library



Part I

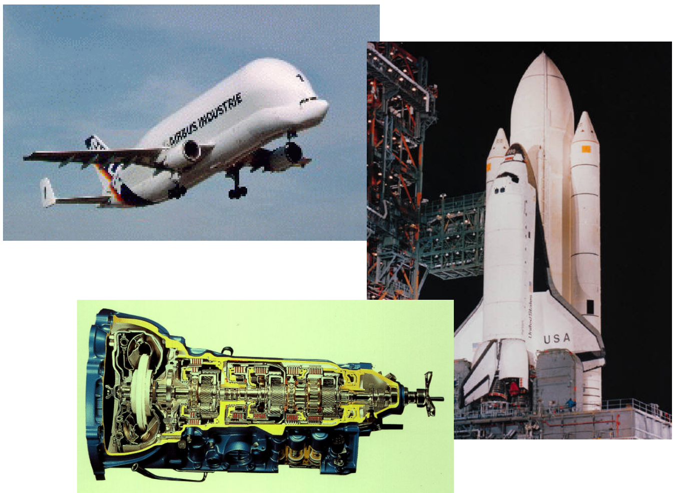
Introduction to Modelica and a demo example



What is Modelica?

A language for modeling of complex physical systems

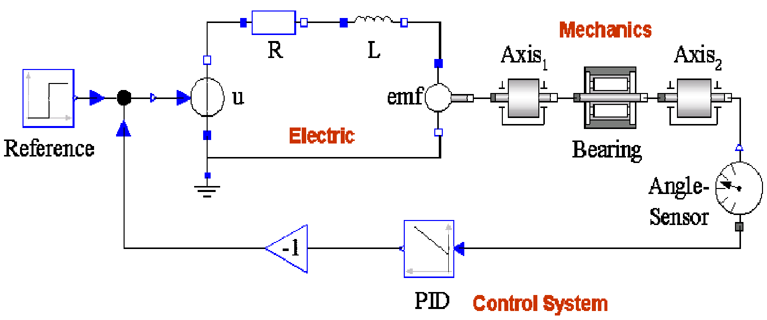
- Robotics
- Automotive
- Aircrafts
- Satellites
- Power plants
- Systems biology



What is Modelica?

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

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?

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

A language for modeling of complex physical systems

i.e., Modelica is **not** a tool

Free, open language specification:

MODELICA

Modelica® - A Unified Object-Oriented Language for Physical Systems Modeling

Language Specification

Version 2.2
February 2, 2005

by the

Modelica Association

Abstract
This document defines the Modelica language, version 2.2, which is developed by the Modelica Association, a non-profit organization with seat in Linköping, Sweden. Modelica is a freely available, object-oriented language for modeling of large, complex, and heterogeneous physical systems. It is suited for multi-domain modeling, for example, mechatronic models in robotics, automotive and aerospace applications involving mechanical, electrical, hydraulic and control subsystems, process oriented applications and generation and distribution of electric power. Models in Modelica are mathematically described by differential, algebraic and discrete equations. No particular variable needs to be solved for manually. A Modelica tool will have enough information to decide that automatically. Modelica is designed such that available, specialized algorithms can be utilized to enable efficient handling of large models having more than one hundred thousand equations. Modelica is suited and used for hardware-in-the-loop simulations and for embedded control systems. More information is available at **http://www.Modelica.org/**

Modelica® is a registered trademark of the "Modelica Association".

Available at: www.modelica.org

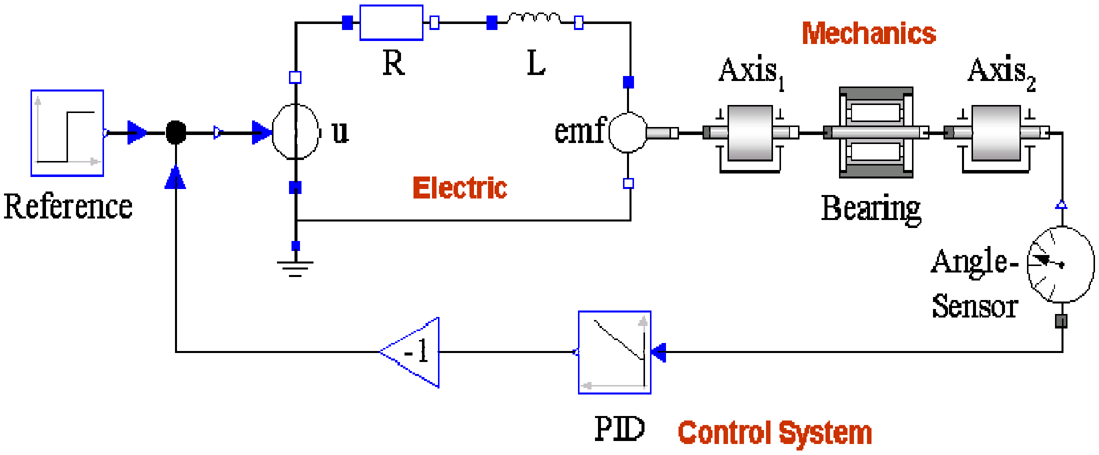
There exist several free and commercial tools, for example:

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

What is special about Modelica?

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

Multi-Domain Modeling



What is special about Modelica?

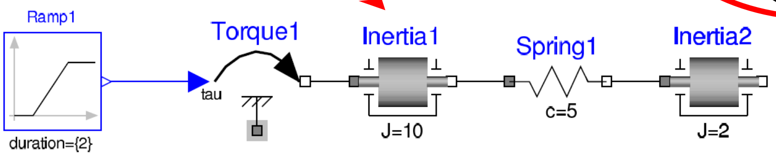
davbr@ida.liu.se
jan.brugard@mathcore.com

Multi-Domain Modeling

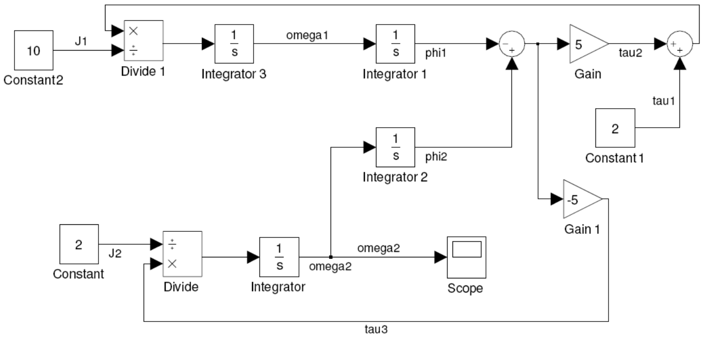
Visual Acausal Component Modeling

Keeps the physical structure

Acausal model (Modelica)



Causal block-based model (Simulink)



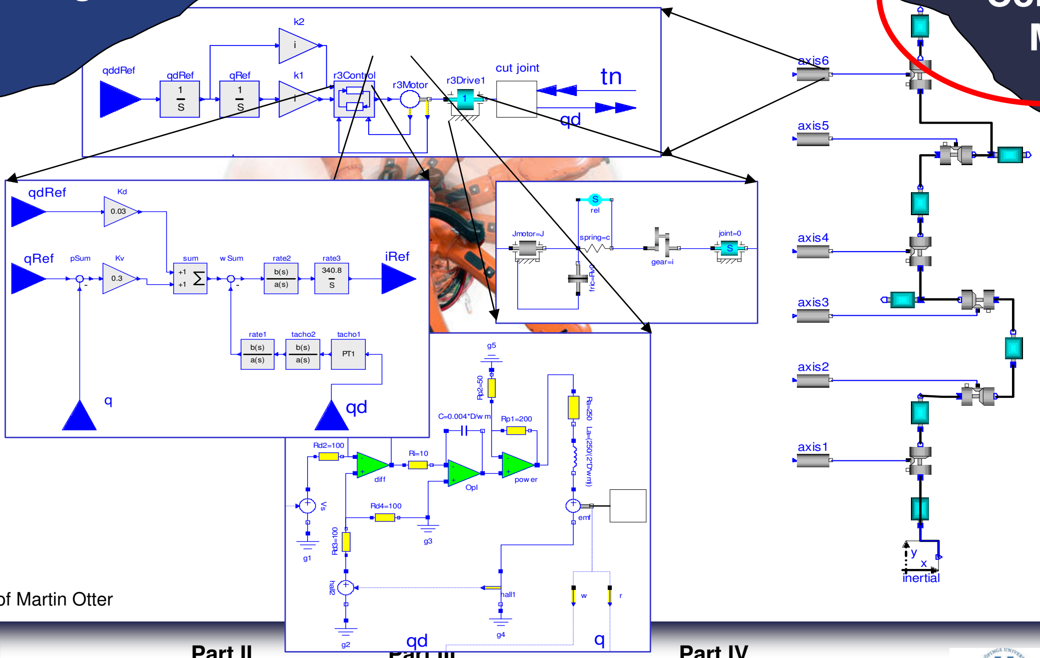
What is special about Modelica?

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

Multi-Domain Modeling

Hierarchical system modeling

Visual Acausal Component Modeling



Courtesy of Martin Otter

Courtesy of Martin Otter

What is special about Modelica?

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

Multi-Domain Modeling

Visual Acausal 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;
```

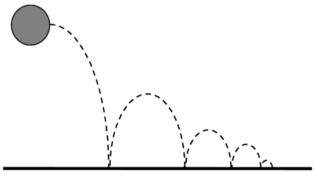
Differential equations

Typed Declarative Textual Language

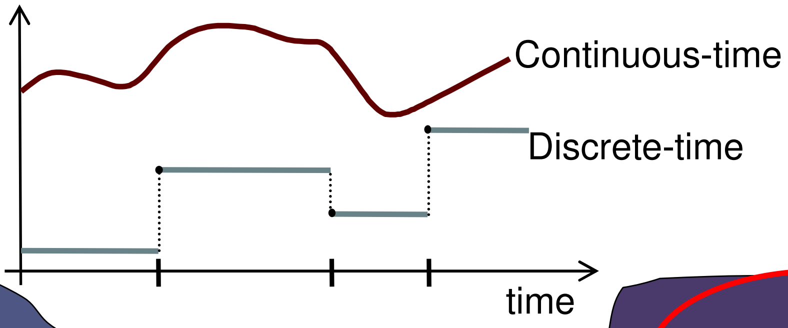
What is special about Modelica?

Multi-Domain Modeling

Visual Acausal Component Modeling



Hybrid modeling =
continuous-time + discrete-time modeling



Typed Declarative Textual Language

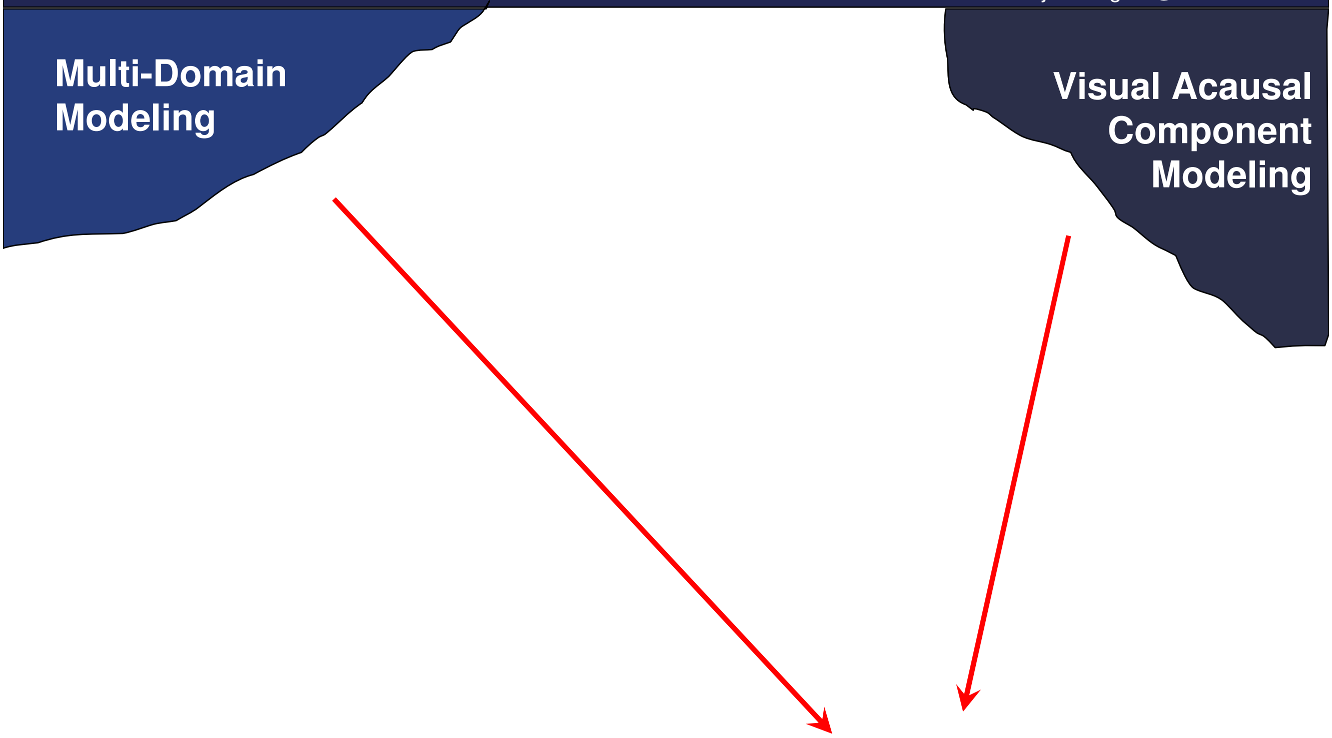
Hybrid Modeling

Coming up in part III

Typed Declarative Textual Language

Hybrid Modeling

Coming up in part IV



Brief Modelica History

Modelica design group meetings

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

Modelica Language Versions

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

Modelica Association established 2000

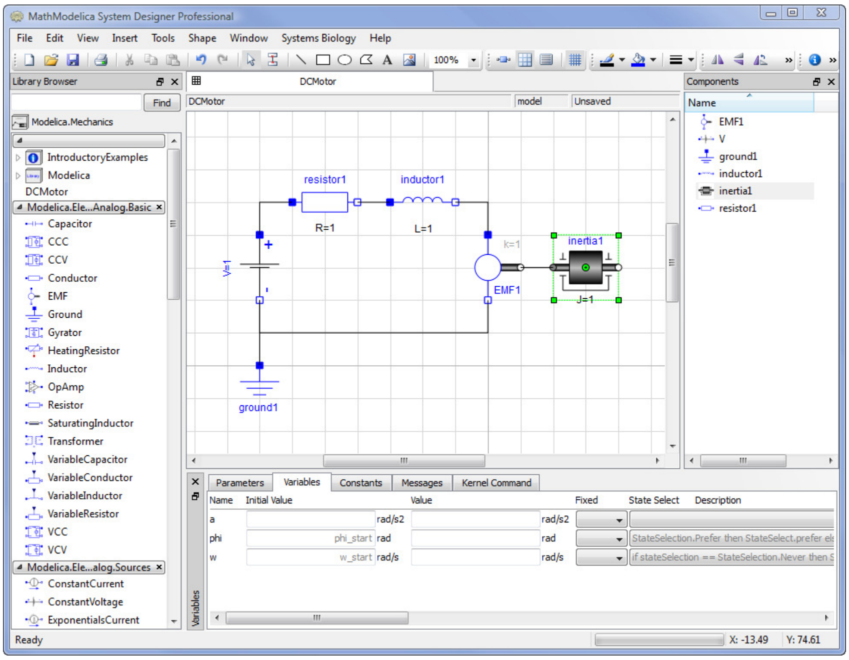
- Open, non-profit organization

Modelica Conferences

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

Introductory example

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

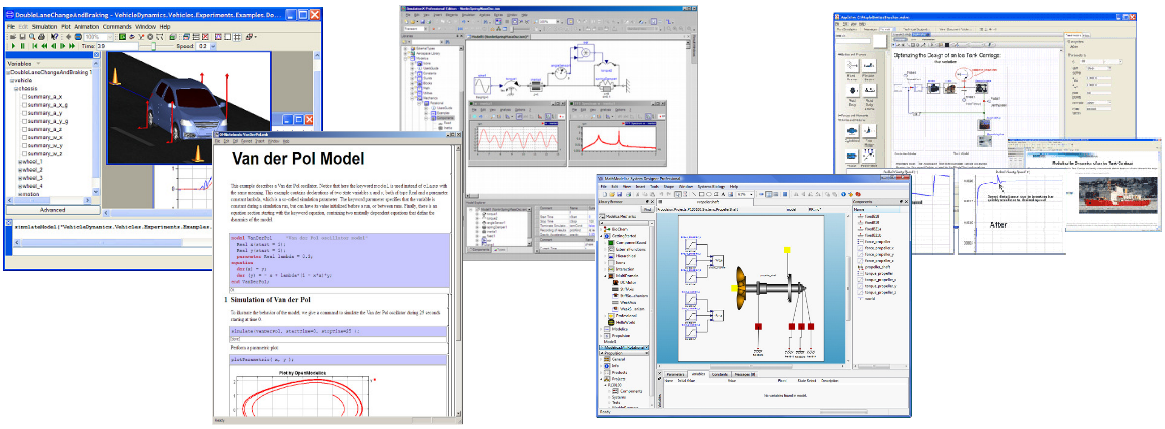


Name	Initial Value		Value		Fixed	State Select	Description
a		rad/s2		rad/s2			
phi	phi_start	rad		rad		StateSelection.Prefer then StateSelect.prefer el	
w	w_start	rad/s		rad/s		if stateSelection == StateSelection.Never then S	

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

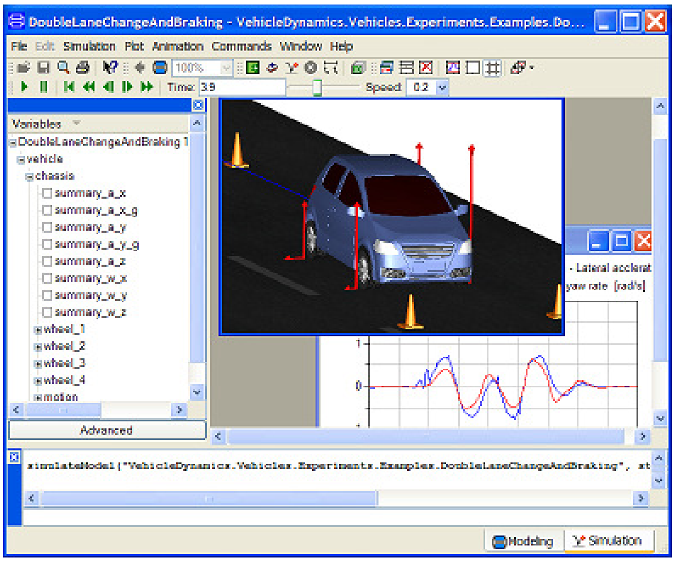
Part II

Modelica environments


DoubleLaneChangeAndBraking - VehicleDynamics.Vehicles.Experiments.Examples.Do...
Van der Pol Model
Simulation of Van der Pol
Plot by OpenModelica
Optimizing the Design of an Ice Tank Carriage
After

Dymola

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

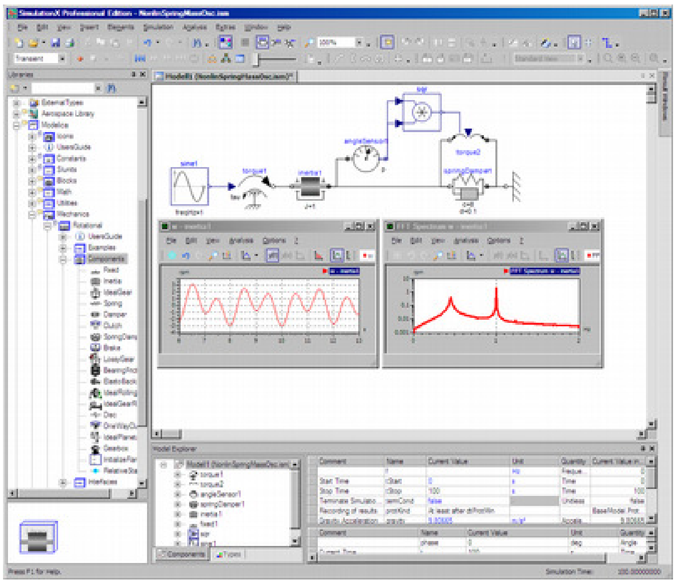


```
simulateModel("VehicleDynamics.Vehicles.Experiments.Examples.DoubleLaneChangeAndBraking", st
```

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

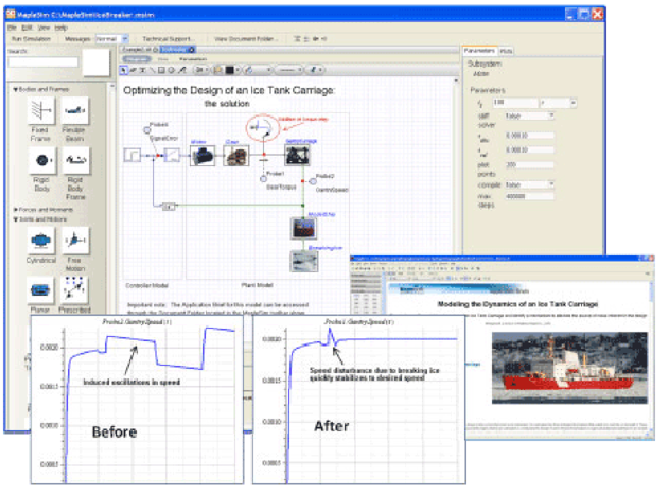
Simulation X

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



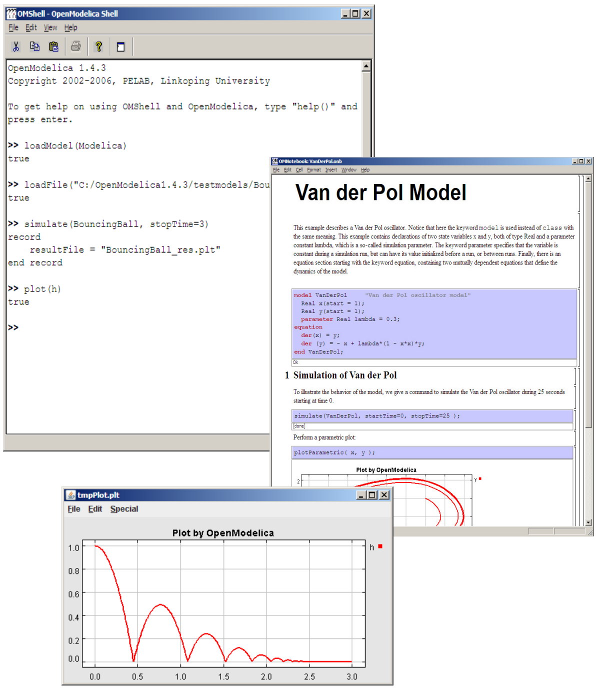
- ITI
- Germany
- Mechatronic systems
- www.simulationx.com

MapleSim


Optimizing the Design of an Ice Tank Carriage
the solution
Induced oscillations in speed
Before
Speed disturbance due to breaking ice quickly stabilizes to desired speed
After
Modeling the Dynamics of an Ice Tank Carriage

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

OpenModelica



```
OpenModelica 1.4.3
Copyright 2002-2006, PELAB, Linkoping University

To get help on using OMShell and OpenModelica, type "help()" and
press enter.

>> loadModel(Modelica)
true

>> loadFile("C:/OpenModelica1.4.3/testmodels/Bou
true

>> simulate(BouncingBall, stopTime=3)
record
    resultFile = "BouncingBall_res.plt"
end record

>> plot(h)
true

>>
```

Van der Pol Model

This example describes a Van der Pol oscillator. Notice that here the keyword `model` is used instead of `class` with the same meaning. This example contains declarations of two state variables x and y, both of type Real and a parameter constant lambda, which is a so-called simulation parameter. The keyword parameter specifies that the variable is constant during a simulation run, but can have its value initialized before a run, or between runs. Finally, there is an equation section starting with the keyword equation, containing two mutually dependent equations that define the dynamics of the model.

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

Ok

1 Simulation of Van der Pol

To illustrate the behavior of the model, we give a command to simulate the Van der Pol oscillator during 25 seconds starting at time 0.

```
simulate(VanDerPol, startTime=0, stopTime=25 );
```

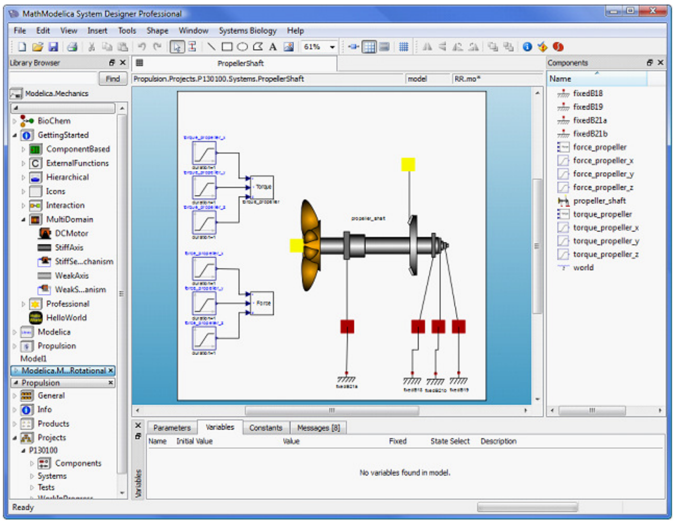
[done]

Perform a parametric plot:

```
plotParametric( x, y );
```

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

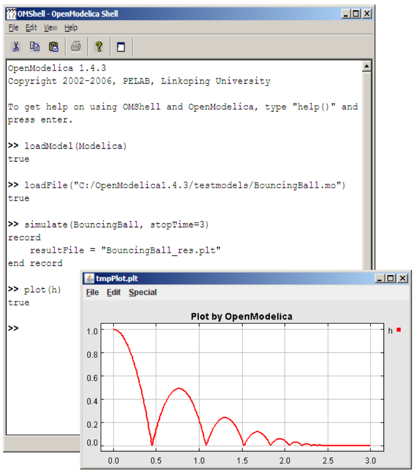
MathModelica



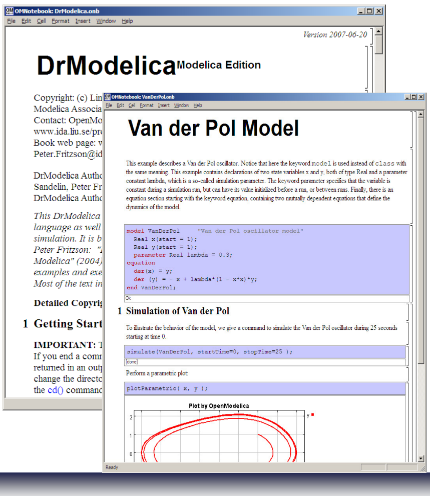
- MathCore
- Sweden
- Released 2006
- General purpose
- Mathematica connection
- www.mathcore.com

OpenModelica

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



```
OpenModelica 1.4.3
Copyright 2002-2006, PELAB, Linkoping University

To get help on using OMShell and OpenModelica, type "help()" and
press enter.

>> loadModel(Modelica)
true

>> loadFile("C:/OpenModelica1.4.3/testmodels/BouncingBall.mo")
true

>> simulate(BouncingBall, stopTime=3)
record
    resultFile = "BouncingBall_res.plt"
end record

>> plot(h)
true

>>
```


OMNotebook: DrModelica.onb
File Edit Cell Format Insert Window Help
OMNotebook: VanDerPol.onb
File Edit Cell Format Insert Window Help
Plot by OpenModelica
2
1
0
y
Ready

Version 2007-06-20

DrModelica Modelica Edition

Copyright: (c) Lin
Modelica Associa
Contact: OpenMo
www.ida.liu.se/pr
Book web page: w
Peter.Fritzson@id

DrModelica Autho
Sandelin, Peter Fr
DrModelica Autho

This DrModelica
language as well
simulation. It is b
Peter Fritzson: "
Modelica" (2004)
examples and exe
Most of the text in

Detailed Copyrig

1 Getting Start

IMPORTANT: T
If you end a comm
returned in an out
change the directo
the cd() command

Van der Pol Model

This example describes a Van der Pol oscillator. Notice that here the keyword model is used instead of class with the same meaning. This example contains declarations of two state variables x and y, both of type Real and a parameter constant lambda, which is a so-called simulation parameter. The keyword parameter specifies that the variable is constant during a simulation run, but can have its value initialized before a run, or between runs. Finally, there is an equation section starting with the keyword equation, containing two mutually dependent equations that define the dynamics of the model.

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

Ok

1 Simulation of Van der Pol

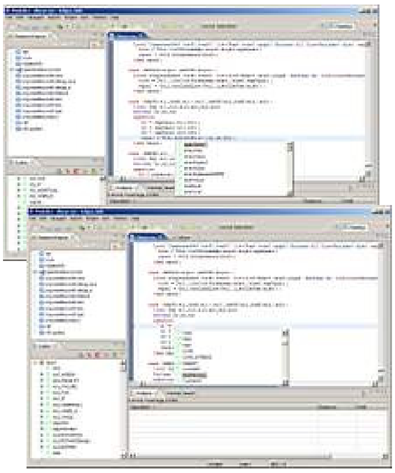
To illustrate the behavior of the model, we give a command to simulate the Van der Pol oscillator during 25 seconds starting at time 0.

```
simulate(VanDerPol, startTime=0, stopTime=25 );
```

[done]

Perform a parametric plot:

```
plotParametric( x, y );
```



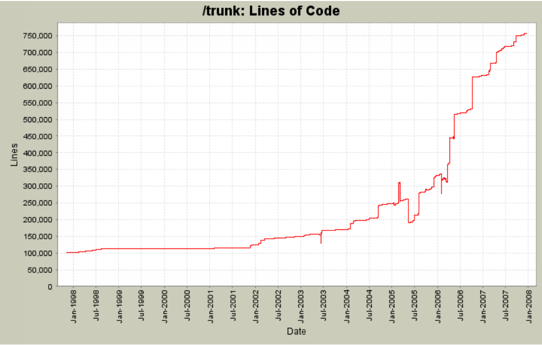
Open Source Modelica Consortium

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

Open-source community services

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

Code Statistics



Founded Dec 4, 2007

Industrial members

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

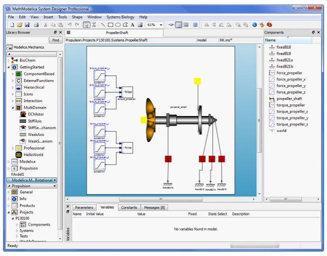
Academic members

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

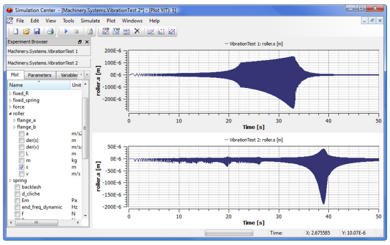
MathModelica Components

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

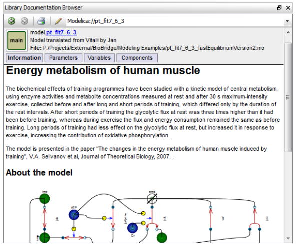
Modeling


MathModelica System Designer Professional
Library Browser
Components
Parameters
Variables
Constants
Messages
No variables found in model.

Simulation



Documentation



model pt_fit7_6_3
Model translated from Vitalii by Jan
File: P:/Projects/External/BioBridge/Modeling Examples/pt_fit7_6_3_fastEquilibriumVersion2.mo

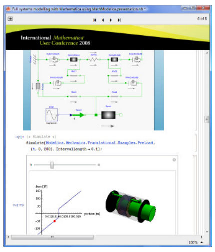
Energy metabolism of human muscle

The biochemical effects of training programmes have been studied with a kinetic model of central metabolism, using enzyme activities and metabolite concentrations measured at rest and after 30 s maximum-intensity exercise, collected before and after long and short periods of training, which differed only by the duration of the rest intervals. After short periods of training the glycolytic flux at rest was three times higher than it had been before training, whereas during exercise the flux and energy consumption remained the same as before training. Long periods of training had less effect on the glycolytic flux at rest, but increased it in response to exercise, increasing the contribution of oxidative phosphorylation.

The model is presented in the paper "The changes in the energy metabolism of human muscle induced by training", V.A. Selivanov et.al, Journal of Theoretical Biology, 2007, .

About the model

Analysis



MathModelica Editions

Lite

Ideal for small student projects. It is free for academic and personal use, but not available for industry.

Runs with, but is not part of OpenModelica

System Designer

Suited for **modeling and simulation projects** in industry and academia.

Stand-alone

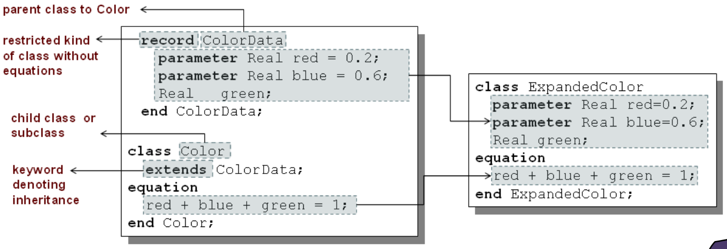
System Designer Professional

Targeted at **research in industry and academia**, offering unparalleled possibilities for analyzing results.

Includes a tight Mathematica connection

Part III

Modelica language concepts and textual modeling


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;
```

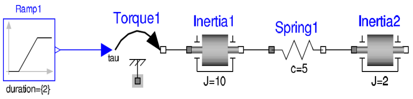
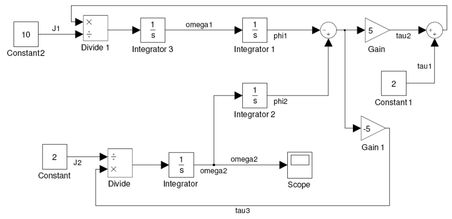
```
class ExpandedColor
  parameter Real red=0.2;
  parameter Real blue=0.6;
  Real green;
equation
  red + blue + green = 1;
end ExpandedColor;
```

Typed
Declarative
Textual Language

Hybrid
Modeling

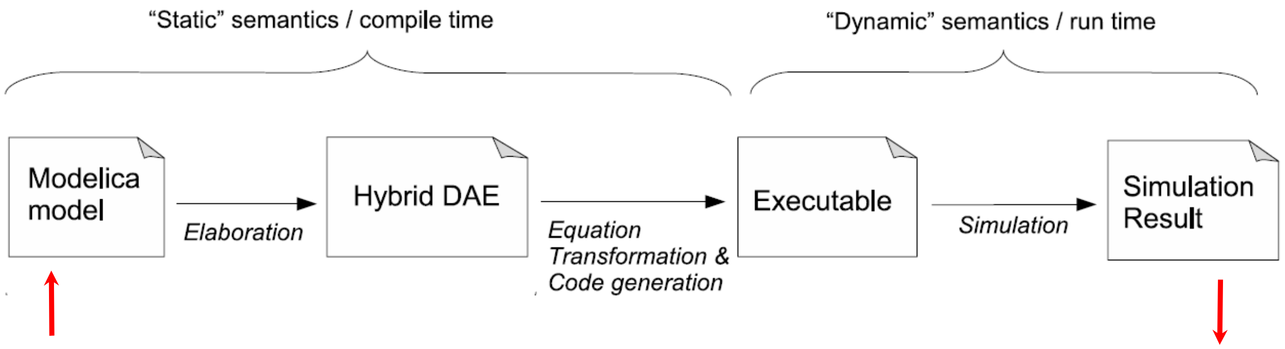
Acausal Modeling

The order of computations is not decided at modeling time

	Acausal	**Causal**
Visual Component Level	 Ramp1 duration={2} Torque1 tau Inertia1 J=10 Spring1 c=5 Inertia2 J=2	 10 Constant2 J1 Divide 1 Integrator 3 omega1 Integrator 1 phi1 5 Gain tau2 tau1 2 Constant 1 Integrator 2 phi2 -5 Gain 1 2 Constant J2 Divide Integrator omega2 Scope tau3

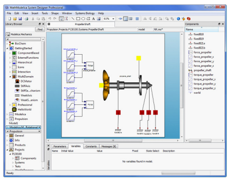
Equation Level	A resistor *equation*:
	`R*i = v;`
		Causal possibilities:
		`i := v/R;`
		`v := R*i;`
		`R := v/i;`

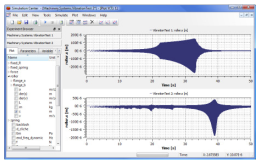
Typical Simulation Process


“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!

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

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

Name of model

Initial condition

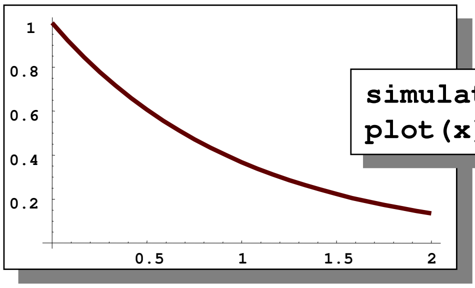
Continuous-time variable

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)
```

Differential Algebraic Equations

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

Informally: ODE + algebraic equations = DAE

General representation of DAEs:

$$\underline{0} = \underline{f}\left(t, \underline{\dot{x}}(t), \underline{x}(t), \underline{y}(t), \underline{u}(t), \underline{p}\right)$$

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

Force it to be the start value

Algebraic variable

Differentiated variable

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

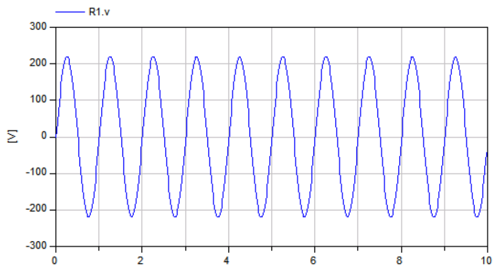
Algebraic equation

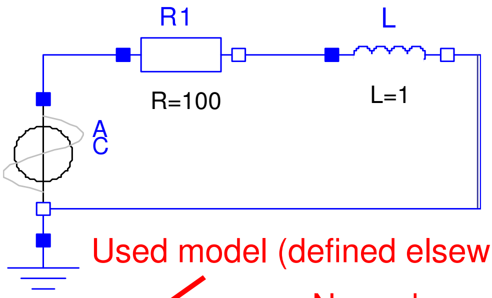
Built-in functions and global time variable

Typically, the compiler transforms the DAE to an ODE before simulation, sometimes using an *index reduction* algorithms.

Textual and Graphical Models

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





Used model (defined elsewhere)

Named component = model instance

Enables later modification of component

Modification of parameter value

Connect equations

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

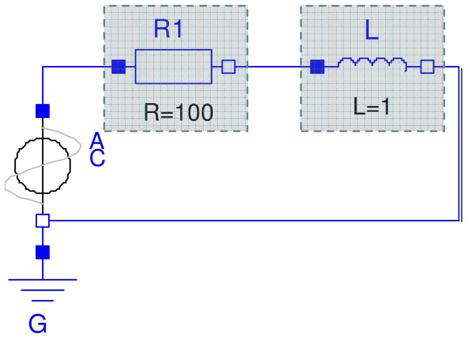
Equations and Inheritance

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

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

Pin p, n and Reals v and i are copied to the subclass

Equations are copied as well.



Inherits equations and components from **TwoPin**

Differential equation

Algebraic equation

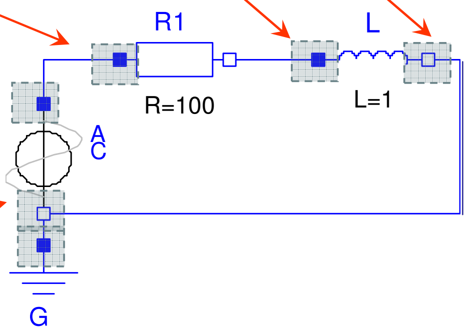
```
model Resistor
   extends TwoPin;
   Real R = 100;
equation
   R*i = v;
end Resistor;
```

```
model Inductor
   extends TwoPin;
   Real L = 1;
equation
   L*der(i) = v;
end Inductor;
```

Connectors (Ports)

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

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



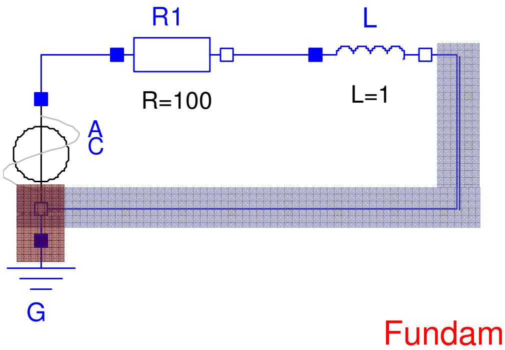
Connectors are instances of a `connector` class.

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

Connections and Flow Variables

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

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



Fundamental concept making acausal modeling work (simplified)

Equations from potential variables:

```
L.n.v = AC.n.v
AC.n.v = G.p.v
```

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

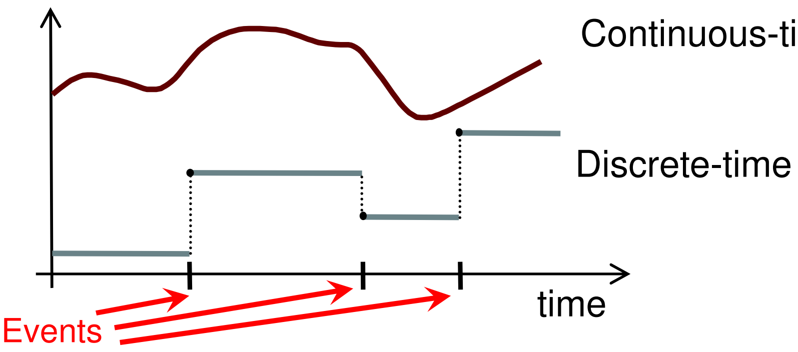
Equation from flow variables:

```
L.n.i + AC.n.i + G.p.i = 0
```

Hybrid Modeling

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

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

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

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`

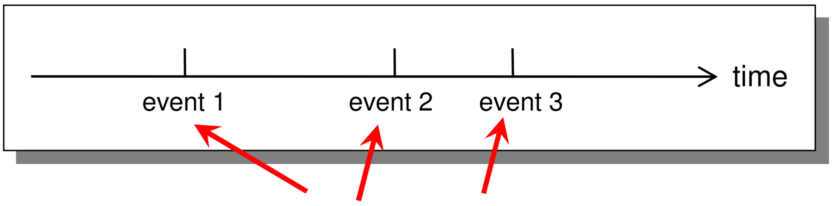
If-expression

Event creation – when

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

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

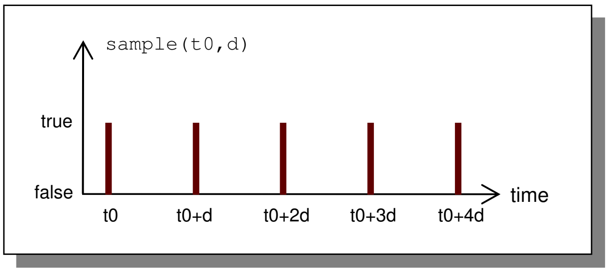
```
when sin(x) > 0.5 then
  ...
end when;
```

Related to a state. Check for zero-crossing

Generating Repeated Events

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

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

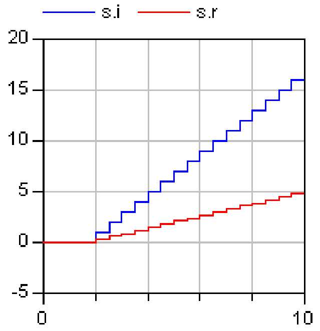


Variables need to be discrete

```
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

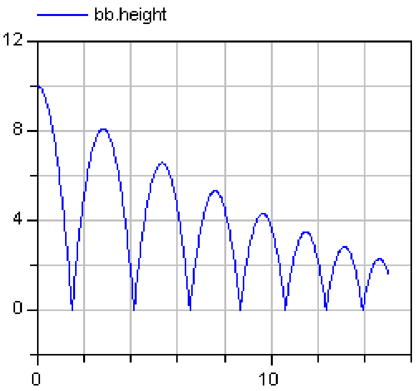
pre(...) takes the previous value before the event.



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

Modelica – large and complex

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

Examples of the features which has not been covered

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

And much more...

MODELICA

Exercise 2.1 Simulate Bouncing Ball

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

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

Exercise 2.2 Bouncing Ball with Stairs

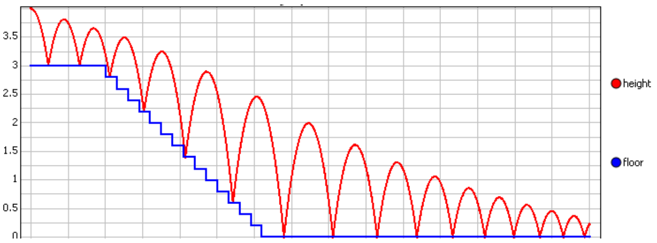
davbr@ida.liu.se
jan.brugard@mathcore.com

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

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

Tips: Plot two variables `x` and `y`:

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



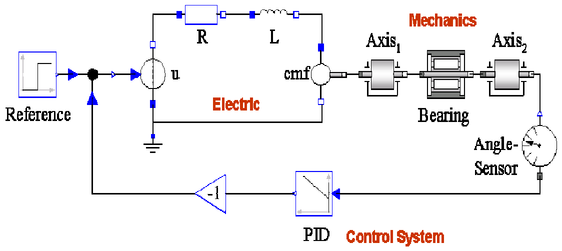
davbr@ida.liu.se
jan.brugard@mathcore.com

Multi-Domain Modeling

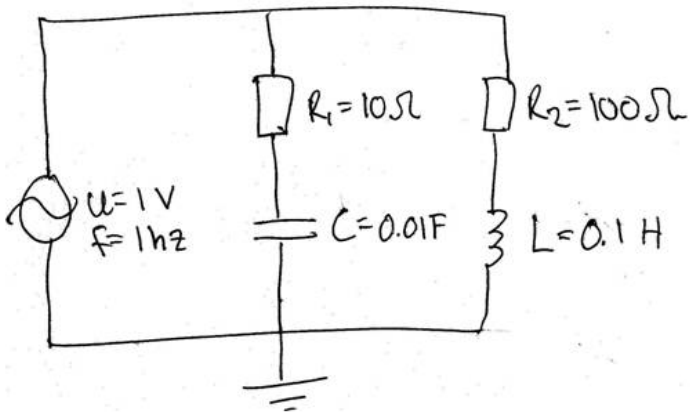
Visual Acausal Component Modeling

Part IV

Graphical modeling and the Modelica standard library



Building an Electric Circuit



$$u(t) = R_1 i_1(t) + \frac{1}{C}\int i_1(t)dt$$

$$u(t) = R_2 i_2(t) + L\frac{di_2(t)}{dt}$$

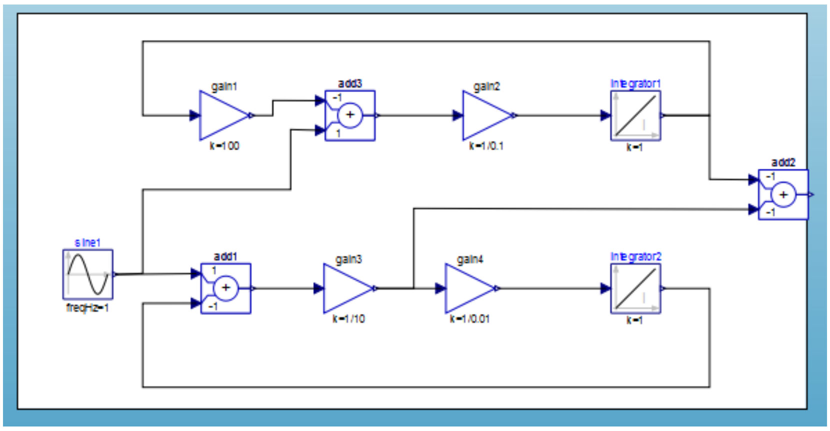
$$i(t) = i_1(t) + i_2(t)$$

$$i_1(t) = \frac{1}{R_1}\left(u(t) - \frac{1}{C}\int i_1(t)dt\right)$$

$$i_2(t) = \frac{1}{R_2}\left(U(s) - L\frac{di_2(t)}{dt}\right)$$

Building an Electric Circuit

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



$$u(t) = R_1 i_1(t) + \frac{1}{C}\int i_1(t)dt$$

$$u(t) = R_2 i_2(t) + L\frac{di_2(t)}{dt}$$

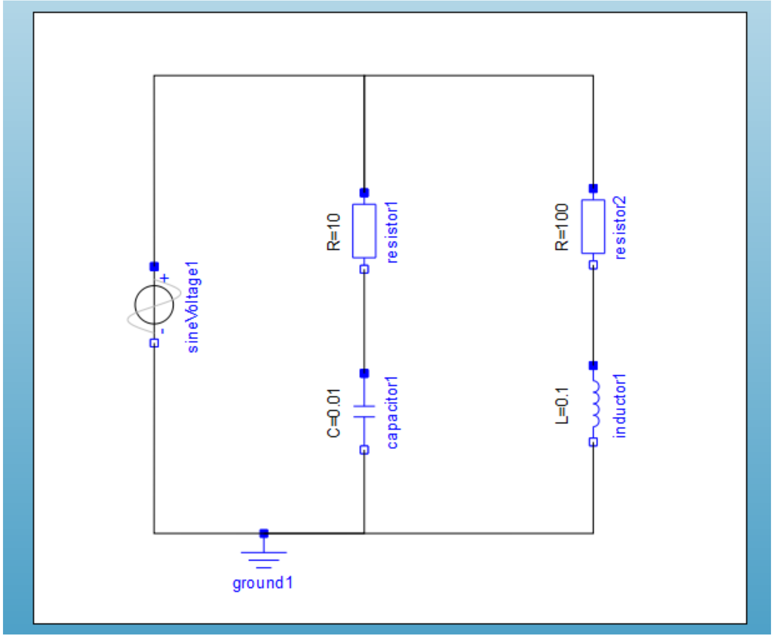
$$i(t) = i_1(t) + i_2(t)$$

$$i_1(t) = \frac{1}{R_1}\left(u(t) - \frac{1}{C}\int i_1(t)dt\right)$$

$$i_2(t) = \frac{1}{R_2}\left(U(s) - L\frac{di_2(t)}{dt}\right)$$

Building an Electric Circuit

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



$$u(t) = R_1 i_1(t) + \frac{1}{C}\int i_1(t)dt$$

$$u(t) = R_2 i_2(t) + L\frac{di_2(t)}{dt}$$

$$i(t) = i_1(t) + i_2(t)$$

$$i_1(t) = \frac{1}{R_1}\left(u(t) - \frac{1}{C}\int i_1(t)dt\right)$$

$$i_2(t) = \frac{1}{R_2}\left(U(s) - L\frac{di_2(t)}{dt}\right)$$

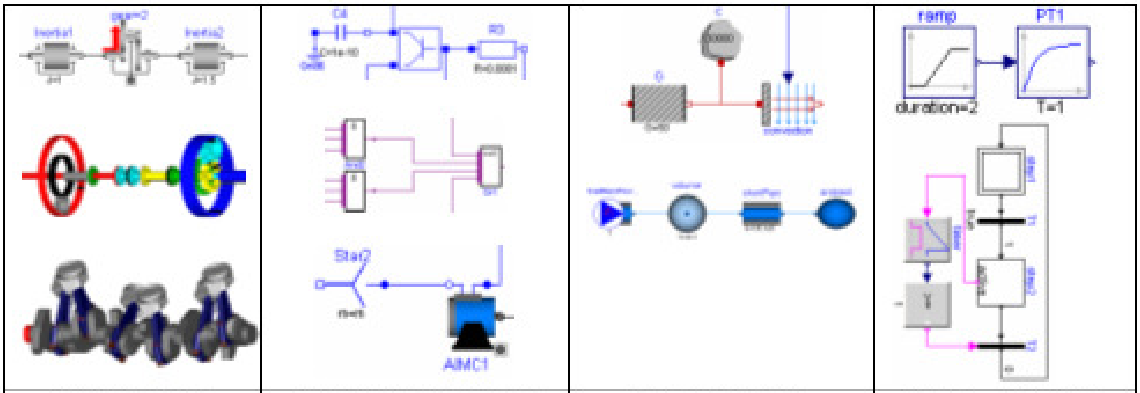
Some Other Domains

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

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	Presure	Mass flow	Air	`PneuLibLight`

Modelica Standard Library

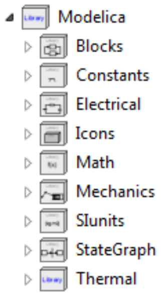
davbr@ida.liu.se
jan.brugard@mathcore.com



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

Modelica Standard Library

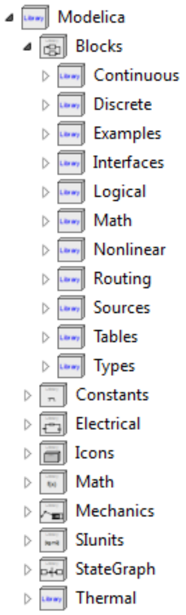
davbr@ida.liu.se
jan.brugard@mathcore.com



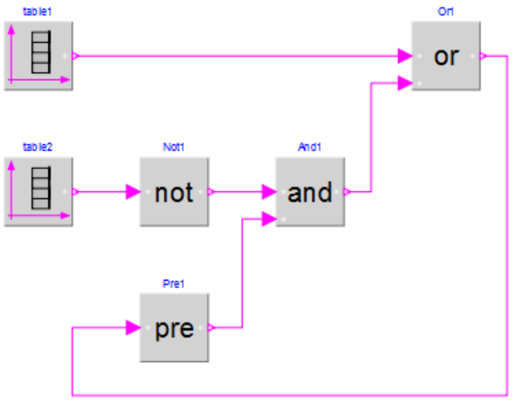
- Modelica
  - Blocks
  - Constants
  - Electrical
  - Icons
  - Math
  - Mechanics
  - SIunits
  - StateGraph
  - Thermal

Blocks

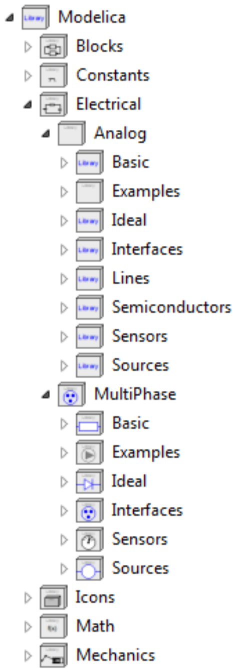
davbr@ida.liu.se
jan.brugard@mathcore.com



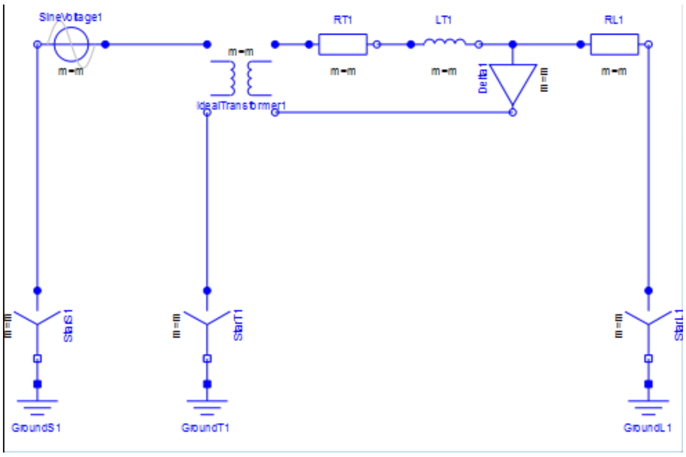
- Modelica
  - Blocks
    - Continuous
    - Discrete
    - Examples
    - Interfaces
    - Logical
    - Math
    - Nonlinear
    - Routing
    - Sources
    - Tables
    - Types
  - Constants
  - Electrical
  - Icons
  - Math
  - Mechanics
  - SIunits
  - StateGraph
  - Thermal



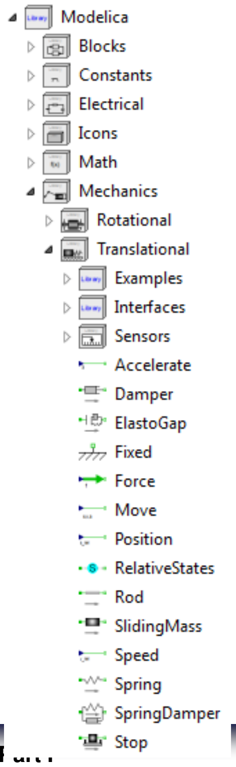
Electrical



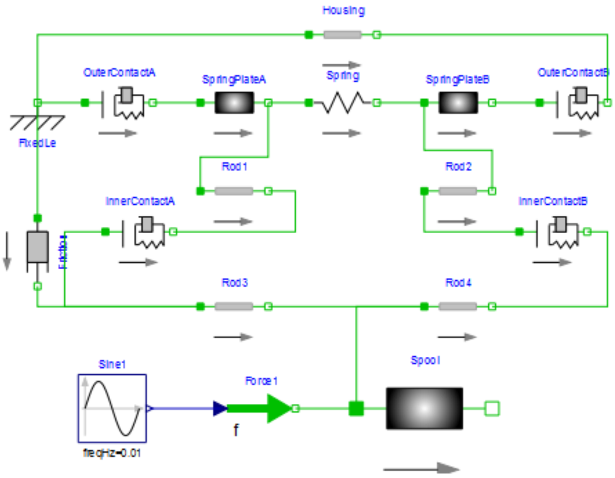
- Modelica
  - Blocks
  - Constants
  - Electrical
    - Analog
      - Basic
      - Examples
      - Ideal
      - Interfaces
      - Lines
      - Semiconductors
      - Sensors
      - Sources
    - MultiPhase
      - Basic
      - Examples
      - Ideal
      - Interfaces
      - Sensors
      - Sources
  - Icons
  - Math
  - Mechanics



Modelica Standard Library

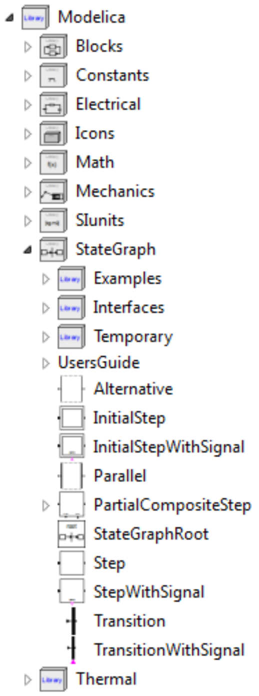


- Modelica
  - Blocks
  - Constants
  - Electrical
  - Icons
  - Math
  - Mechanics
    - Rotational
    - Translational
      - Examples
      - Interfaces
      - Sensors
      - Accelerate
      - Damper
      - ElastoGap
      - Fixed
      - Force
      - Move
      - Position
      - RelativeStates
      - Rod
      - SlidingMass
      - Speed
      - Spring
      - SpringDamper
      - Stop

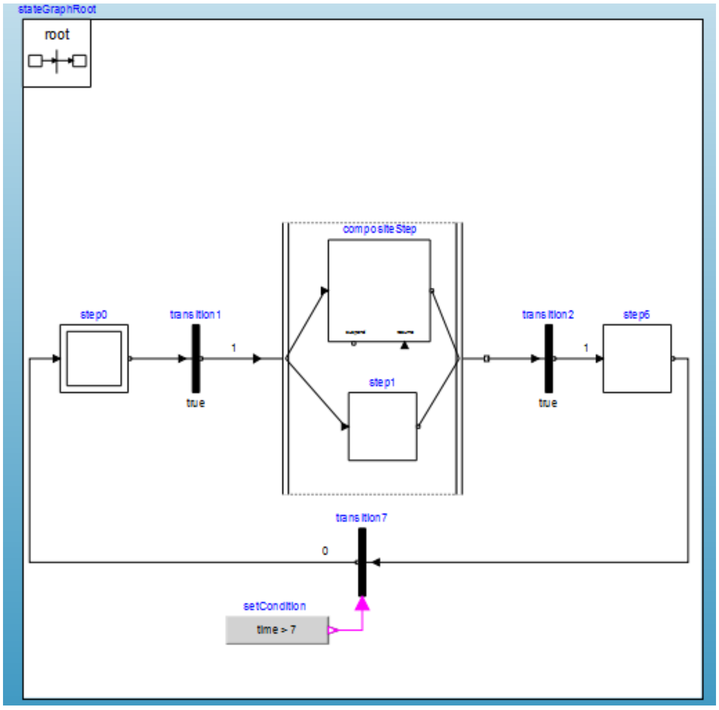


Modelica Standard Library

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

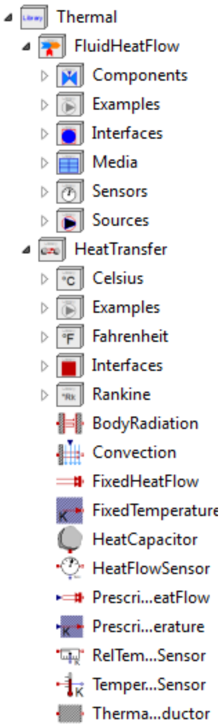


- Modelica
  - Blocks
  - Constants
  - Electrical
  - Icons
  - Math
  - Mechanics
  - SIunits
  - StateGraph
    - Examples
    - Interfaces
    - Temporary
    - UsersGuide
    - Alternative
    - InitialStep
    - InitialStepWithSignal
    - Parallel
    - PartialCompositeStep
    - StateGraphRoot
    - Step
    - StepWithSignal
    - Transition
    - TransitionWithSignal
  - Thermal

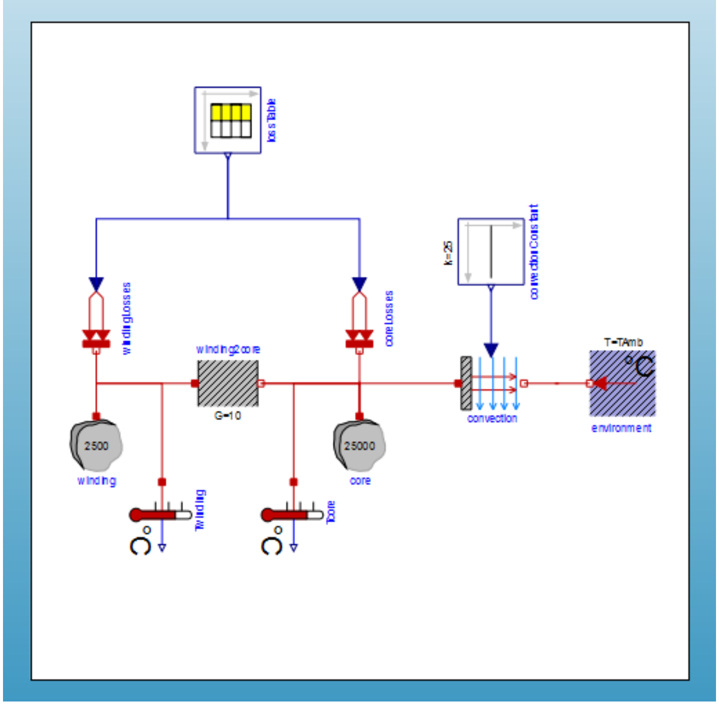


Modelica Standard Library

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



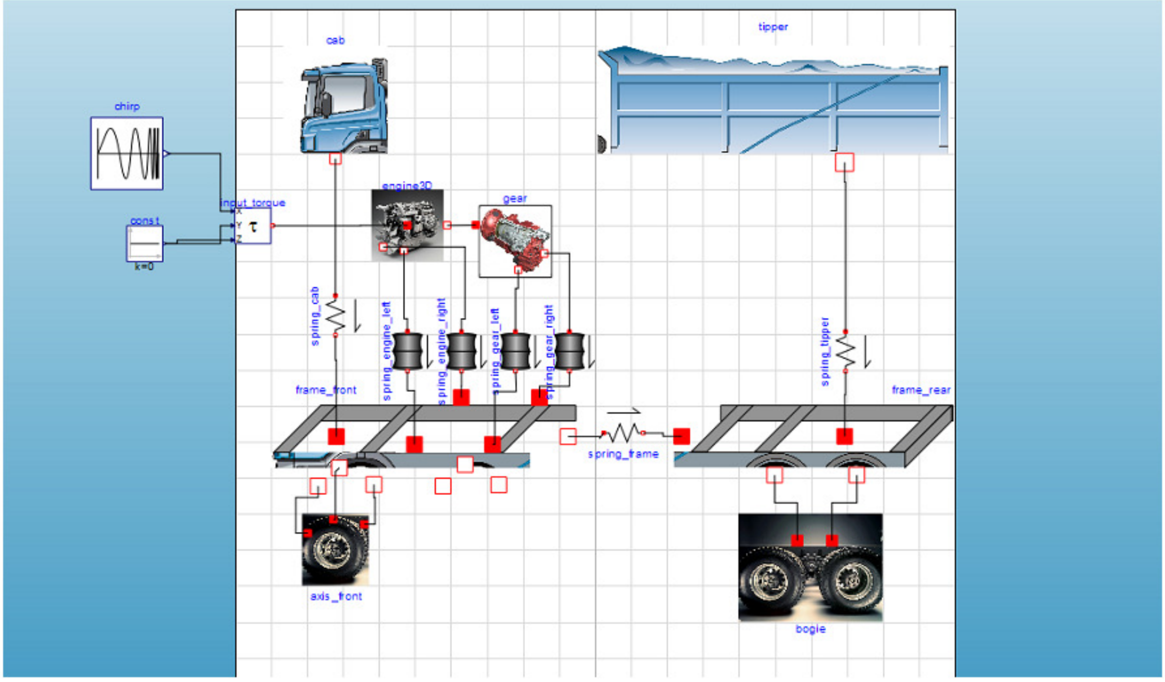
- Thermal
  - FluidHeatFlow
    - Components
    - Examples
    - Interfaces
    - Media
    - Sensors
    - Sources
  - HeatTransfer
    - Celsius
    - Examples
    - Fahrenheit
    - Interfaces
    - Rankine
    - BodyRadiation
    - Convection
    - FixedHeatFlow
    - FixedTemperature
    - HeatCapacitor
    - HeatFlowSensor
    - Prescri...eatFlow
    - Prescri...erature
    - RelTem...Sensor
    - Temper...Sensor
    - Therma...ductor



Other Libraries

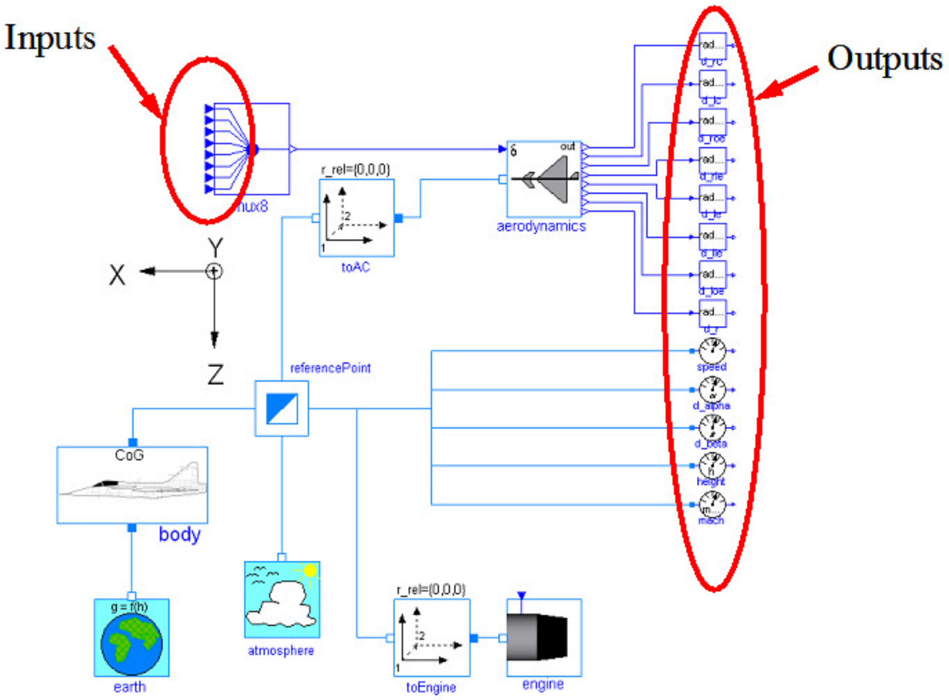
- Bond graphs
- Magnetic
- Systems biology
- Hydraulics
- Pneumatics
- Powertrain
- Petri Nets
- Etc…

Modelica in Autmotive Industry



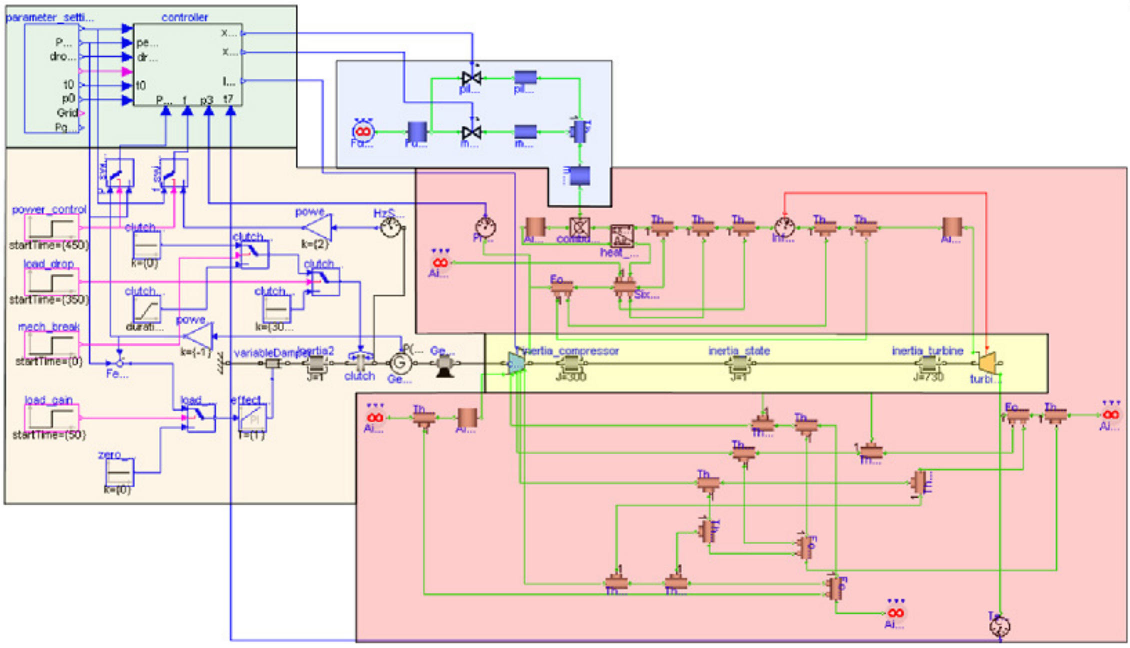
Modelica in Avionics

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

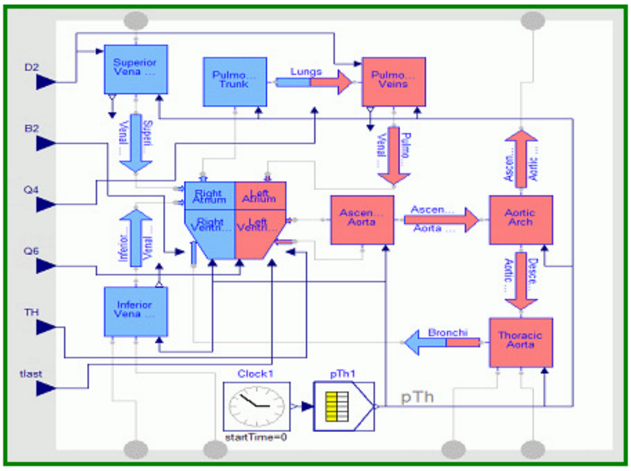


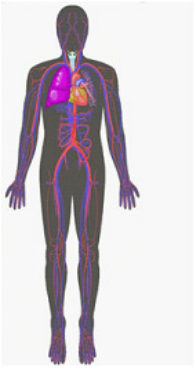
Modelica in Power Generation

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



Modelica in Biomechanics

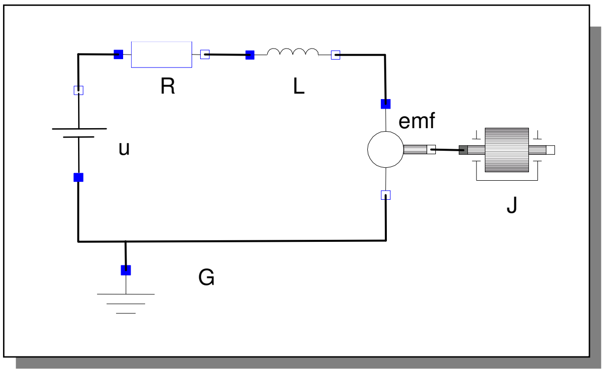




Exercise 3.1

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

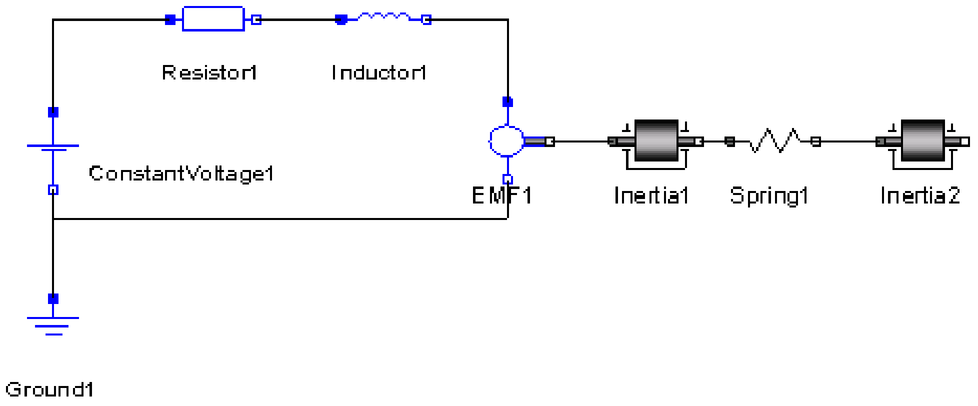
- Simulate it for 15s and plot the variables for the outgoing rotational speed on the inertia axis and the voltage on the voltage source (denoted u in the figure) in the same plot.



Optional Exercise 3.2

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

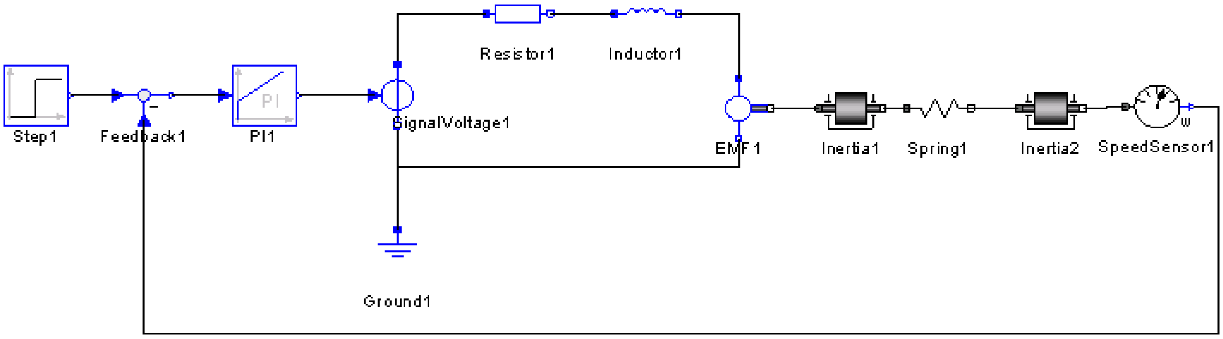
- If there is enough time: Add a torsional spring to the outgoing shaft and another inertia element. Simulate again and see the results. Adjust some parameters to make a rather stiff spring.



Optional Exercise 3.3

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

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

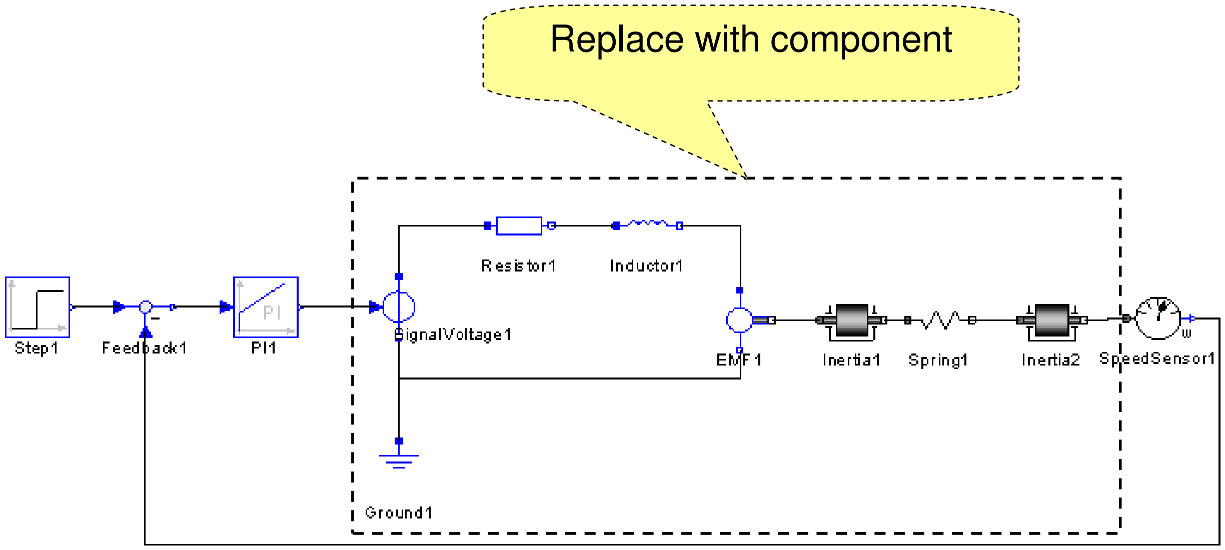


Live example

- Building a component with icon

Optional Exercise 3.4

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



Replace with component

Learn more…

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

PRINCIPLES OF OBJECT-ORIENTED MODELING AND SIMULATION WITH MODELICA 2.1
Introducción al Modelado y Simulación de Sistemas Técnicos y Físicos con Modelica
Peter Fritzson

- Modelica Association
  - www.modelica.org
- OpenModelica
  - www.openmodelica.org
- MathModelica
  - www.mathcore.com,
  - info@mathcore.com
- Books
  - Principles of Object Oriented Modeling and Simulation with Modelica 2.1, Peter Fritzson
  - Introducción al Modelado y Simulación de Sistemas Técnicos y Físicos con Modelica, Peter Fritzson
  - Introduction to Modelica, Michael Tiller

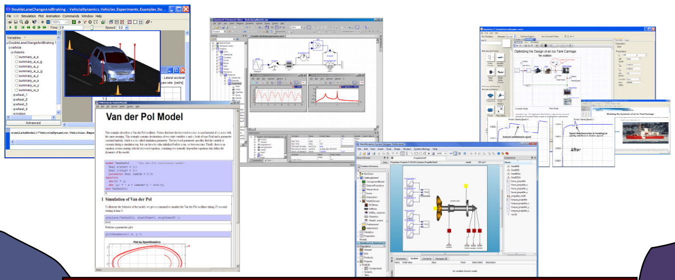
Summary

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

Multi-Domain Modeling

MODELICA

Visual Acausal Component Modeling



Typed Declarative Textual Language

Thanks for listening!

Hybrid Modeling

Acknowledgements and Copyrights

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

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