

Introduction to Object-Oriented Modeling and Simulation with Modelica and OpenModelica

$\tau_2 = \frac{1}{k_2} \tau_1$

$e = \omega_{ref} - \omega_{out}$

$u = K \left(e + \frac{1}{T_I} \int_0^t e \, dt \right)$

$v = u \quad u_R = R i \quad u_{emf} = k_1 \omega_{emf}$

$J_1 \frac{d^2 \theta_1}{dt^2} = \tau_{emf} + \tau_1$

$J_2 \frac{d^2 \theta_2}{dt^2} = \tau_2 + \tau_3$

$J_3 \frac{d^2 \theta_3}{dt^2} = -\tau_4 - \tau_{load}$

$v = u$

$\theta_2 = k_2 \theta_1$

$u_L = L \frac{di}{dt}$

$u = K \left(e + \frac{1}{T_I} \int_0^t e \, dt \right)$

$e = \omega_{ref} - \omega_{out}$

$v - u_R - u_L - u_{emf} = 0$

$u_{emf} = k_1 \omega_{emf} \quad i = \frac{1}{k_1} \tau_{emf} \quad \tau_2 =$

$\frac{J_1 - J_2 k_2^2}{k_2} \frac{d^2 \theta_2}{dt^2} = \tau_{emf} - k_2$

Tutorial, Version Feb 7, 2017

Peter Fritzson

Linköping University, peter.fritzson@liu.se

Director of the Open Source Modelica Consortium

Vice Chairman of Modelica Association

Bernhard Thiele, Ph.D., bernhard.thiele@liu.se

Researcher at PELAB, Linköping University

Slides

Based on book and lecture notes by Peter Fritzson
Contributions 2004-2005 by Emma Larsdotter Nilsson, Peter Bunus

Contributions 2006-2008 by Adrian Pop and Peter Fritzson

Contributions 2009 by David Broman, Peter Fritzson, Jan Brugård, and Mohsen Torabzadeh-Tari

Contributions 2010 by Peter Fritzson

Contributions 2011 by Peter F., Mohsen T., Adeel Asghar,

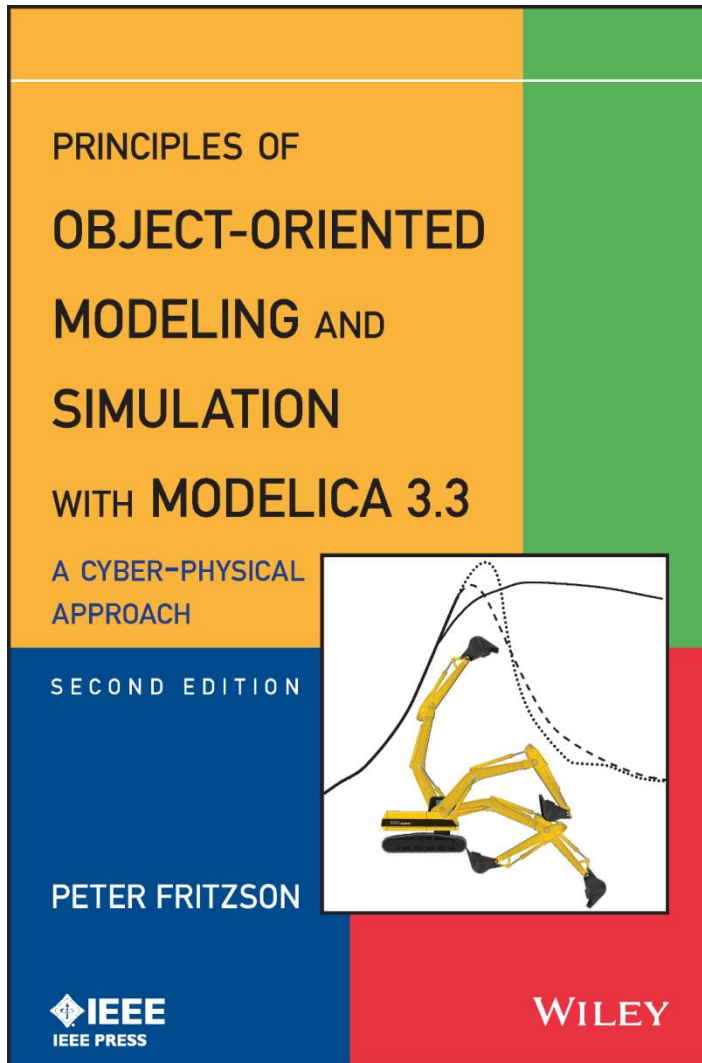
Contributions 2012, 2013, 2014, 2015, 2016 by Peter Fritzson,

Lena Buffoni, Mahder Gebremedhin, Bernhard Thiele



Tutorial Based on Book, December 2014

Download OpenModelica Software



Peter Fritzson

Principles of Object Oriented Modeling and Simulation with Modelica 3.3

A Cyber-Physical Approach

Can be ordered from Wiley or Amazon

Wiley-IEEE Press, 2014, 1250 pages

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

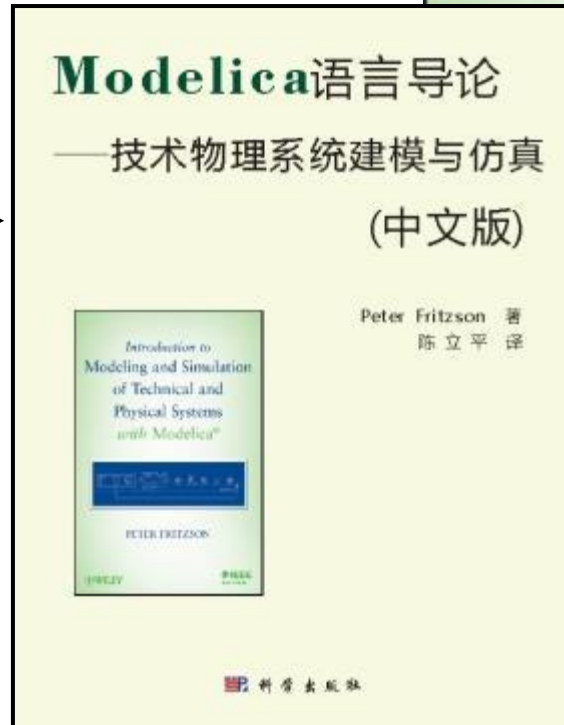
Introductory Modelica Book

September 2011
232 pages

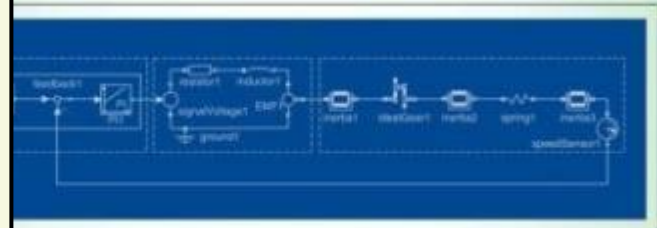
2015 – Translations
available in
Chinese,
Japanese,
Spanish

Wiley
IEEE Press

**For Introductory
Short Courses on
Object Oriented
Mathematical Modeling**



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



PETER FRITZSON

 WILEY

 IEEE
IEEE PRESS

Acknowledgements, Usage, Copyrights

- If you want to use the Powerpoint version of these slides in your own course, send an email to: peter.fritzson@ida.liu.se
- Thanks to Emma Larsdotter Nilsson, Peter Bunus, David Broman, Jan Brugård, Mohsen-Torabzadeh-Tari, Adeel Asghar, Lena Buffoni, for contributions to these slides.
- Most examples and figures in this tutorial are adapted with permission from Peter Fritzson's book "Principles of Object Oriented Modeling and Simulation with Modelica 2.1", copyright Wiley-IEEE Press
- Some examples and figures reproduced with permission from Modelica Association, Martin Otter, Hilding Elmqvist, Wolfram MathCore, Siemens
- Modelica Association: www.modelica.org
- OpenModelica: www.openmodelica.org

Outline

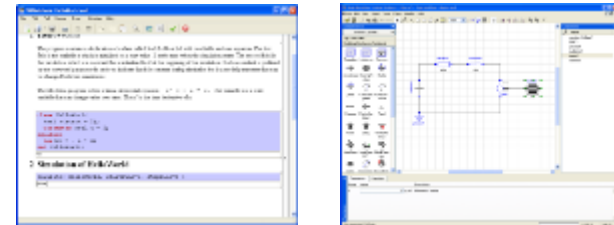
Part I

Introduction to Modelica and a demo example



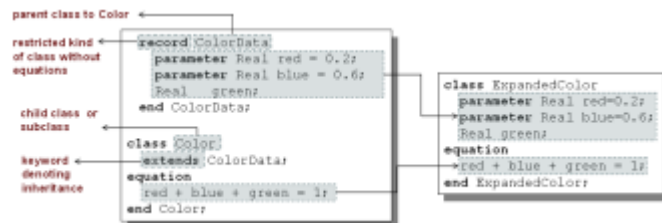
Part II

Modelica environments



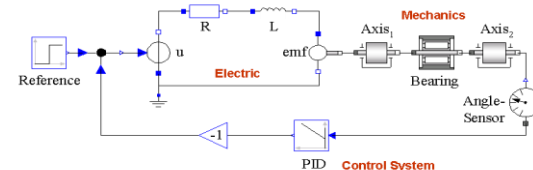
Part III

Modelica language concepts and textual modeling



Part IV and Part V

Graphical modeling and the Modelica standard library
Dynamic Optimization



Detailed Schedule (morning version) 09.00-13.00

09:00 - Introduction to Modeling and Simulation

- Start installation of **OpenModelica** including **OMEdit** graphic editor

09:10 - Modelica – The Next Generation Modeling Language

09:25 - *Exercises Part I (15 minutes)*

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

09:50 – Part II: Modelica Environments and the OpenModelica Environment

10:10 – Part III: Modelica Textual Modeling

10:15 - *Exercises Part IIIa (10 minutes)*

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

10:25 – Coffee Break

10:40 - Modelica Discrete Events, Hybrid, Clocked Properties (Bernhard Thiele)

11:00- *Exercises Part IIIb (15 minutes)*

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

11:20– Part IV: Components, Connectors and Connections
- Modelica Libraries

11:30 – *Part V* Dynamic Optimization (Bernhard Thiele)

- Hands-on exercise on dynamic optimization using **OpenModelica**

12:00 – Exercise Graphical Modeling DCMotor using OpenModelica

Software Installation - Windows

- Start the software installation
- Install OpenModelica-1.11.0.exe Download or from the USB Stick

Software Installation – Linux (requires internet connection)

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

Software Installation – MAC (requires internet connection)

- Go to <https://openmodelica.org/index.php/download/download-mac> and follow the instructions or follow the instructions written below.
- The installation uses MacPorts. After setting up a MacPorts installation, run the following commands on the terminal (as root):
 - *echo rsync://build.openmodelica.org/macports/ >> /opt/local/etc/macports/sources.conf # assuming you installed into /opt/local*
 - *port selfupdate*
 - *port install openmodelica-devel*

Part I

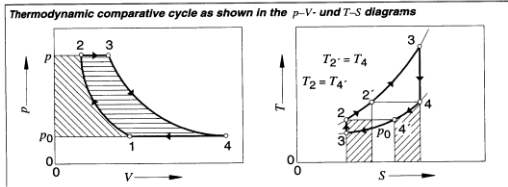
Introduction to Modelica and a demo example



Modelica Background: Stored Knowledge

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

Internal-combustion engines 417



from T_2 to T_2' , supplied by the heat exchanger is coupled with a thermal discharge ($4 \rightarrow 4'$). If heat is completely exchanged, the quantity of heat to be added per unit of gas is reduced to

$$q_{in} = c_p \cdot (T_3 - T_2) = c_p \cdot (T_3 - T_4)$$

and the quantity of heat to be removed is

$$q_{out} = c_p \cdot (T_4 - T_1) = c_p \cdot (T_2 - T_1).$$

The maximum thermal efficiency for the gas turbine with heat exchanger is:

$$\eta_{th} = 1 - Q_{out}/Q_{in} = 1 - (T_2 - T_1)/(T_3 - T_4)$$

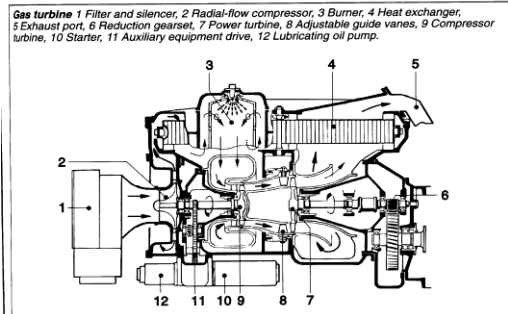
Where $p_2/p_1 = (T_2/T_1)^{\frac{\gamma-1}{\gamma}} = (T_3/T_4)^{\frac{\gamma-1}{\gamma}}$ and $T_4 = T_3 \cdot (T_1/T_2)$ thus

$$\eta_{th} = 1 - (T_2/T_3)$$

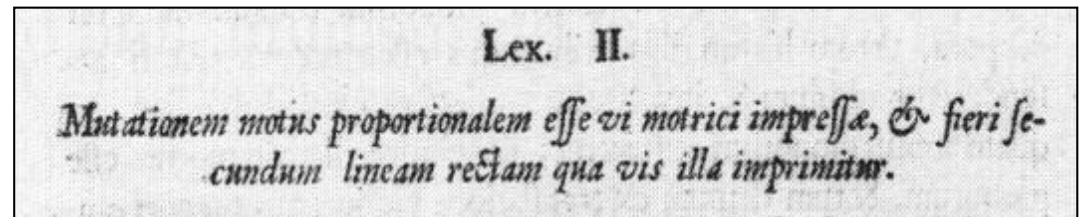
Current gas-turbine powerplants achieve thermal efficiencies of up to 35%.

Advantages of the gas turbines: clean exhaust without supplementary emissions-control devices; extremely smooth running; multifuel capability; good static torque curve; extended maintenance intervals.

Disadvantages: manufacturing costs still high; poor transitional response; higher fuel consumption; less suitable for low-power applications.

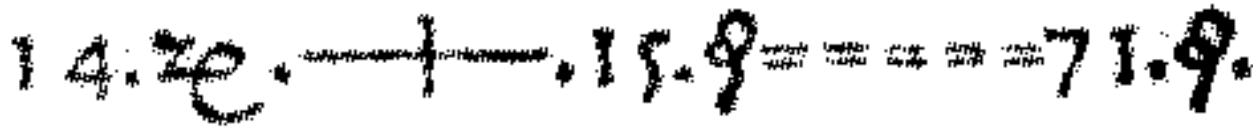


“The change of motion is proportional to the motive force impressed”
– Newton



Modelica Background: The Form – Equations

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



Newton still wrote text (Principia, vol. 1, 1686)

“The change of motion is proportional to the motive force impressed”

CSSL (1967) introduced a special form of “equation”:

variable = expression

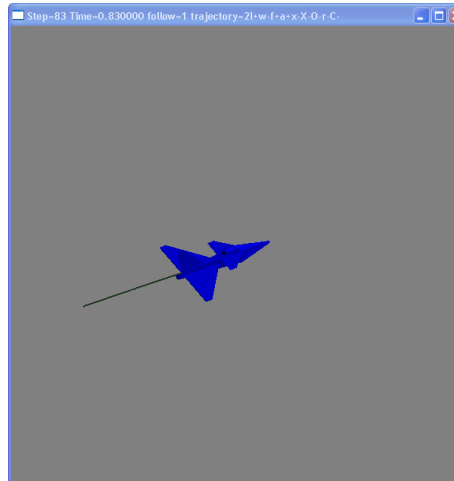
$v = \text{INTEG}(F) / m$

Programming languages usually do not allow equations!

What is Modelica?

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

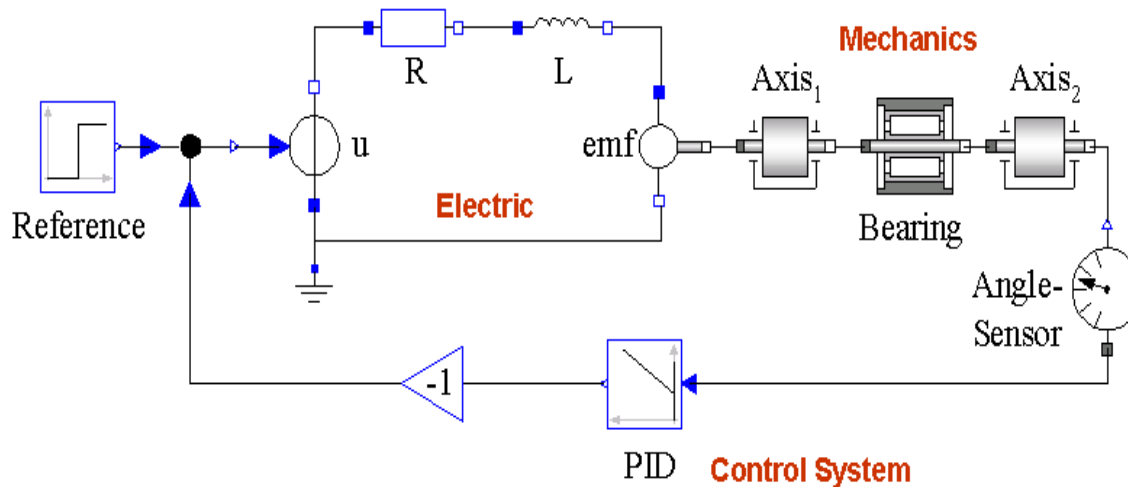
- Robotics
- Automotive
- Aircrafts
- Satellites
- Power plants
- Systems biology



 MODELICA

What is Modelica?

A language for modeling of complex cyber-physical systems



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

What is Modelica?

A language for modeling of complex cyber-physical systems

i.e., Modelica is not a tool

Free, open language
specification:



There exist several free and commercial tools, for example:

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

Available at: www.modelica.org

*Developed and standardized
by Modelica Association*

Modelica – The Next Generation Modeling Language

Declarative language

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

Multi-domain modeling

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

Everything is a class

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

Visual component programming

Hierarchical system architecture capabilities

Efficient, non-proprietary

Efficiency comparable to C; advanced equation compilation, e.g. 300 000 equations, ~150 000 lines on standard PC

Modelica Acausal Modeling

What is *acausal* modeling/design?

Why does it increase *reuse*?

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

Example: a resistor *equation*:

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

can be used in three ways:

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

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

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

What is Special about Modelica?

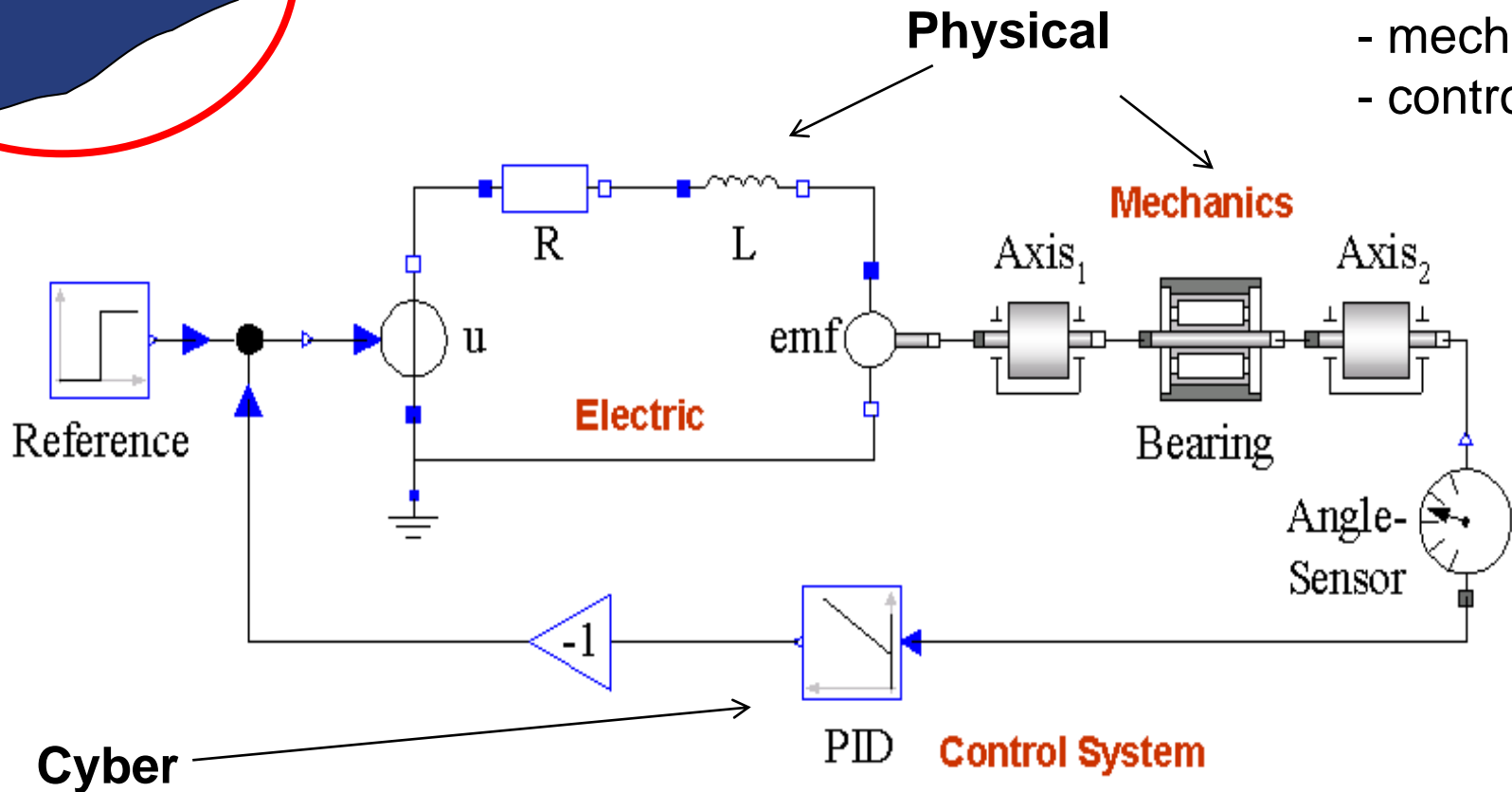
- Multi-Domain Modeling
- Visual acausal hierarchical component modeling
- Typed declarative equation-based textual language
- Hybrid modeling and simulation

What is Special about Modelica?

Multi-Domain
Modeling

Cyber-Physical Modeling

- 3 domains
- electric
 - mechanics
 - control



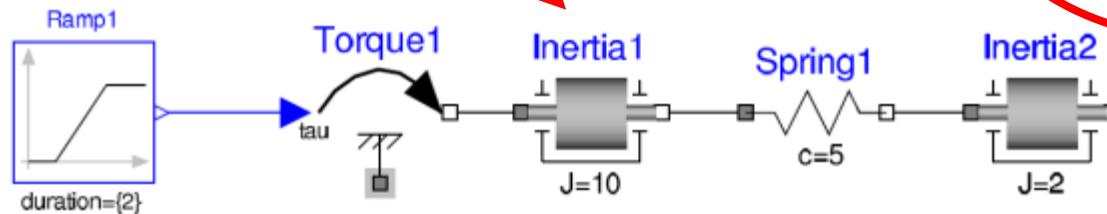
What is Special about Modelica?

Multi-Domain
Modeling

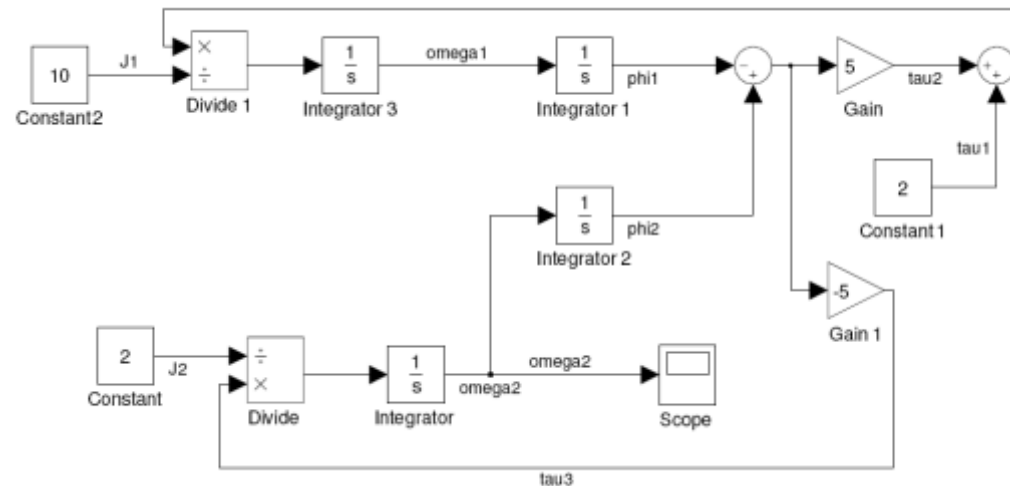
Visual Acausal
Hierarchical
Component
Modeling

Keeps the physical
structure

Acausal model
(Modelica)



Causal
block-based
model
(Simulink)

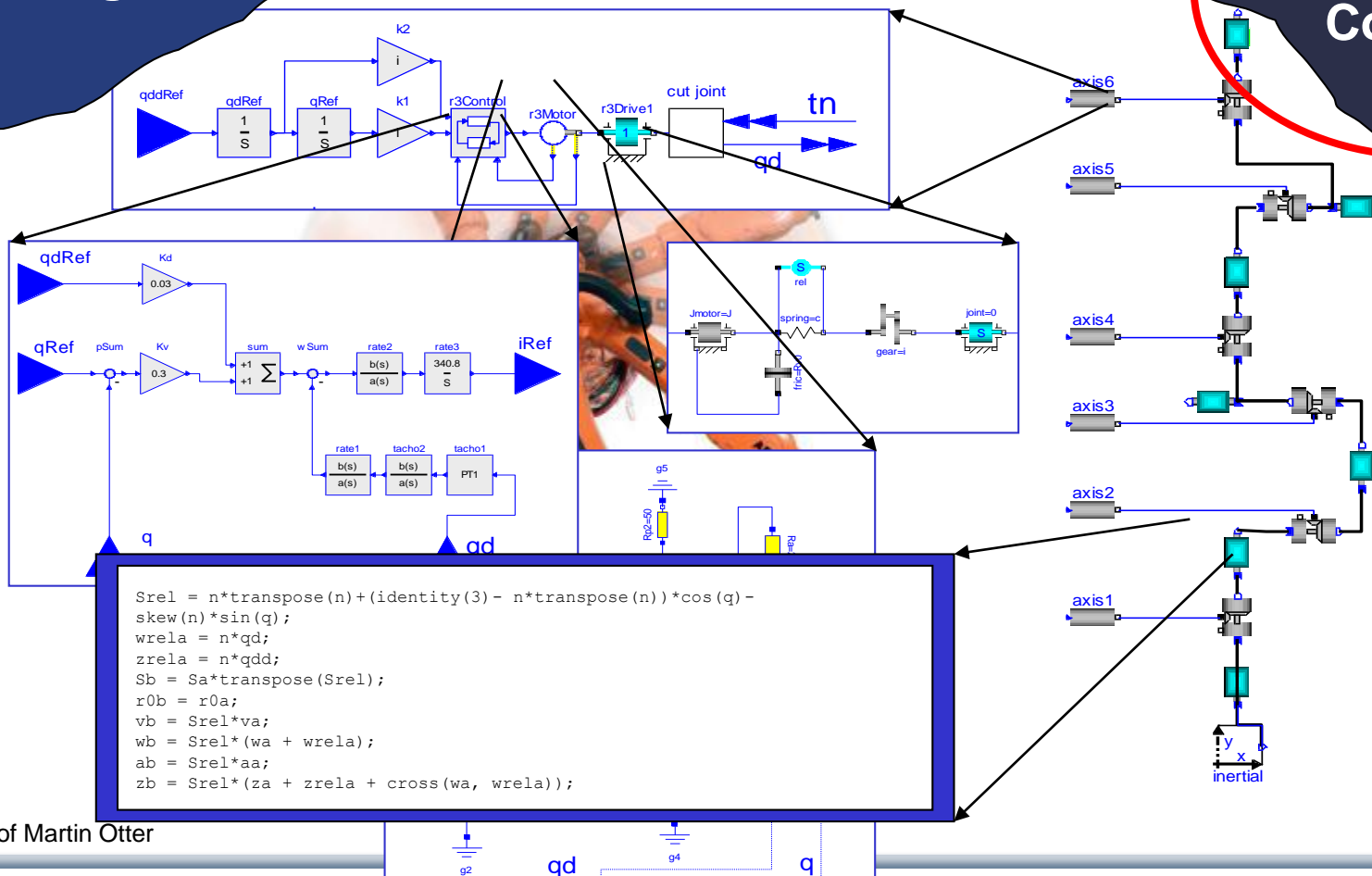


What is Special about Modelica?

Multi-Domain
Modeling

Hierarchical system
modeling

Visual Acausal
Hierarchical
Component
Modeling



Courtesy of Martin Otter

What is Special about Modelica?

Multi-Domain
Modeling

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

Visual Acausal
Hierarchical
Component
Modeling

Behaviour described declaratively using

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

Variable
declarations

```
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;
```

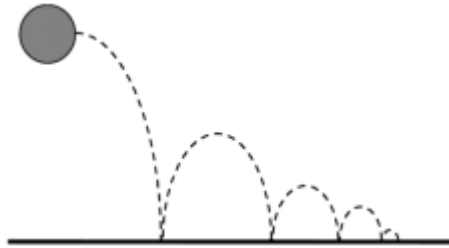
Differential equations

Typed
Declarative
Equation-based
Textual Language

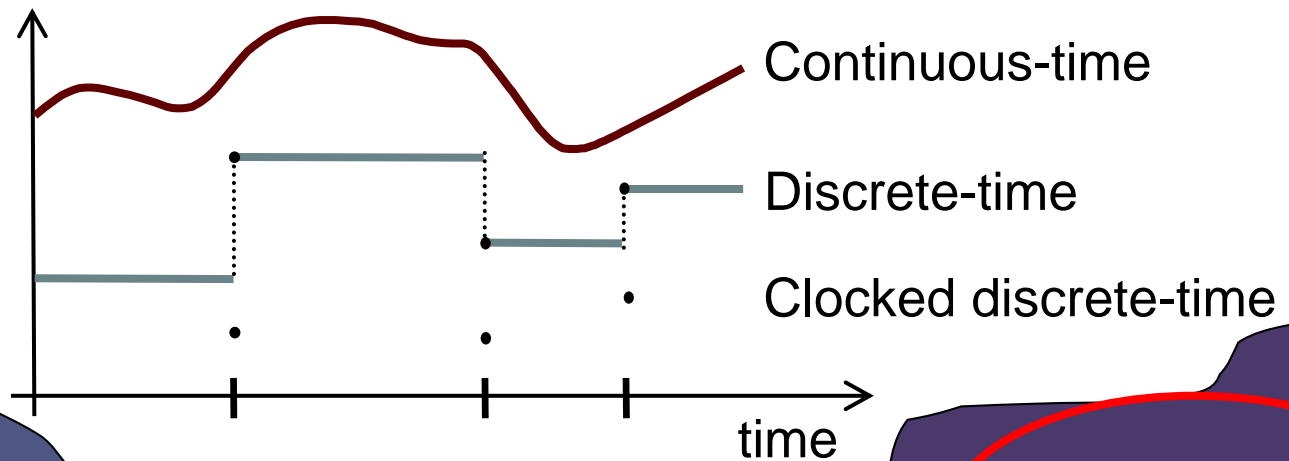
What is Special about Modelica?

Multi-Domain
Modeling

Visual Acausal
Component
Modeling



Hybrid modeling =
continuous-time + discrete-time modeling

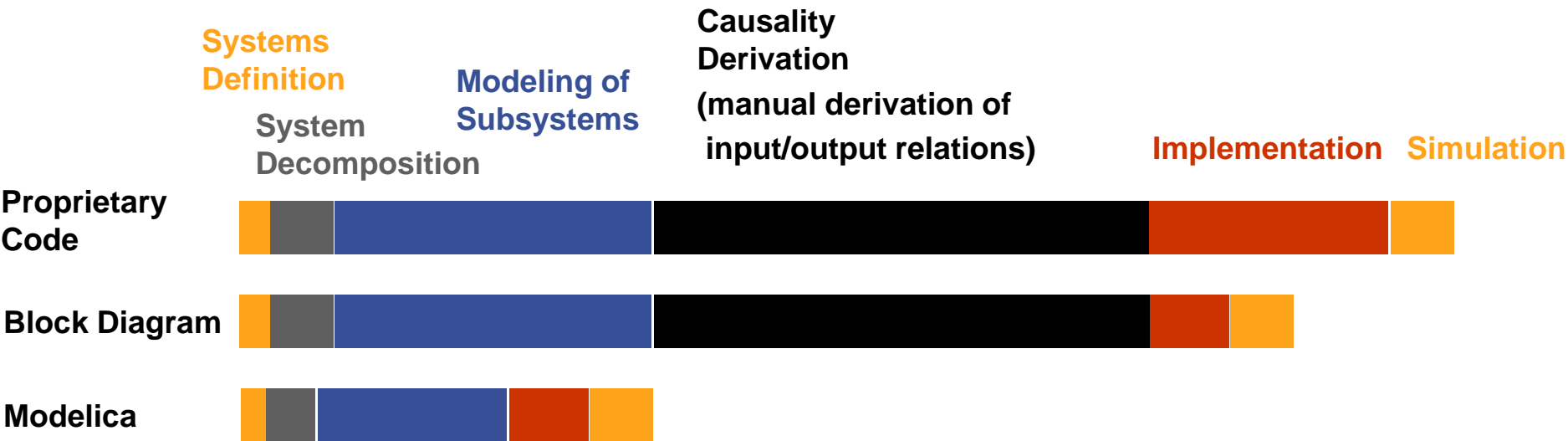


Typed
Declarative
Equation-based
Textual Language

Hybrid
Modeling

Modelica – Faster Development, Lower Maintenance than with Traditional Tools

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

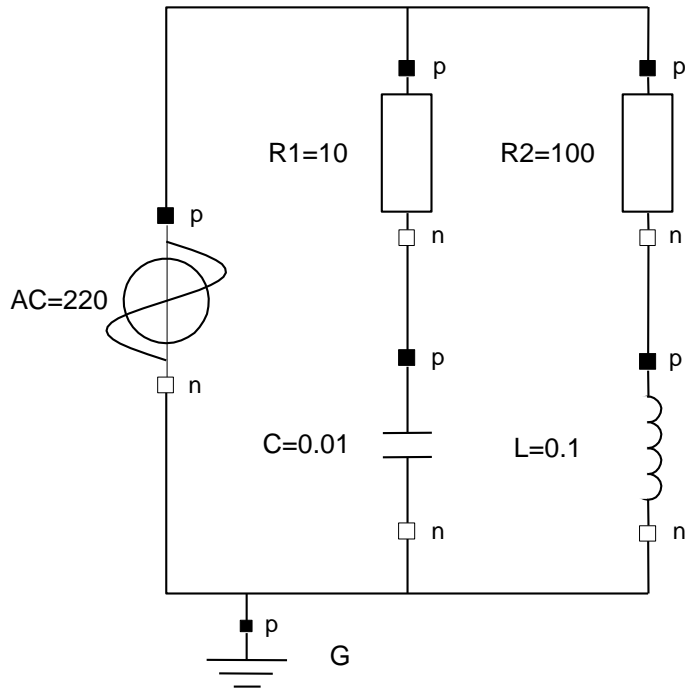


Modelica vs Simulink Block Oriented Modeling

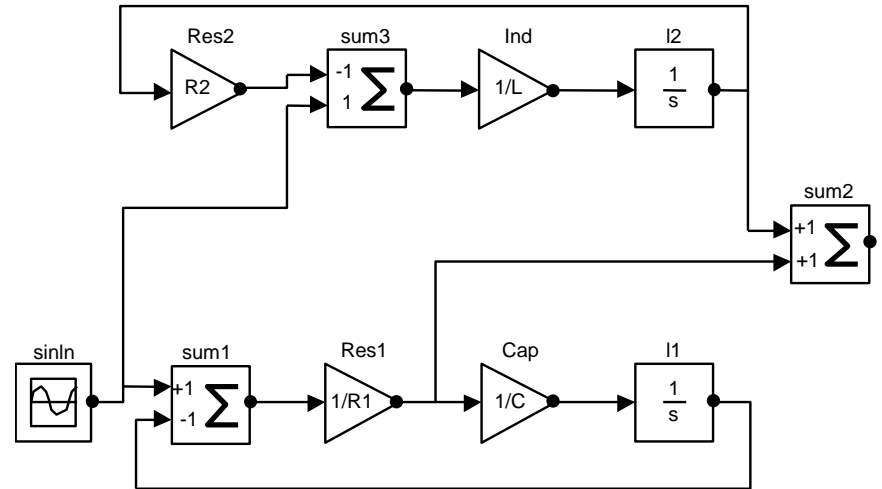
Simple Electrical Model

Modelica:
Physical model –
easy to understand

Keeps the
physical
structure



Simulink:
Signal-flow model – hard to
understand



Graphical Modeling - Using Drag and Drop Composition

The screenshot displays the OpenModelica Connection Editor (OMEdit) interface. The main window shows a circuit diagram titled "RL_Circuit*" in "Diagram View". The circuit consists of a sine voltage source labeled "sineVoltage1" connected in series with a resistor labeled "resistor1" (with the equation $R = \%R$) and an inductor labeled "inductor1" (with the equation $L = \%L$). The circuit is connected to a ground symbol labeled "ground1".

The left sidebar shows the "Modelica Standard Library" with the following structure:

- Modelica
 - Blocks
 - Constants
 - Electrical
 - Analog
 - Basic
 - Examples
 - Ideal
 - Interfaces
 - Lines
 - Semiconductors
 - Sensors
 - Sources
 - ConstantCurrent
 - ConstantVoltage
 - ExponentialsCurrent
 - ExponentialsVoltage
 - ExpSineCurrent
 - ExpSineVoltage
 - PulseCurrent
 - PulseVoltage
 - RampCurrent
 - RampVoltage
 - SawToothCurrent
 - SawToothVoltage

The bottom status bar shows the following messages:

```
--- Info 3 : 10:39:17 ---  
Connected: (resistor1.n, inductor1.p)  
--- Info 4 : 10:39:39 ---  
Connected: (inductor1.n, sineVoltage1.n)
```

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);
```

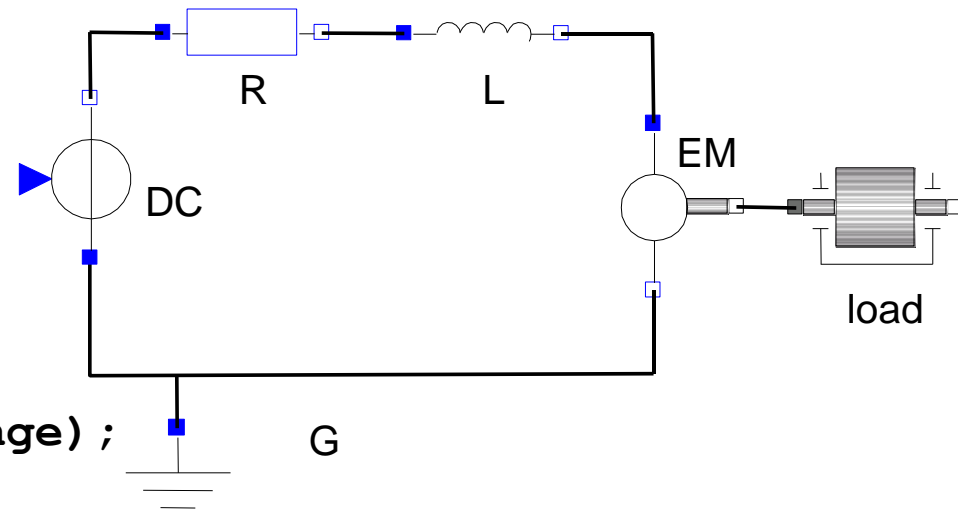
```
connect (L.p, EM.n);
```

```
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:

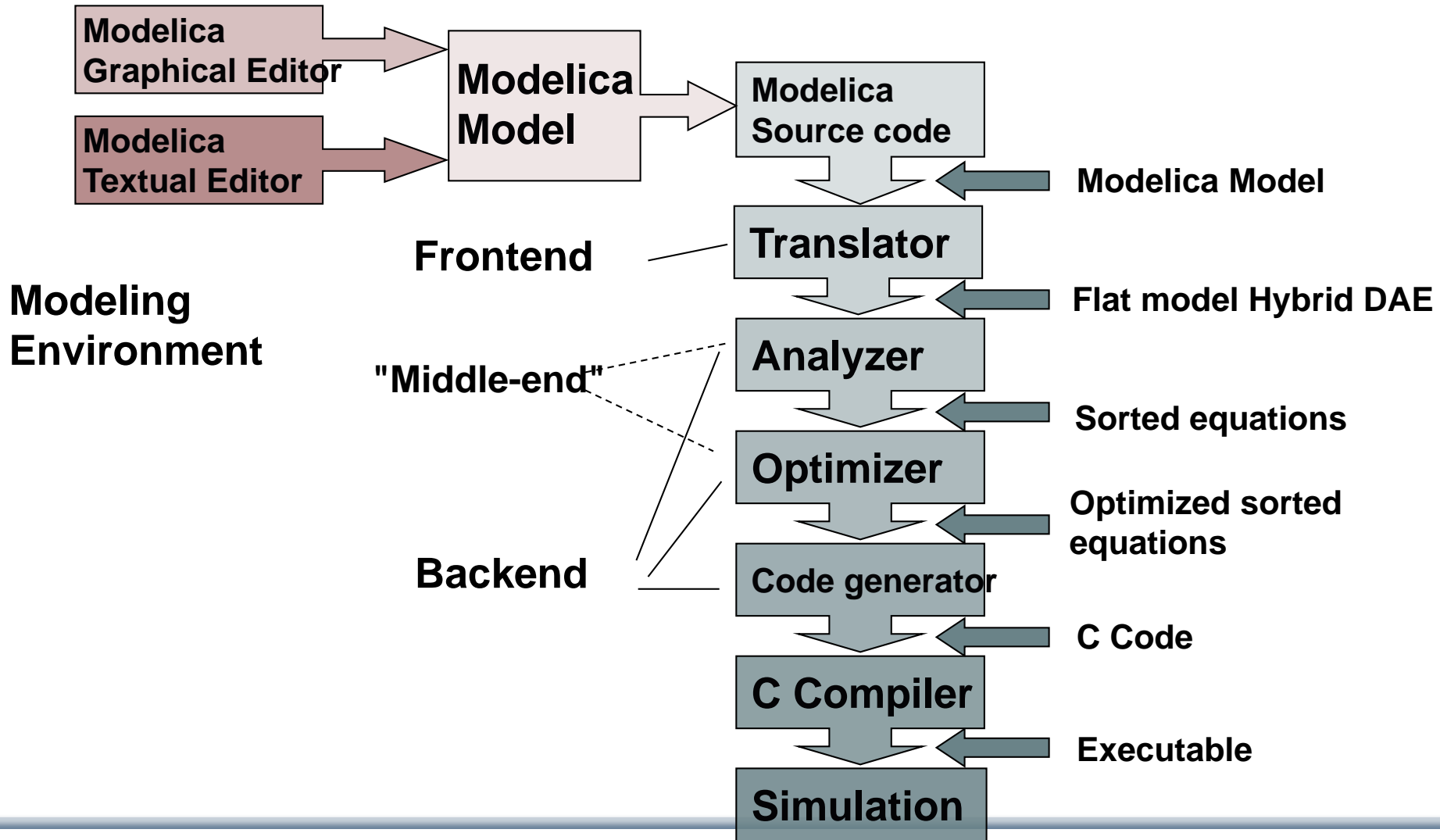
$0 == DC.p.i + R.n.i$	$EM.u == EM.p.v - EM.n.v$	$R.u == R.p.v - R.n.v$
$DC.p.v == R.n.v$	$0 == EM.p.i + EM.n.i$	$0 == R.p.i + R.n.i$
	$EM.i == EM.p.i$	$R.i == R.p.i$
$0 == R.p.i + L.n.i$	$EM.u == EM.k * EM.\omega$	$R.u == R.R * R.i$
$R.p.v == L.n.v$	$EM.i == EM.M / EM.k$	
	$EM.J * EM.\omega == EM.M - EM.b * EM.\omega$	$L.u == L.p.v - L.n.v$
$0 == L.p.i + EM.n.i$		$0 == L.p.i + L.n.i$
$L.p.v == EM.n.v$	$DC.u == DC.p.v - DC.n.v$	$L.i == L.p.i$
	$0 == DC.p.i + DC.n.i$	$L.u == L.L * L.i'$
$0 == EM.p.i + DC.n.i$	$DC.i == DC.p.i$	
$EM.p.v == DC.n.v$	$DC.u == DC.Amp * Sin[2 \pi DC.f * t]$	
$0 == DC.n.i + G.p.i$		
$DC.n.v == G.p.v$		

(load component not included)

Automatic transformation to ODE or DAE for simulation:

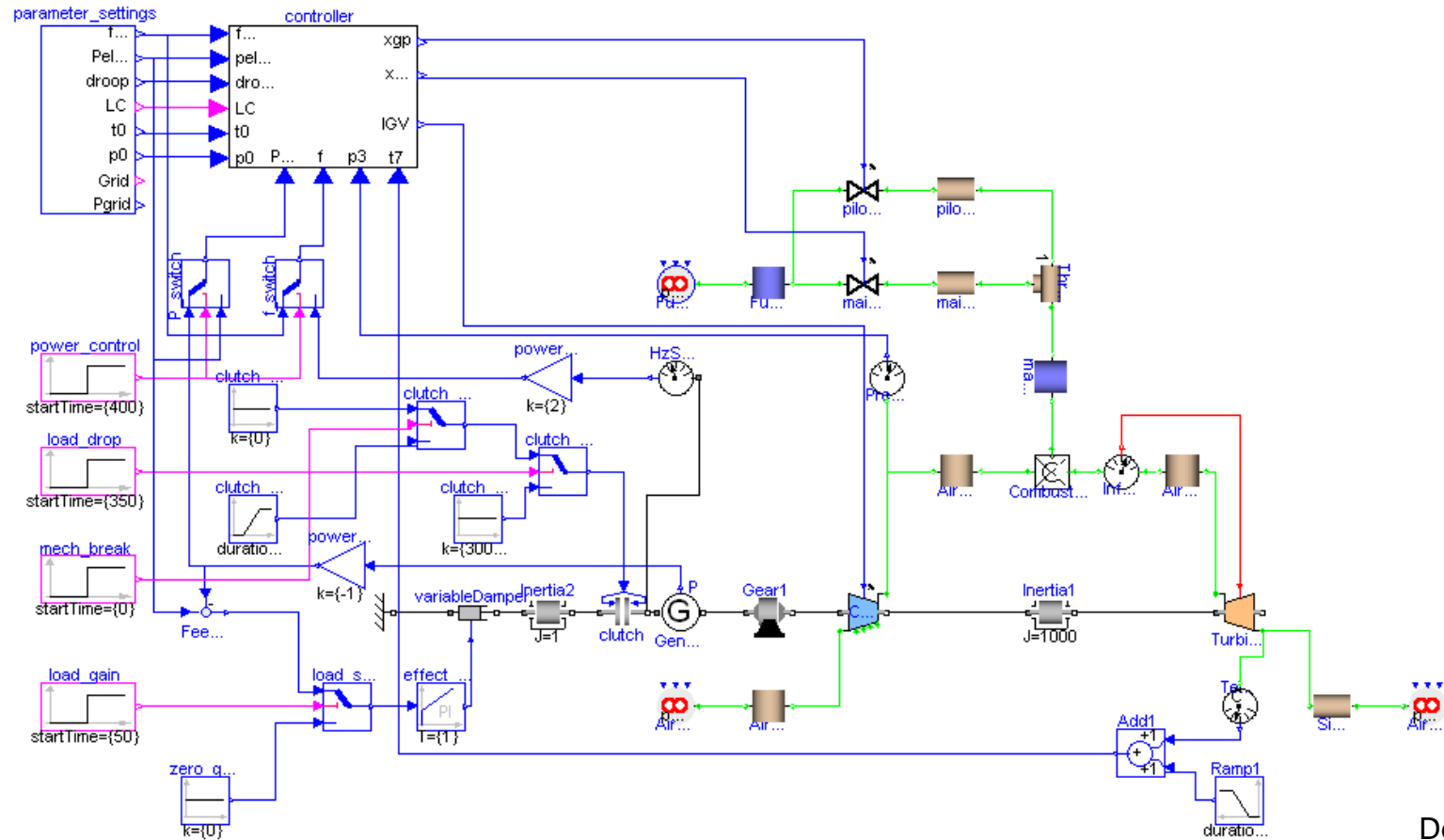
$$\frac{dx}{dt} == f[x, u, t] \quad g\left[\frac{dx}{dt}, x, u, t\right] == 0$$

Model Translation Process to Hybrid DAE to Code



Modelica in Power Generation

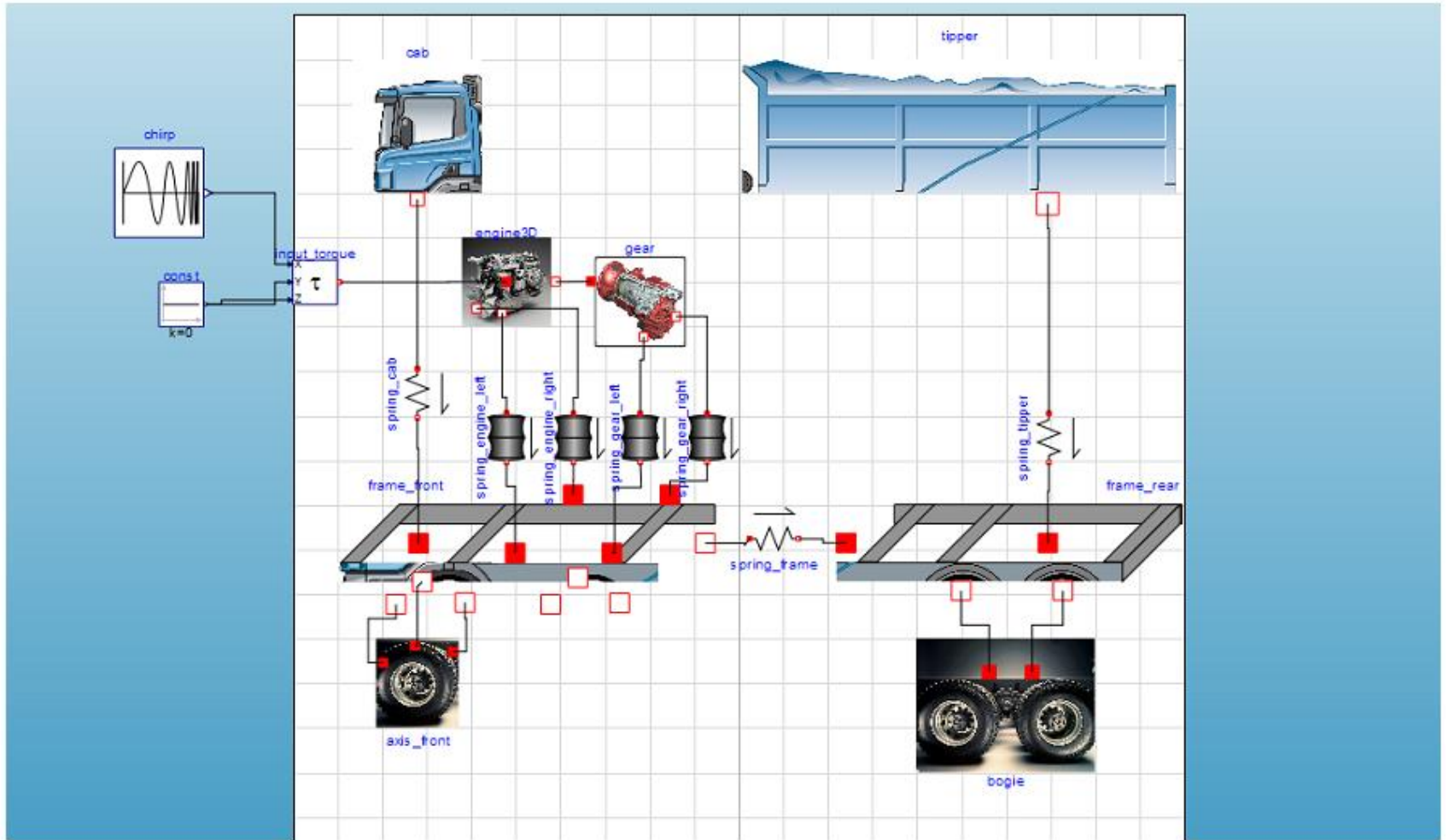
GTX Gas Turbine Power Cutoff Mechanism



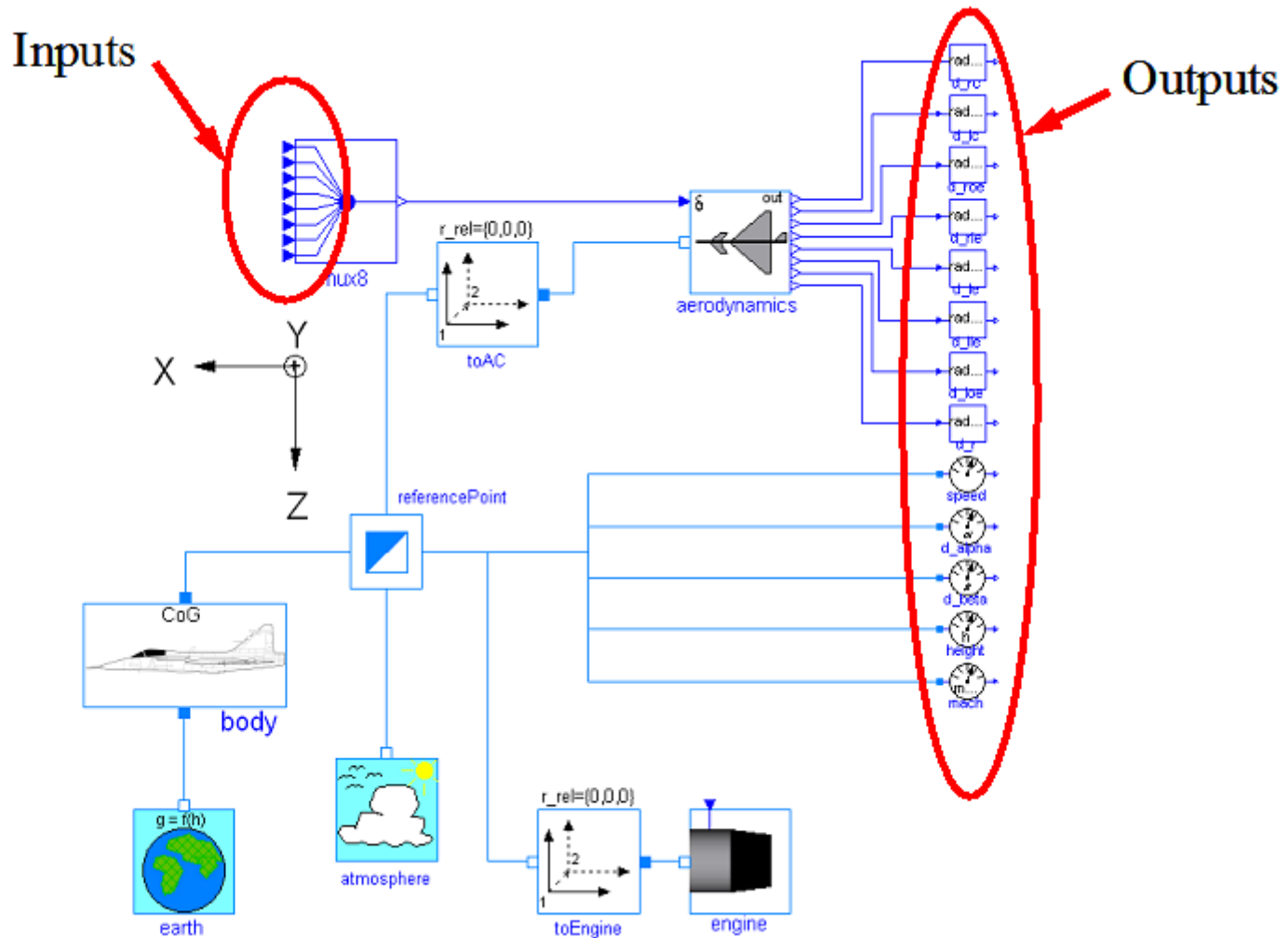
Developed
by MathCore
for Siemens

Courtesy of Siemens Industrial Turbomachinery AB

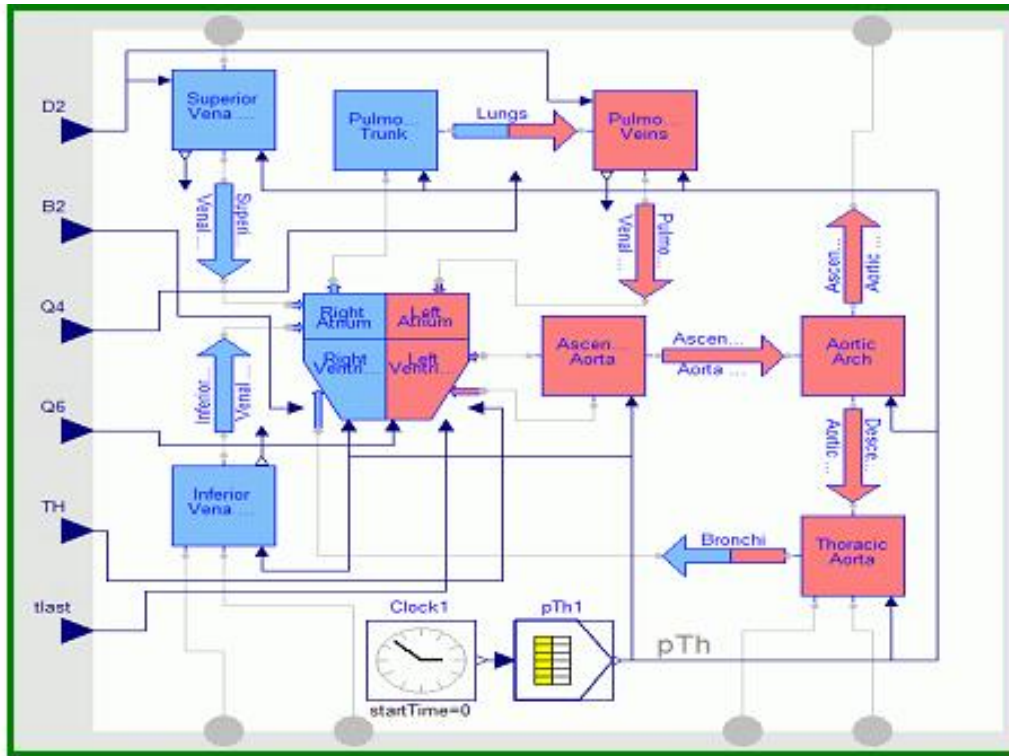
Modelica in Automotive Industry



Modelica in Avionics



Modelica in Biomechanics



Application of Modelica in Robotics Models

Real-time Training Simulator for Flight, Driving

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

(Movie demo)

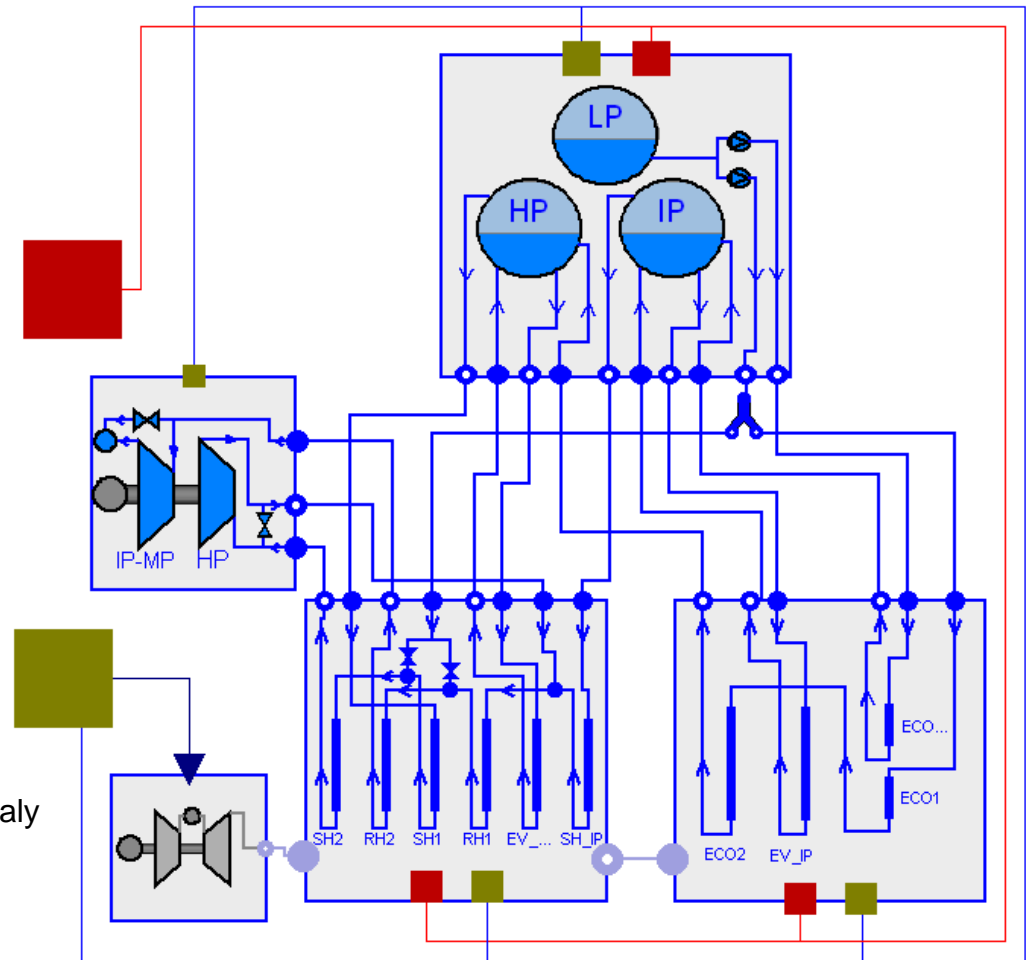


Courtesy of Tobias Bellmann, DLR,
Oberpfaffenhofen, Germany

Combined-Cycle Power Plant

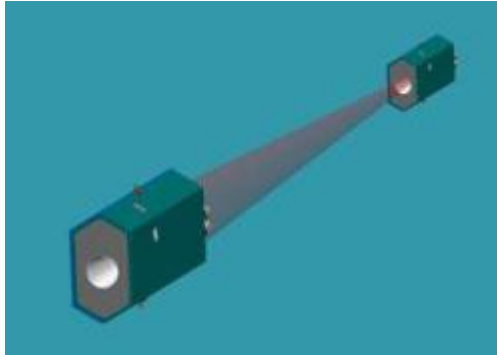
Plant model – system level

- GT unit, ST unit, Drum boilers unit and HRSG units, connected by thermo-fluid ports and by signal buses
- Low-temperature parts (condenser, feedwater system, LP circuits) are represented by trivial boundary conditions.
- GT model: simple law relating the electrical load request with the exhaust gas temperature and flow rate.



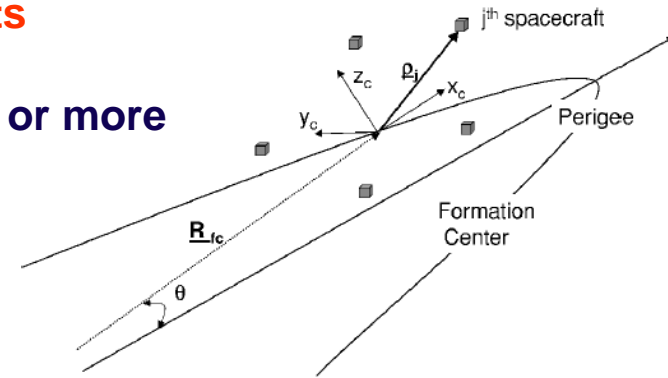
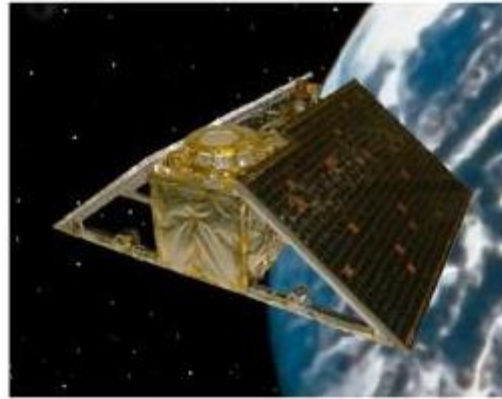
Courtesy Francesco Casella, Politecnico di Milano – Italy
and Francesco Pretolani, CESI SpA - Italy

Modelica Spacecraft Dynamics Library



Formation flying on elliptical orbits

Control the relative motion of two or more spacecraft



Attitude control for satellites using magnetic coils as actuators

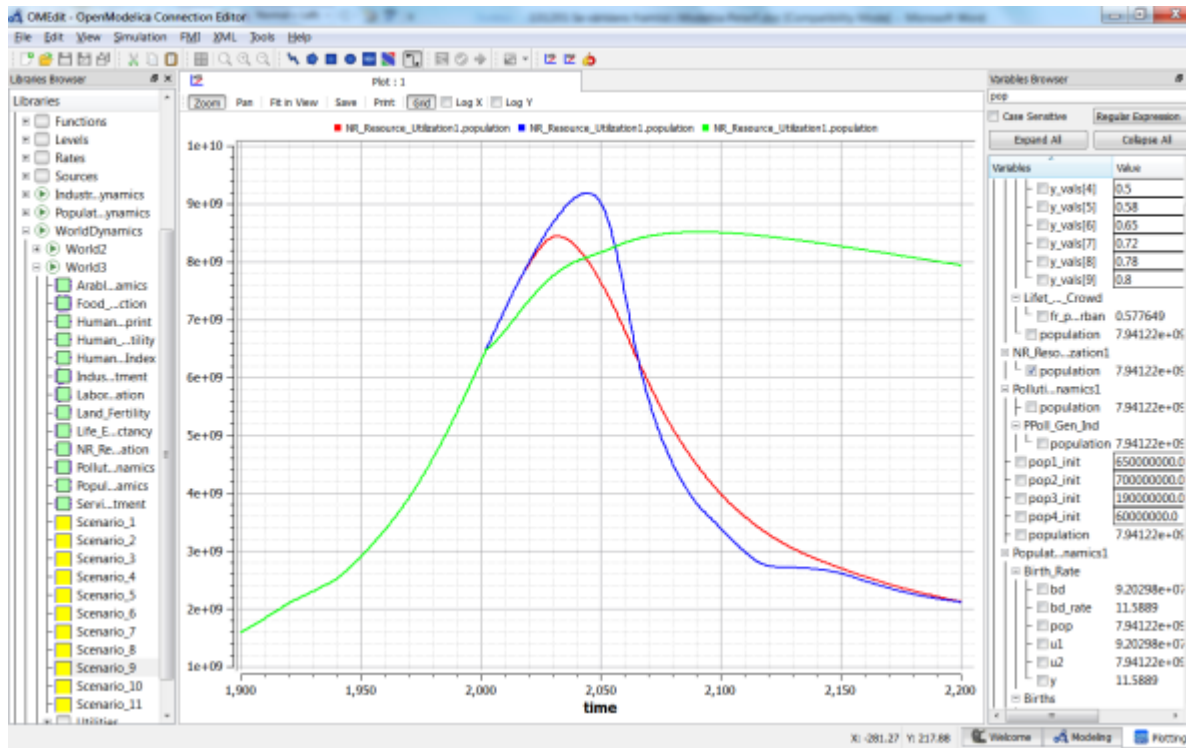
Torque generation mechanism: interaction between coils and geomagnetic field

Courtesy of Francesco Casella, Politecnico di Milano, Italy



System Dynamics – World Society Simulation

Limits to Material Growth; Population, Energy and Material flows



Left. World3 simulation with OpenModelica

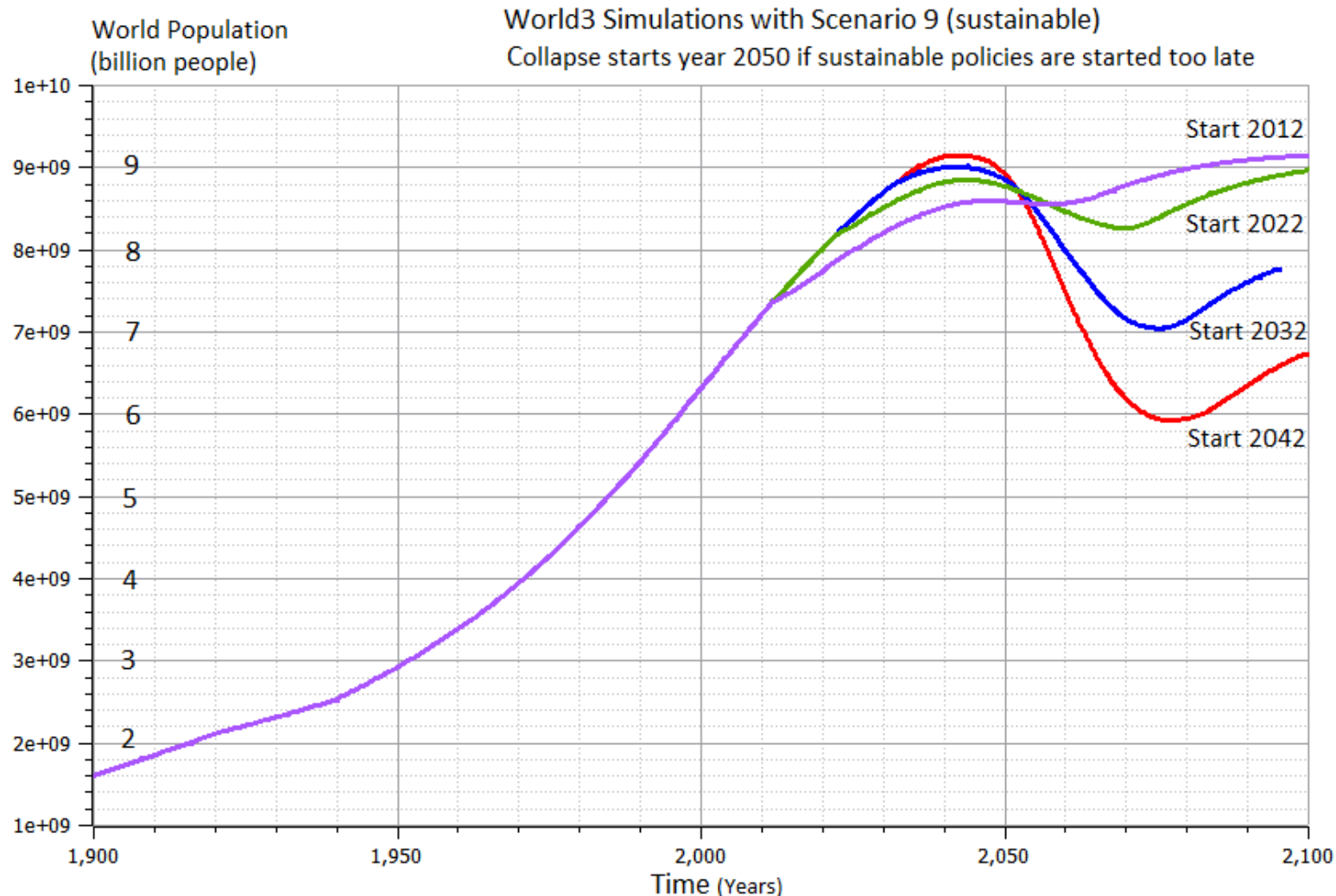
- 2 collapse scenarios (close to current developments)
- 1 sustainable scenario (green).

CO2 Emissions per person:

- USA 17 ton/yr
- Sweden 7 ton/yr
- India 1.4 ton/yr
- Bangladesh 0.3 ton/yr

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

World3 Simulations with Different Start Years for Sustainable Policies – Collapse if starting too late



LIMITS TO GROWTH



The 30-Year Update

DONELLA MEADOWS | JORGEN RANDERS | DENNIS MEADOWS

THE NEW YORK TIMES BESTSELLER

COLLAPSE

HOW SOCIETIES CHOOSE

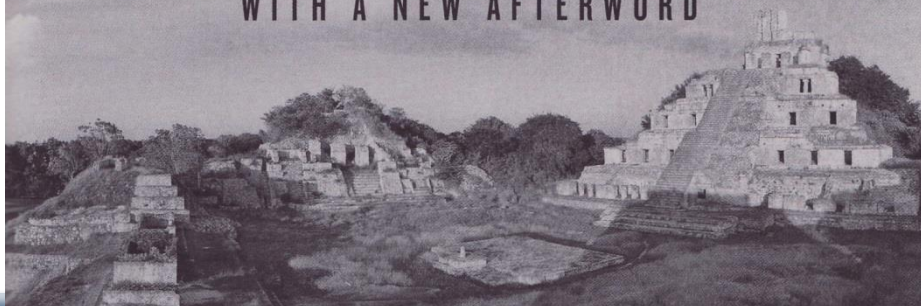
TO FAIL OR SUCCEED

JARED DIAMOND

author of the Pulitzer Prize-winning

GUNS, GERMS, and STEEL

WITH A NEW AFTERWORD



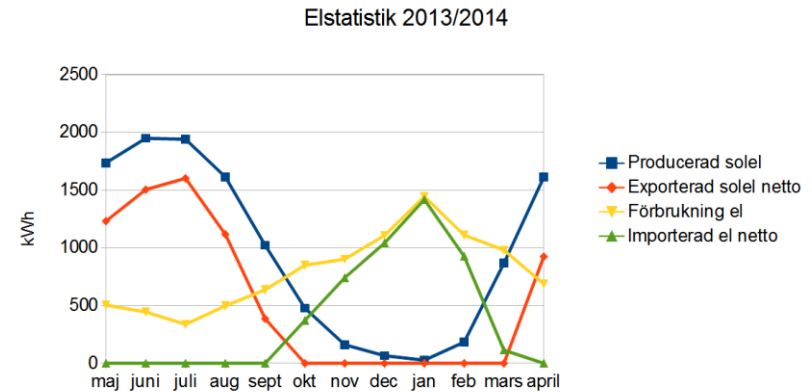
What Can You Do?

Need Global Sustainability Mass Movement

- Develop smart Cyber-Physical systems for reduced energy and material footprint
- Model-based circular economy for re-use of products and materials
- Promote sustainable lifestyle and technology
- Install electric solar PV panels
- Buy shares in cooperative wind power



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



Expanded to 93 sqm, 12 kW, March 2013
House produced 11600 kWh, used 9500 kWh
Avoids 10 ton CO₂ emission per year

Example Electric Cars

Can be charged by electricity from own solar panels



- Renault ZOE; 5 seat; Range:**
22kw (2014) vs 40 kw battery (2017)
- **EU-drive cycle 210 km, now 400 km**
 - **Realistic Swedish drive cycle:**
 - **Summer: 165 km, now 300 km**
 - **Winter: 110 km, now 200 km**
- Cheap fast AC chargers (22kw, 43kw)**



Tesla model S
range 480 km



DLR ROboMObil

- **experimental electric car**
- **Modelica models**

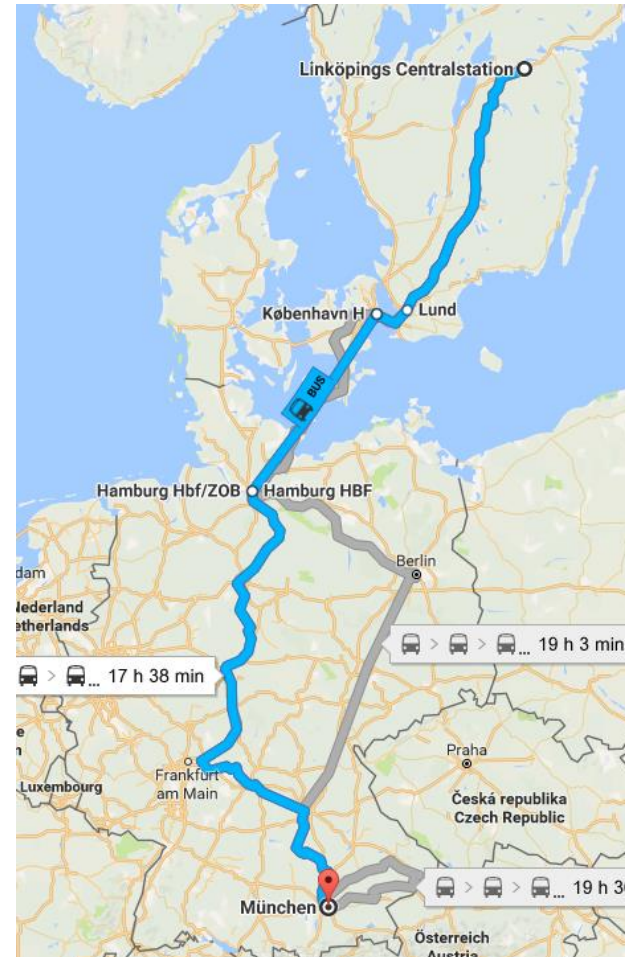
What Can You Do?

More Train Travel – Less Air Travel

- Air travel by Swedish Citizens – about the same emissions as all personal car traffic in Sweden!
- By train from Linköping to Munich and back – saves almost 1 ton of CO₂e emissions compared to flight
- Leave Linköping 07.00 in Munich 23.14

More Examples, PF travel 2016:

- Train Linköping-Paris, Dec 3-6, EU project meeting
- Train Linköping-Dresden, Dec 10-16, 1 week workshop



Train
travel
Linköping
- Munich

A satellite-style world map showing the continents. Overlaid on the map are several small, semi-transparent yellow rectangles. One rectangle is located in the southwestern United States. Another is in the Amazon basin of South America. A third is in the Sahel region of Africa. A fourth is in the Middle East. A fifth is in India. A sixth is in Southeast Asia. A seventh is in Australia. A large yellow text box is positioned in the upper center of the map.

Small rectangles – surface needed for 100% solar energy for humanity

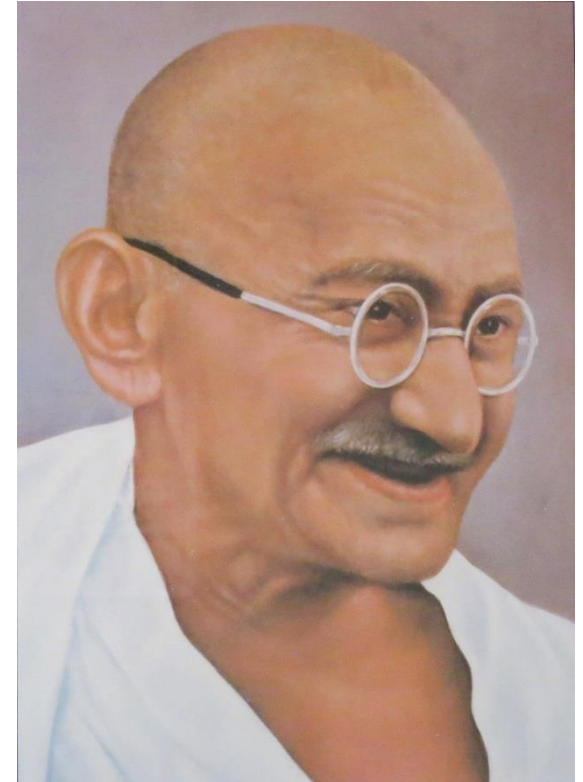
Sustainable Society Necessary for Human Survival

Almost Sustainable

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

Non-sustainable

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



Gandhi – role model for future less materialistic life style

Brief Modelica History

- First Modelica design group meeting in fall 1996
 - International group of people with expert knowledge in both language design and physical modeling
 - Industry and academia
- Modelica Versions
 - 1.0 released September 1997
 - 2.0 released March 2002
 - 2.2 released March 2005
 - 3.0 released September 2007
 - 3.1 released May 2009
 - 3.2 released March 2010
 - 3.3 released May 2012
 - 3.2 rev 2 released November 2013
 - 3.3 rev 1 released July 2014
 - 3.4 planned spring 2017
- Modelica Association established 2000 in Linköping
 - Open, non-profit organization

Modelica Conferences

- The 1st International Modelica conference October, 2000
- The 2nd International Modelica conference March 18-19, 2002
- The 3rd International Modelica conference November 5-6, 2003 in Linköping, Sweden
- The 4th International Modelica conference March 6-7, 2005 in Hamburg, Germany
- The 5th International Modelica conference September 4-5, 2006 in Vienna, Austria
- The 6th International Modelica conference March 3-4, 2008 in Bielefeld, Germany
- The 7th International Modelica conference Sept 21-22, 2009 in Como, Italy
- The 8th International Modelica conference March 20-22, 2011 in Dresden, Germany
- The 9th International Modelica conference Sept 3-5, 2012 in Munich, Germany
- The 10th International Modelica conference March 10-12, 2014 in Lund, Sweden
- The 11th International Modelica conference Sept 21-23, 2015 in Versailles, Paris
- The 12th International Modelica conference planned May 15-17, 2017 in Prague, Czech Republic

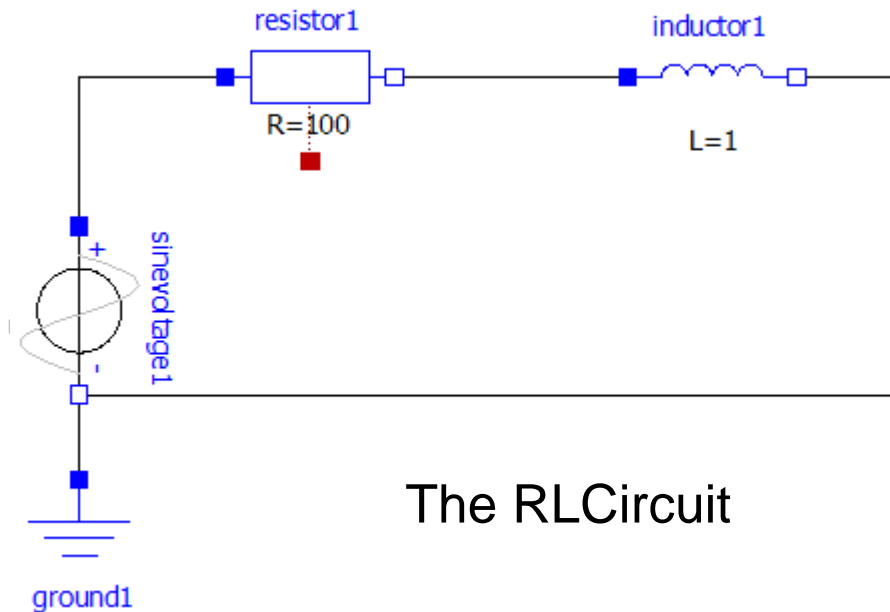
Exercises Part I

Hands-on graphical modeling

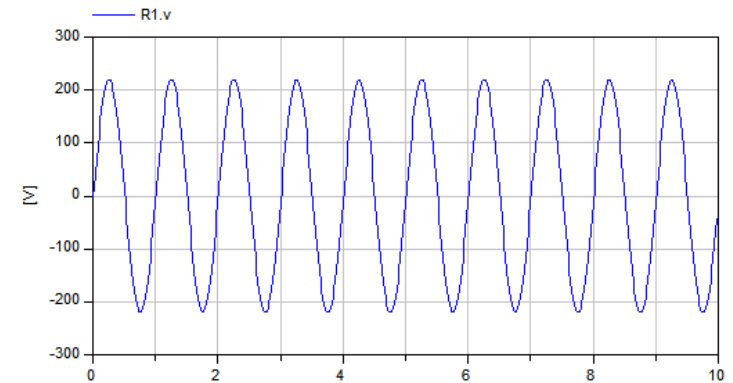
(15 minutes)

Exercises Part I – Basic Graphical Modeling

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



The RLCircuit




Simulation

Exercises Part I – OMEdit Instructions (Part I)

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

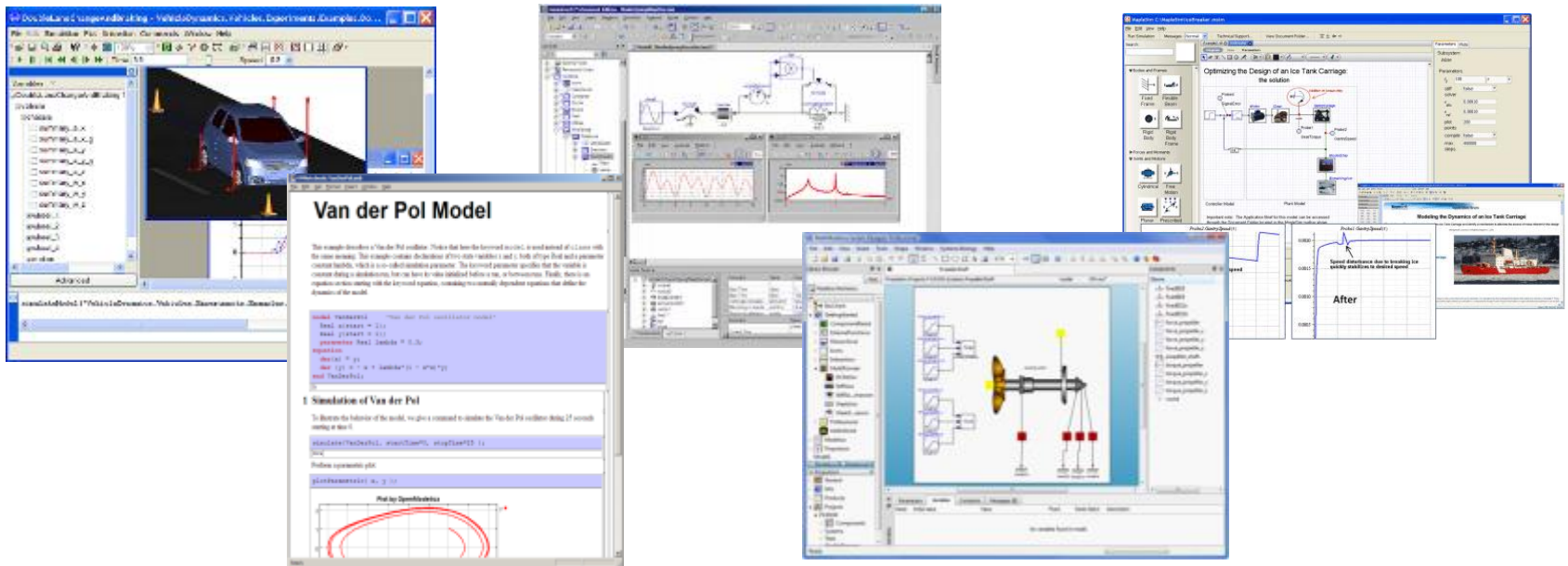
- Under the **Modelica Library**:
 - Contains The standard Modelica library components
 - The **Modelica files** contains the list of models you have created.

Exercises Part I – OMEdit Instructions (Part II)

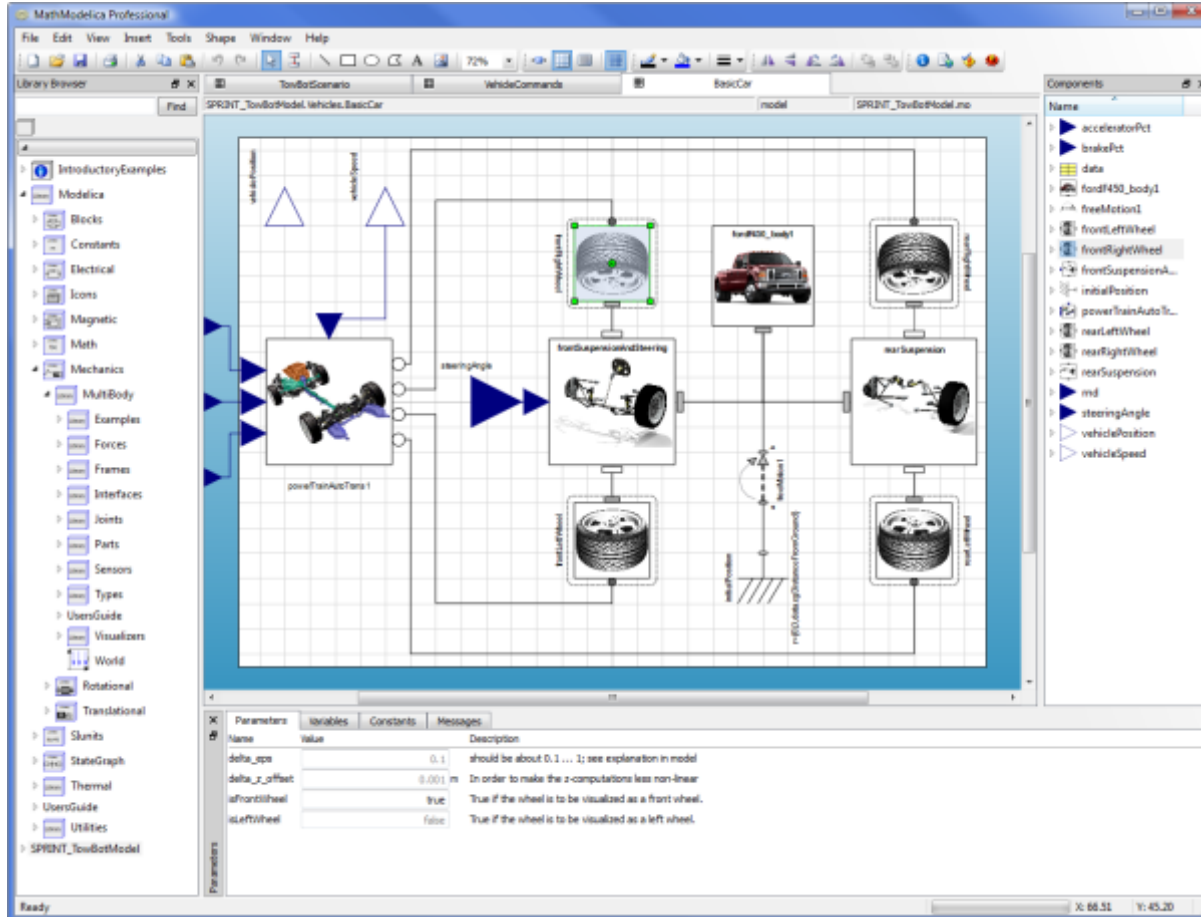
- For the RLCircuit model, browse the Modelica standard library and add the following component models:
 - Add `Ground`, `Inductor` and `Resistor` component models from `Modelica.Electrical.Analog.Basic` package.
 - Add `SineVoltage` component model from `Modelica.Electrical.Analog.Sources` package.
- Make the corresponding connections between the component models as shown in the previous slide.
- Simulate the model
 - Go to Simulation menu and choose simulate or click on the simulate button in the toolbar. 
- Plot the instance variables
 - Once the simulation is completed, a plot variables list will appear on the right side. Select the variable that you want to plot.

Part II

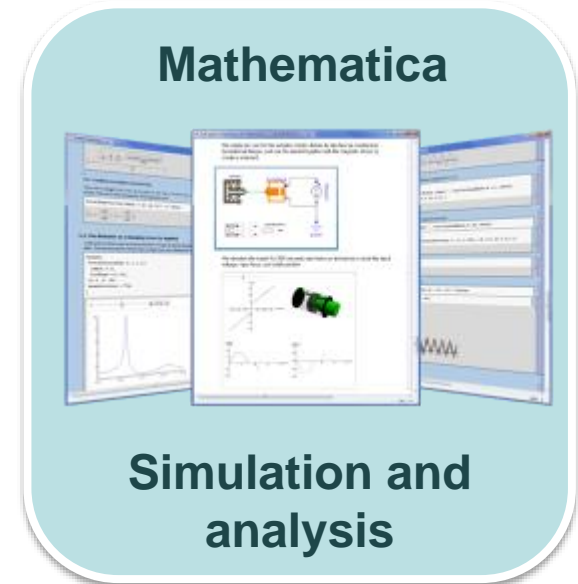
Modelica environments and OpenModelica



Wolfram System Modeler – Wolfram MathCore



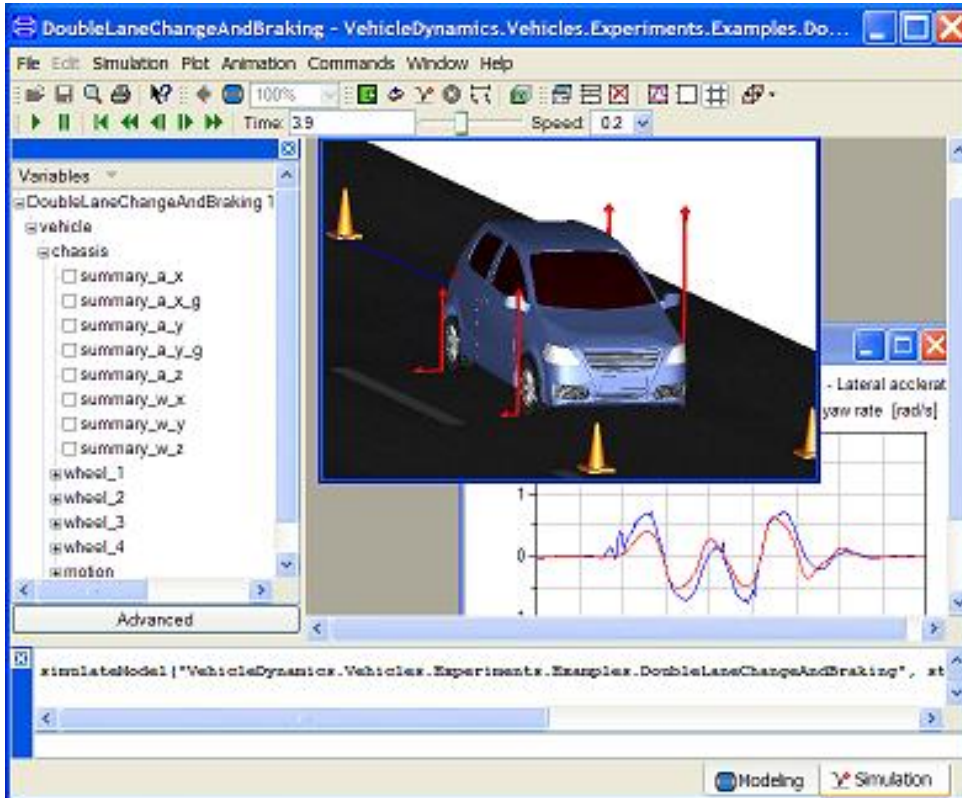
- Wolfram Research
- USA, Sweden
- General purpose
- Mathematica integration
- www.wolfram.com
- www.mathcore.com



Car model graphical view

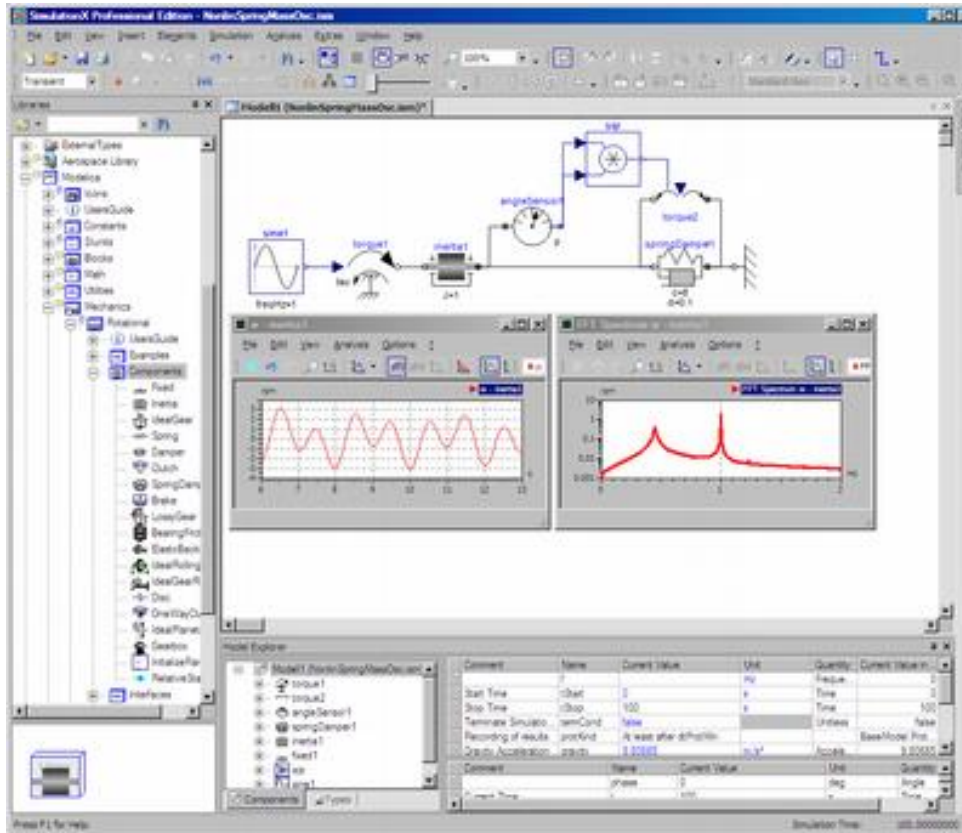
Courtesy
Wolfram
Research

Dymola



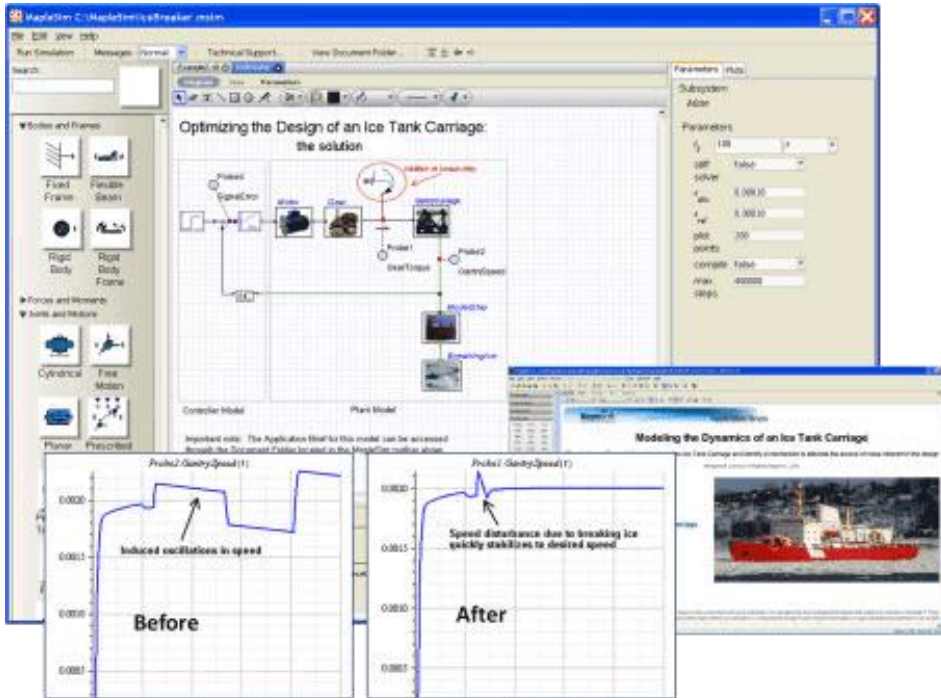
- Dassault Systemes Sweden
- Sweden
- First Modelica tool on the market
- Initial main focus on automotive industry
- www.dymola.com

Simulation X



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

MapleSim



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

The OpenModelica Environment

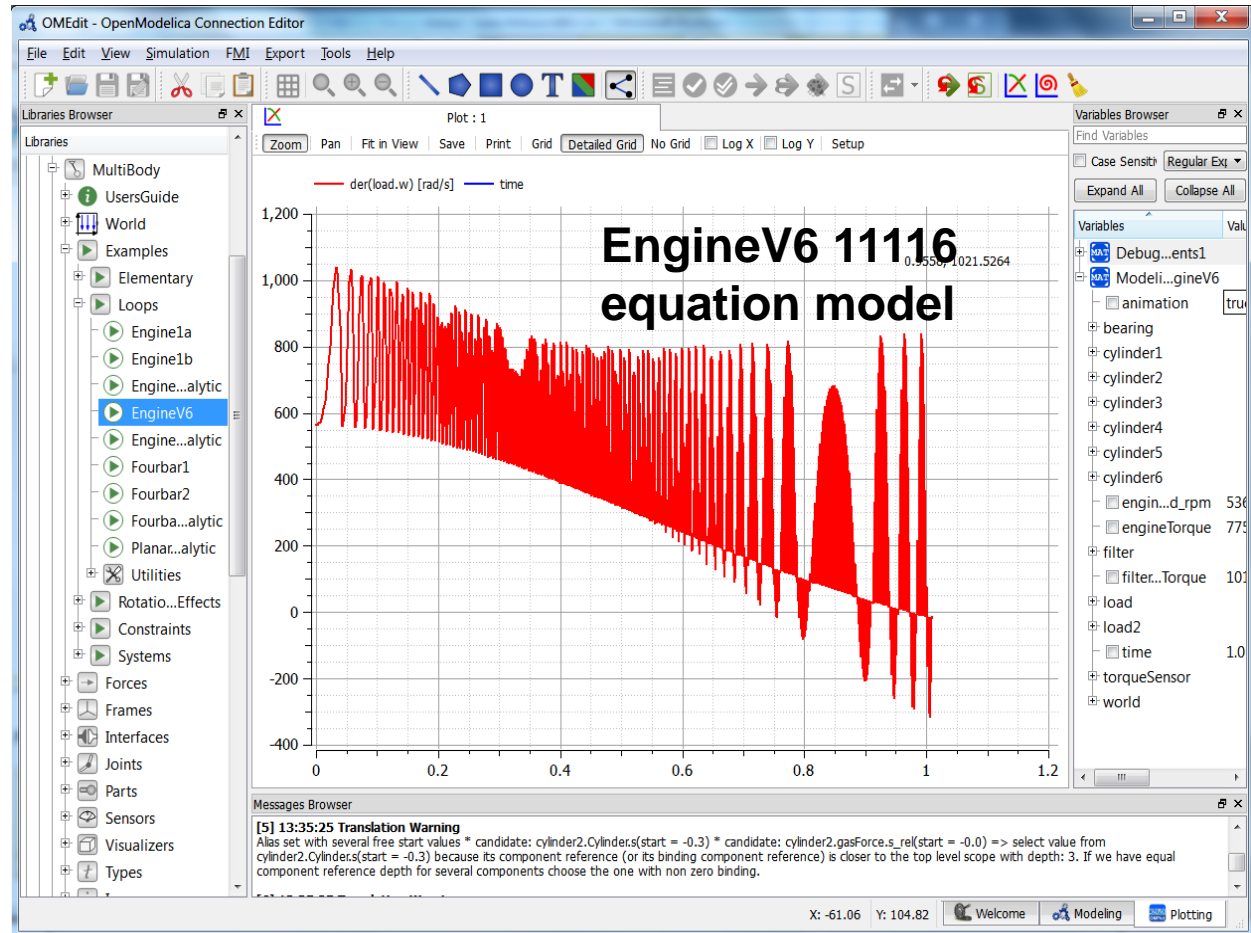
www.OpenModelica.org

The screenshot shows the OpenModelica website homepage. At the top, there is a navigation bar with the OpenModelica logo, a search bar, and links for 'Login' and 'Create an account'. Below the navigation bar is a main content area with several sections:

- Top information:** A sidebar on the left containing links for 'Industrial Products' (Commercial Applications using OpenModelica), 'OMEdit' (Enhanced OpenModelica Connection Editor), and 'Library Coverage' (Latest library coverage).
- Introduction:** A central text block stating that OPENMODELICA is an open-source Modelica-based modeling and simulation environment intended for industrial and academic usage. It is supported by the Open Source Modelica Consortium (OSMC). The goal is to create a comprehensive Open Source Modelica modeling, compilation and simulation environment based on free software distributed in binary and source code form for research, teaching, and industrial usage. It invites researchers and students, or any interested developer to participate in the project and cooperate around OpenModelica, tools, and applications.
- Latest news:** A sidebar on the right listing recent releases and events, including: October 25, 2014: OpenModelica 1.9.1 released; Preliminary Program OpenModelica Annual Workshop 2015; October 07, 2014: OpenModelica 1.9.1 Beta4 released; March 08, 2014: OpenModelica 1.9.1 Beta2 released; New Book: Peter Fritzson - Principles of Object-Oriented Modeling and Simulation with Modelica 3.3; February 02, 2014: OpenModelica 1.9.1 Beta1 released; CFP OpenModelica Workshop February 2014; October 09, 2013: OpenModelica 1.9.0 released; September 27, 2013: OpenModelica 1.9.0 RC1 released; February 1, 2013: OpenModelica 1.9.0 Beta4 released.
- Modelica/OpenModelica Videos:** A section at the bottom left featuring a video player with the title 'Overview of Modelica, an...'. Below the video player, there are links to 'Register yourself to get information about new releases.', 'Participate in the OpenModelicaInterest mailing list.', 'Help us: get the latest source code or nightly-build and report bugs.', 'To learn about Modelica, read a book or a tutorial about Modelica®.', and 'For systems engineering with requirement traceability and verification, see ModelicaML.'

OpenModelica – Free Open Source Tool developed by the Open Source Modelica Consortium (OSMC)

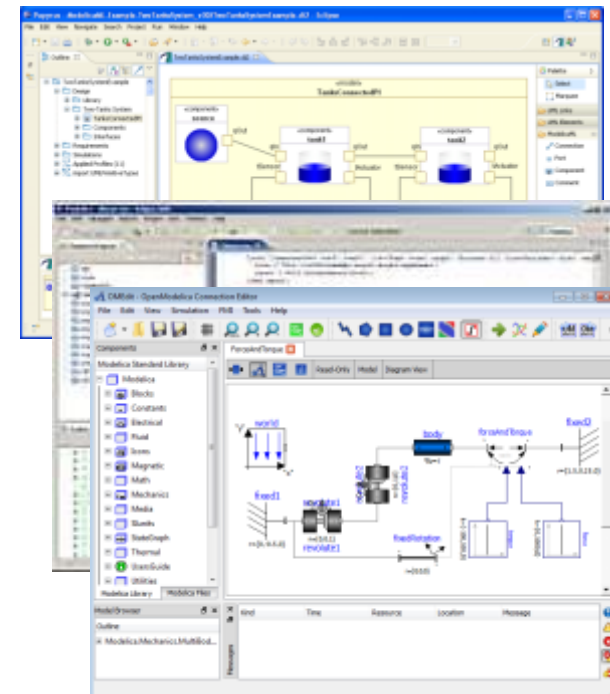
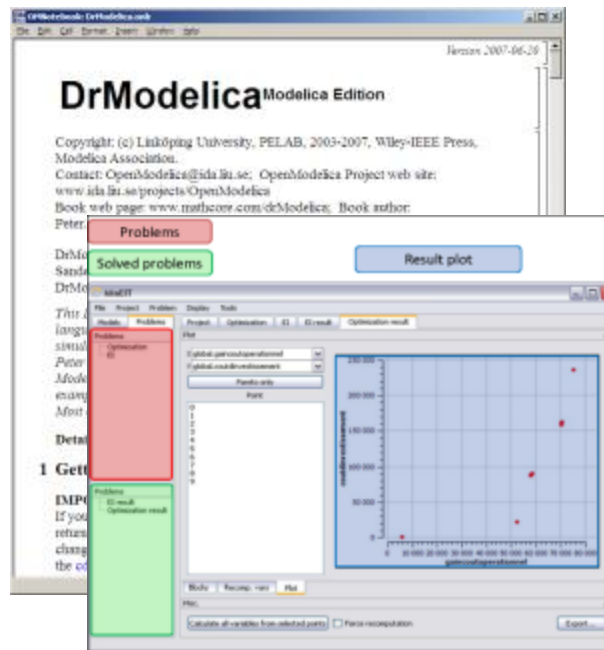
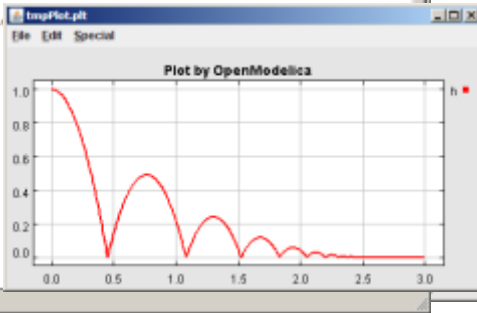
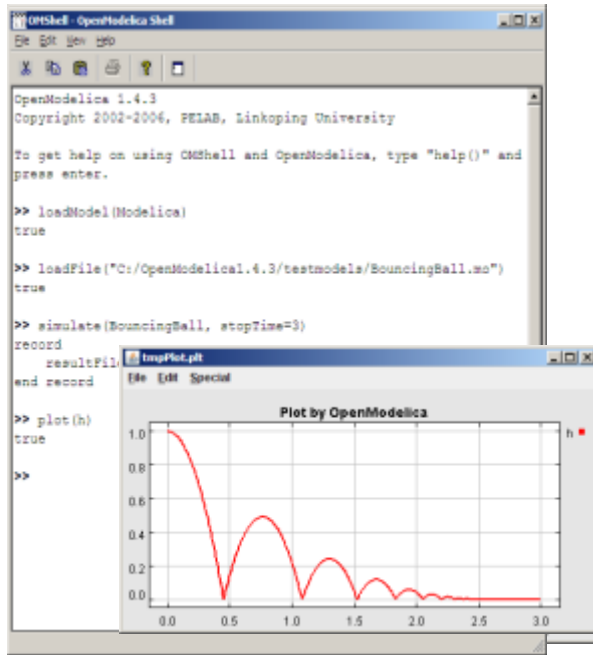
- Graphical editor
- Model compiler and simulator
- Debugger
- Performance analyzer
- Dynamic optimizer
- Symbolic modeling
- Parallelization
- Electronic Notebook and OMWebbook for teaching
- Spokentutorial for teaching



The OpenModelica Open Source Environment

www.openmodelica.org

- **Advanced Interactive Modelica compiler (OMC)**
 - Supports most of the Modelica Language
 - **Modelica and Python scripting**
- **Basic environment for creating models**
 - **OMShell** – an interactive command handler
 - **OMNotebook** – a literate programming notebook
 - **MDT** – an advanced textual environment in Eclipse
- **OMEdit** graphic Editor
- **OMDebugger** for equations
- **OMOptim** optimization tool
- **OM Dynamic optimizer** collocation
- **ModelicaML** UML Profile
- **MetaModelica** extension
- **ParModelica** extension



OSMC – International Consortium for Open Source Model-based Development Tools, 46 members Dec 2016

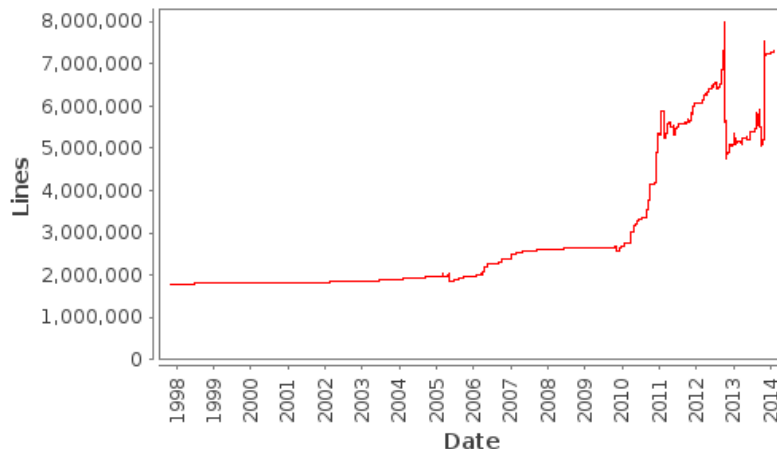
Founded Dec 4, 2007

Open-source community services

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

Code Statistics

/trunk: Lines of Code



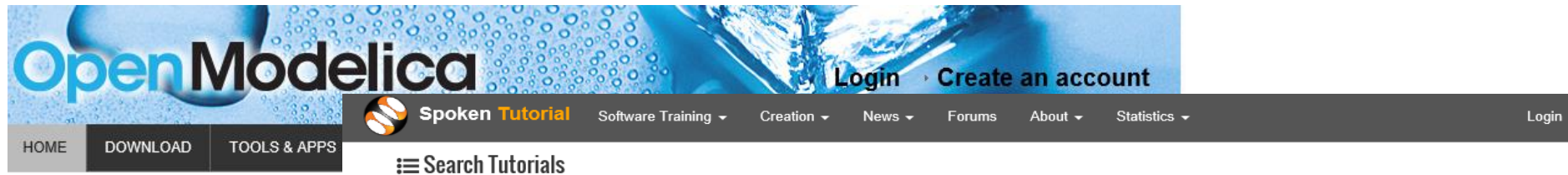
Industrial members

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

University members

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

Interactive OpenModelica Step-by-step Spoken-Tutorial using OMEdit. Link from www.openmodelica.org



To learn about Modelica, read a [book](#) or a [tutorial](#) about [Modelica®](#).
Interactive step-by-step beginners Modelica [on-line spoken tutorials](#)
Interactive [OMWebbook](#) with examples of Modelica textual modeling

[Reset dropdowns](#)

OpenModelica is an open source modelling and simulation environment intended for industrial and academic usage. It is an object oriented declarative multi domain modelling language for complex systems. This environment can be used to work for both steady state as well as dynamic systems. Attractive strategy when dealing with design and optimization problems. As all the equations are solved simultaneously it doesn't matter whether the unknown variable in an input or output variable. [Read more](#)

About 12 results found.

[Instruction Sheet](#)



1. Introduction to OMEdit

Foss : *OpenModelica - English*

Outline: Introduction to OpenModelica Introduction to OMEdit Perspectives in OMEdit Browsers in OMEdit View icons in OMEdit Open a Class from Libraries Browser Checking for correctness..

Basic
■

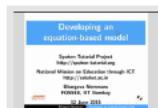


2. Examples through OMEdit

Foss : *OpenModelica - English*

Outline: Expand Modelica library Expand Electrical library Expand Analog library Open Rectifier Class Compare the values of IDC & Losses time vs Losses plot Expand Mechanics library ..

Basic
■



3. Developing an equation-based model

Foss : *OpenModelica - English*

Outline: Introduction to OMEdit Declaration of variables and equations Simulation of a model in

Basic
■

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)



The screenshot shows the OMNotebook application window titled "OMNotebook: DrModelica.omb*". The window contains the following content:

Version 2006-04-11

DrModelica Modelica Edition

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

DrModelica Authors: (2003 version) Susanna Monemar, Eva-Lena Lengquist-Sandelin, Peter Fritzson, Peter Bounus
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 0-471-47163-1. 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 sense. 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

OMnotebook Interactive Electronic Notebook Here Used for Teaching Control Theory

1 Kalman Filter

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

Another difficulty is that the measured quantities of

$$\begin{cases} \hat{x} \\ y \end{cases}$$

Here e denoting a disturbance in the input signal is to be evaluated by the difference

$$K(y(t) - \hat{y}(t))$$

By using this quantity as feedback we obtain the observer

$$\dot{\hat{x}} = A\hat{x}(t) + Bu(t) + K(y(t) - \hat{y}(t))$$

Now form the error as

The differential error is

The screenshot shows the OMNotebook interface with a Modelica model named 'KalmanFeedback'. The model defines parameters for a system with state space matrices A, B, C, K, L, and noise matrices Anew, Bnew, Fnew. It also defines state space noise for both the Kalman filter and the reference system. The simulation is run for 3 seconds, and the results are plotted, showing the state space output y[1] for both the Kalman filter and the reference system. The plot shows that the Kalman filter's output (red line) closely follows the reference system's output (blue line), indicating successful state estimation.

```

model KalmanFeedback
  parameter Real A[:,size(A, 1)] = {{0,1},{1,0}};
  parameter Real B[size(A, 1),:] = {{0},{1}};
  parameter Real C[:,size(A, 1)] = {{1,0}};
  parameter Real[2,1] K = {2.4;3.4};
  parameter Real[1,2] L = {2.4,3.4};
  parameter Real[:,:] ABL = A-B*K;
  parameter Real[:,:] BL = B*L;
  parameter Real[:,:] Z = zeros(size(ABL,2),size(ACL,1));
  parameter Real[:,:] ACL = A-K*C;
  parameter Real[:,:] Anew = [0,1,0,0; -1.4, -3.4, 2.4,3.4; 0,0,-2.4,1;0,0,-2.4,0];
  parameter Real[:,:] Bnew = [0;1;0;0];
  parameter Real[:,:] Fnew = [1;0;0;0];
  stateSpaceNoise Kalman(stateSpace.A=ABL,stateSpace.B=Bnew, stateSpace.C=[1,0,0,0],
stateSpace.F = Fnew);
  stateSpaceNoise noKalman;
end KalmanFeedback;

simulate(KalmanFeedback,stopTime=3)

plot({Kalman.stateSpace.y[1],noKalman.stateSpace.y[1]})
  
```

Plot by OpenModelica

Legend:

- Kalman.stateSpace.y[1] (Red line)
- noKalman.stateSpace.y[1] (Blue line)

OM Web Notebook Generated from OMNotebook

Edit, Simulate, Plot Models on a Web Page

<http://omwebbook.openmodelica.org/>

OMNotebook

OMwebbook

OMNotebook HelloWorld.onb

First Basic Class

1 HelloWorld

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

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

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

2 Simulation of HelloWorld

```
simulate( HelloWorld, startTime=0, stopTime=3 )
record SimulationResult
  resultFile = "HelloWorld_res.mat",
  messages = ""
end SimulationResult;
```

Plot the results.

```
plot( x )
```

[done]

Zoom Pan Auto Scale Fit in View Save Print Grid Detailed Grid No Grid Log X Log Y Setup

Plot by OpenModelica

time	x
0	1.0
0.5	0.61
1.0	0.37
1.5	0.22
2.0	0.14
2.5	0.08
3.0	0.05
3.5	0.03
4.0	0.02

Ready

OMWebBook

omwebbook.openmodelica.org

Evaluate Cell Eval All

First Basic Class

1 HelloWorld

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

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

```
1 class HelloWorld
2   Real x(start = 1, fixed=true);
3   parameter Real a = 1;
4 equation
5   der(x) = - a * x;
6 end HelloWorld;
7
```

2 Simulation of HelloWorld

```
1 simulate( HelloWorld, startTime=0, stopTime=4 )
2
```

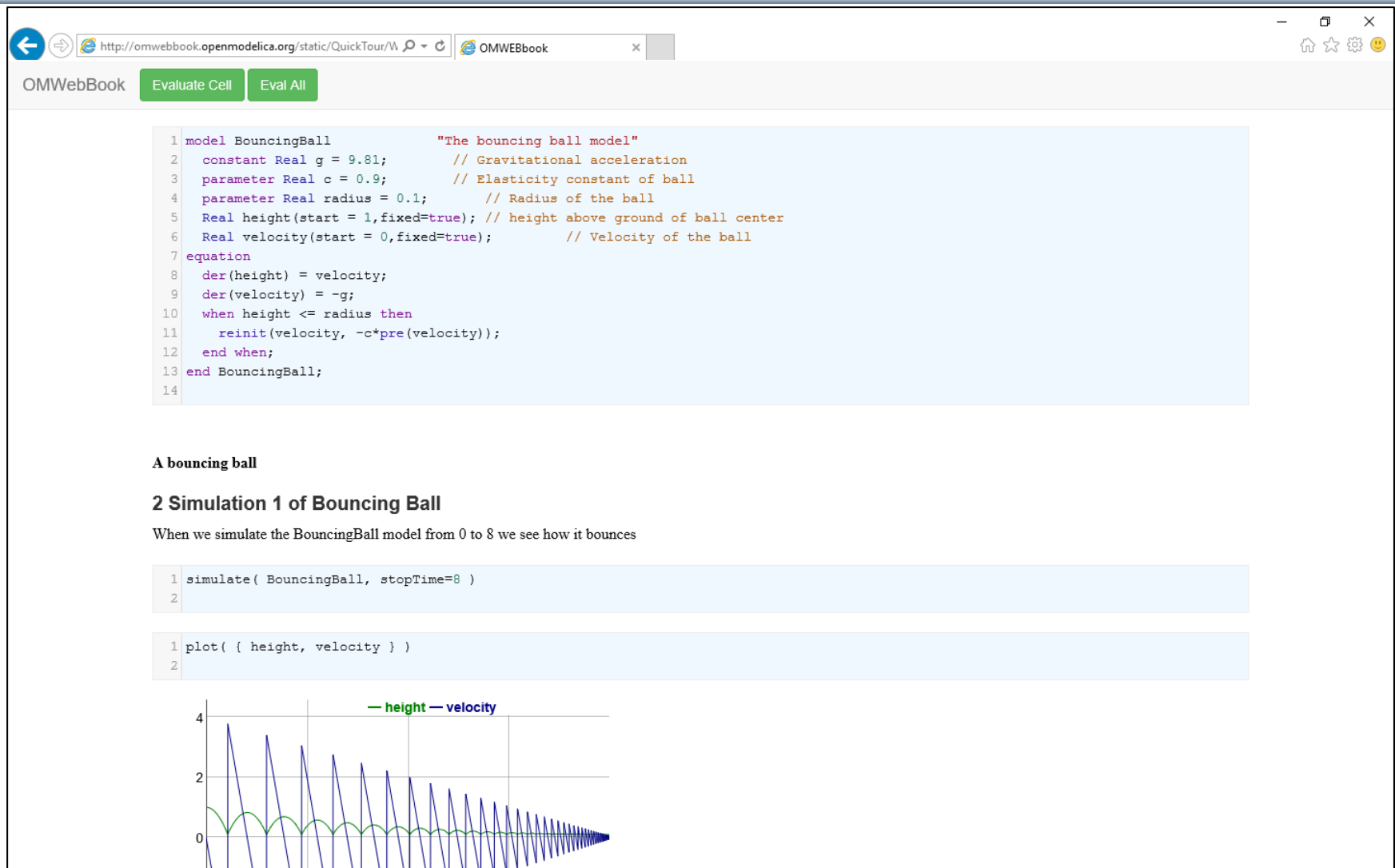
Plot the results.

```
1 plot( x )
2
```

time	x
0	1.0
1	0.37
2	0.14
3	0.05
4	0.02

BouncingBall Example of Using OMWebbook

Editing and Simulating the BouncingBall model



The screenshot shows the OMWebbook interface. At the top, there is a browser address bar with the URL `http://omwebbook.openmodelica.org/static/QuickTour/W` and a tab titled "OMWEBbook". Below the browser, there are two buttons: "Evaluate Cell" and "Eval All".

```
1 model BouncingBall           "The bouncing ball model"
2   constant Real g = 9.81;     // Gravitational acceleration
3   parameter Real c = 0.9;    // Elasticity constant of ball
4   parameter Real radius = 0.1; // Radius of the ball
5   Real height(start = 1, fixed=true); // height above ground of ball center
6   Real velocity(start = 0, fixed=true); // Velocity of the ball
7 equation
8   der(height) = velocity;
9   der(velocity) = -g;
10  when height <= radius then
11    reinit(velocity, -c*pre(velocity));
12  end when;
13 end BouncingBall;
14
```

A bouncing ball

2 Simulation 1 of Bouncing Ball

When we simulate the BouncingBall model from 0 to 8 we see how it bounces

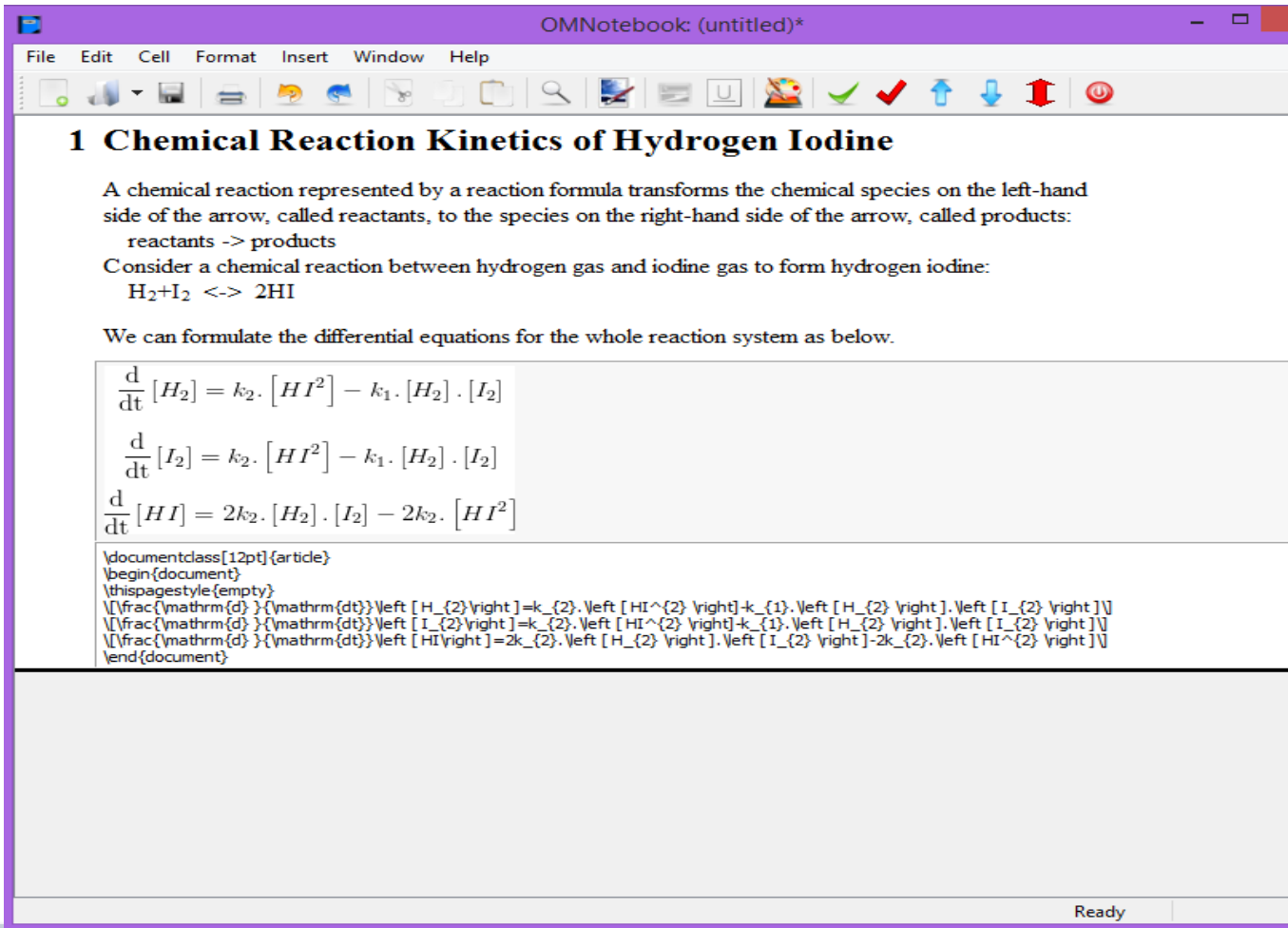
```
1 simulate( BouncingBall, stopTime=8 )
2
```

```
1 plot( { height, velocity } )
2
```

The plot shows two variables over time from 0 to 8. The green line represents "height" and the blue line represents "velocity". The height starts at 1 and bounces, with each bounce reaching a lower peak. The velocity starts at 0 and oscillates between positive and negative values, with the amplitude decreasing over time. The legend indicates "— height — velocity".

Mathematical Typesetting in OMNotebook and OMWebbook

OMNotebook supports Latex formatting for mathematics



The screenshot shows the OMNotebook application window titled "OMNotebook: (untitled)*". The menu bar includes File, Edit, Cell, Format, Insert, Window, and Help. The toolbar contains various icons for file operations, editing, and navigation. The main content area displays the following text and equations:

1 Chemical Reaction Kinetics of Hydrogen Iodine

A chemical reaction represented by a reaction formula transforms the chemical species on the left-hand side of the arrow, called reactants, to the species on the right-hand side of the arrow, called products:
reactants \rightarrow products

Consider a chemical reaction between hydrogen gas and iodine gas to form hydrogen iodine:
 $H_2 + I_2 \leftrightarrow 2HI$

We can formulate the differential equations for the whole reaction system as below.

$$\frac{d}{dt} [H_2] = k_2 \cdot [HI]^2 - k_1 \cdot [H_2] \cdot [I_2]$$
$$\frac{d}{dt} [I_2] = k_2 \cdot [HI]^2 - k_1 \cdot [H_2] \cdot [I_2]$$
$$\frac{d}{dt} [HI] = 2k_2 \cdot [H_2] \cdot [I_2] - 2k_1 \cdot [HI]^2$$

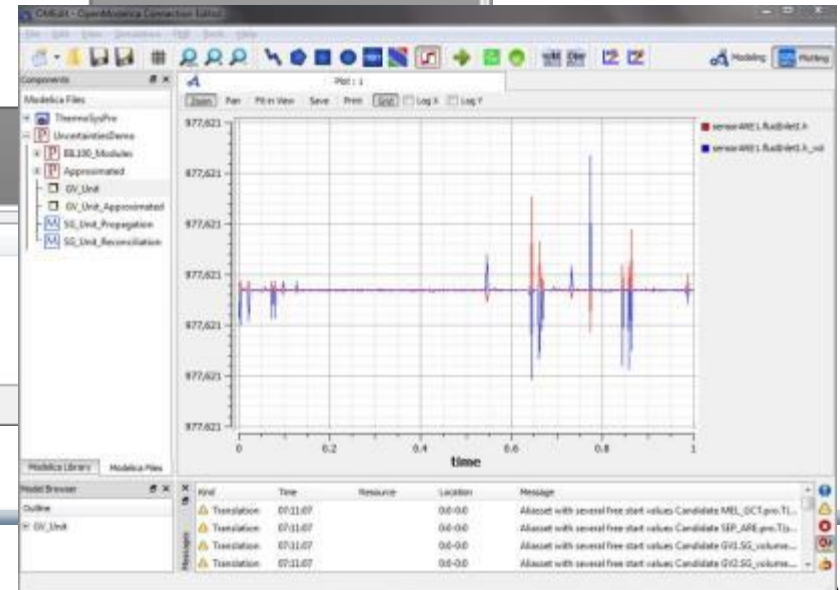
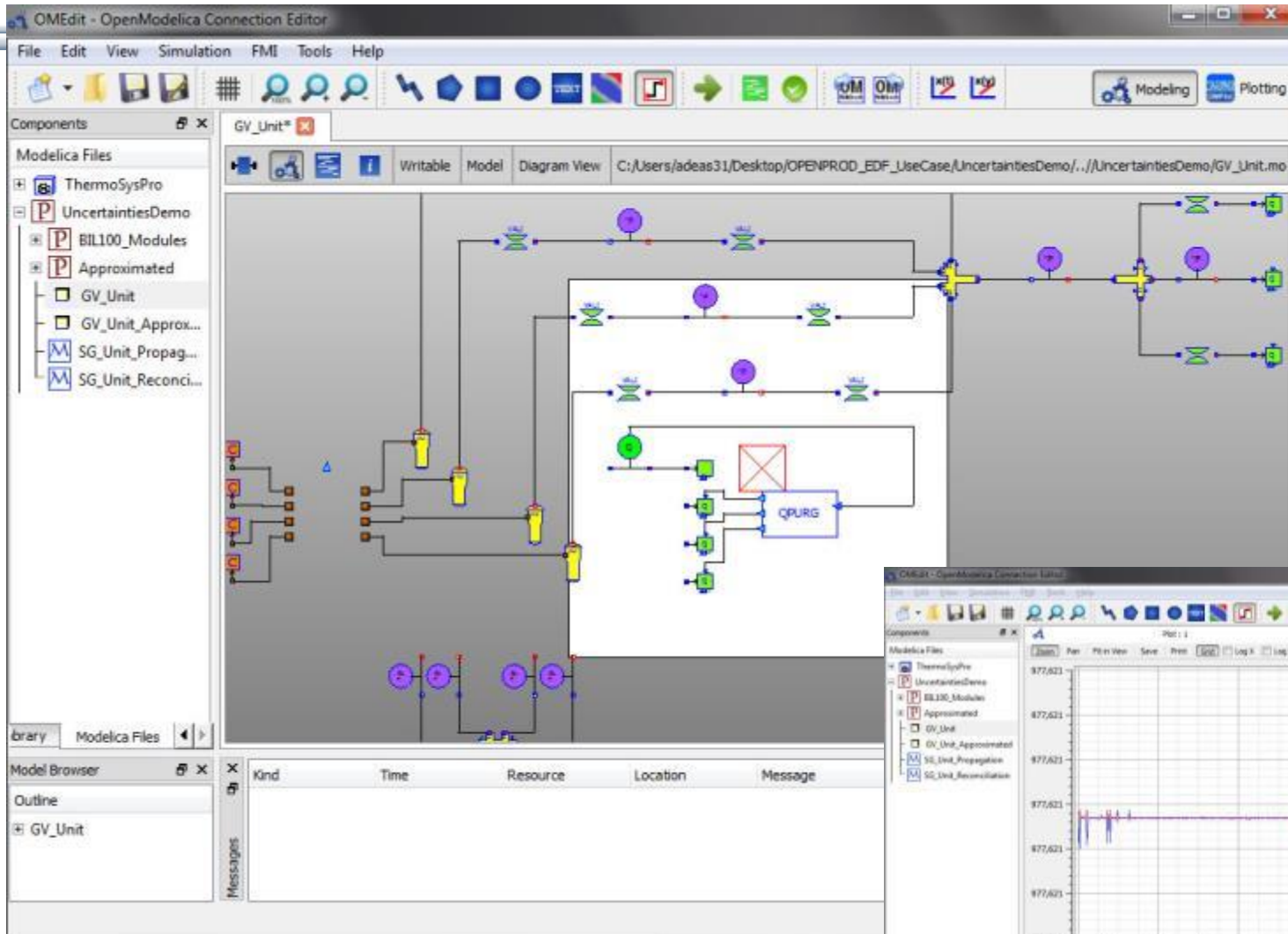
```
\documentclass[12pt]{article}
\begin{document}
\thispagestyle{empty}
\[\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{[H_2]}=k_2\cdot\mathrm{[HI]^2}-k_1\cdot\mathrm{[H_2]}\cdot\mathrm{[I_2]}\]
\[\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{[I_2]}=k_2\cdot\mathrm{[HI]^2}-k_1\cdot\mathrm{[H_2]}\cdot\mathrm{[I_2]}\]
\[\frac{\mathrm{d}}{\mathrm{dt}}\mathrm{[HI]}=2k_2\cdot\mathrm{[H_2]}\cdot\mathrm{[I_2]}-2k_1\cdot\mathrm{[HI]^2}\]
\end{document}
```

Ready

Contents in
OMWebbook
Generated from
OMNotebook

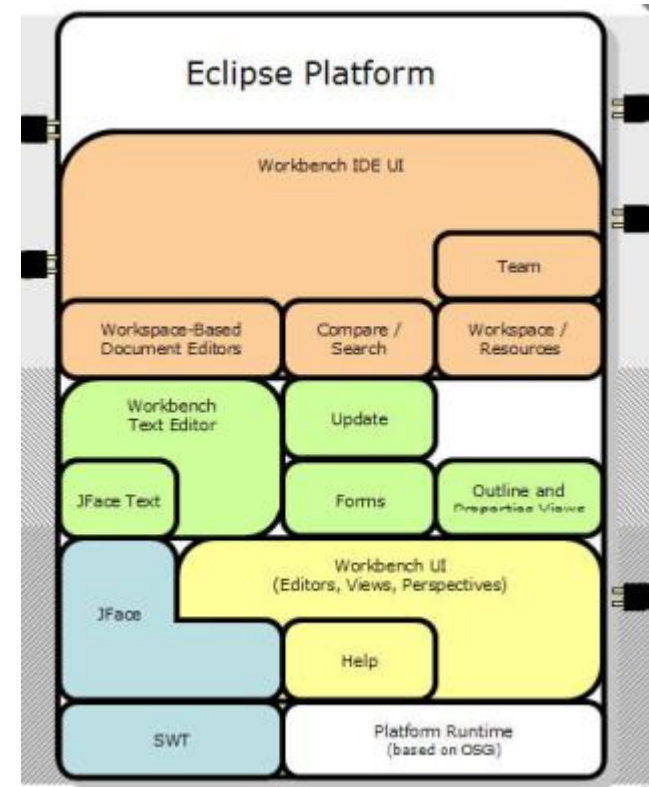
Latex instructions
can be hidden by
double clicking the
Cell in tree view

OpenModelica Environment Demo



OpenModelica MDT – Eclipse Plugin

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



OpenModelica MDT: Code Outline and Hovering Info

The screenshot displays the Eclipse IDE interface for OpenModelica. The top-left pane shows a project tree with files like `rml2sig`, `runtime`, `scripts`, `test_codegen`, `tools`, `VC7`, and `Absyn.mo`. The bottom-left pane shows a code outline for the `Absyn` module, listing various algorithmic constructs such as `ADD`, `ALG_ASSIGN`, `ALG_BREAK`, `ALG_CATCH`, `ALG_EQUALITY`, `ALG_FAILURE`, `ALG_FOR`, `ALG_GOTO`, `ALG_IF`, `ALG_LABEL`, `ALG_NORET_CALL`, `ALG_RETURN`, `ALG_THROW`, `ALG_TRY`, and `ALG_WHEN_A`.

The main editor window shows the `Absyn.mo` file with the following code snippet:

```
case (MATRIX(matrix = exp1))
  local list<list<list<ComponentRef>>> res1;
  equation
    res1 = Util.listListMap(exp1, getCrefFromExp);
    res2 = Util.listFlatten(res1);
    res = Util.listFlatten(res2);
  then
    res;
case (RANGE(start = e1, step = SOME(e3), stop = e2))
  equation
    l1 = getCrefFromExp(e1);
    l2 =
      function getCrefFromExp "function: getCrefFromExp
        Returns a flattened list of the
        component references in an expression"
        input Exp inExp;
        then
          output list<ComponentRef> outComponentRefList;
        algorithm
          outComponentRefList:=matchcontinue inExp
          local
            l1 =
              ComponentRef cr;
```

A tooltip is visible over the `getCrefFromExp` function definition, providing the following information:

- function: `getCrefFromExp`
- Returns a flattened list of the component references in an expression
- input: `Exp inExp;`
- output: `list<ComponentRef> outComponentRefList;`
- algorithm: `outComponentRefList:=matchcontinue inExp`
- local: `ComponentRef cr;`

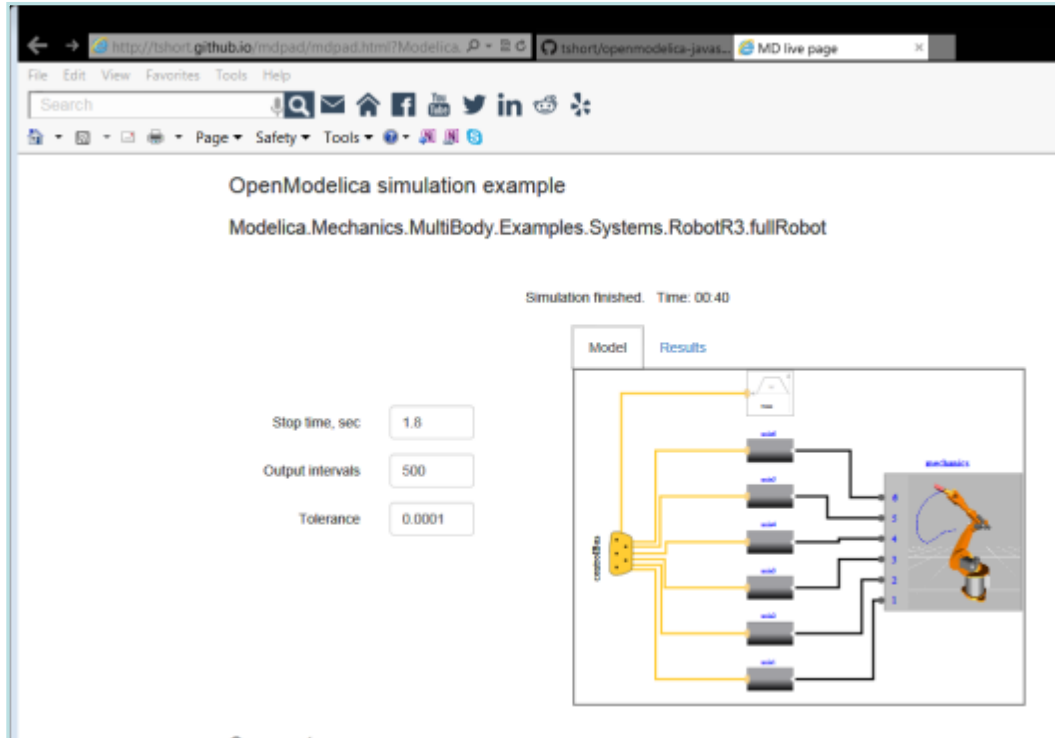
The bottom-right pane shows the `Problems` window with 113 errors, 0 warnings, and 0 infos. The error description is: "The identifier at start and end are different".

Two callout boxes highlight key features:

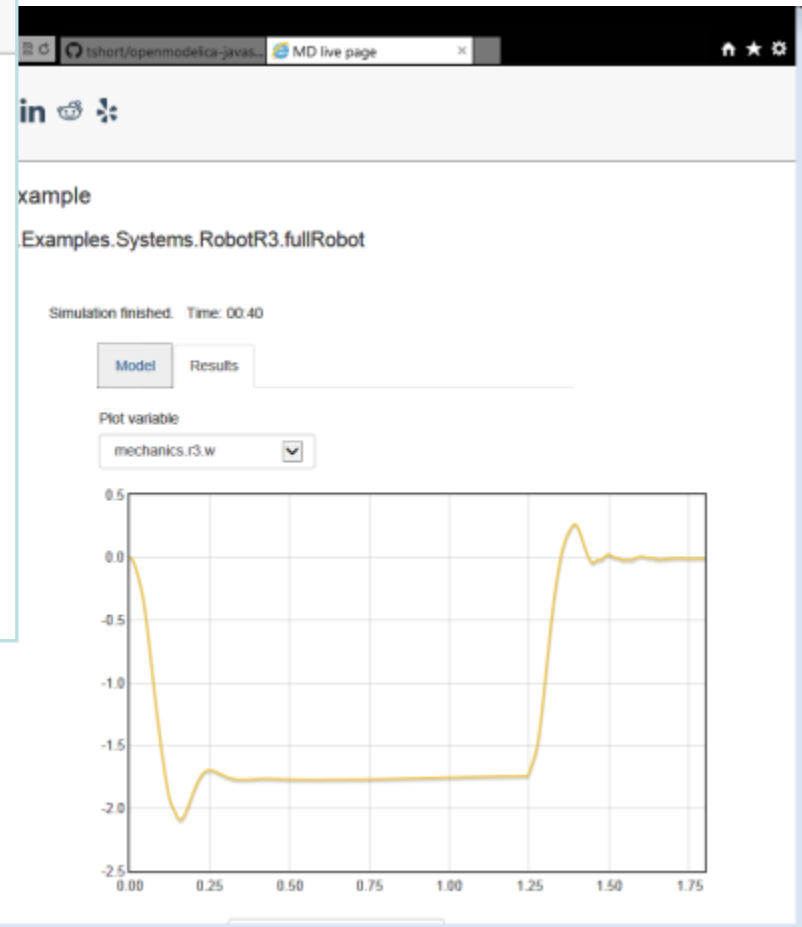
- Code Outline for easy navigation within Modelica files** (pointing to the Outline pane)
- Identifier Info on Hovering** (pointing to the tooltip)

At the bottom of the IDE, the status bar shows "64M of 254M" and "Ctrl Contrib (Bottom)".

OpenModelica Simulation in Web Browser Client



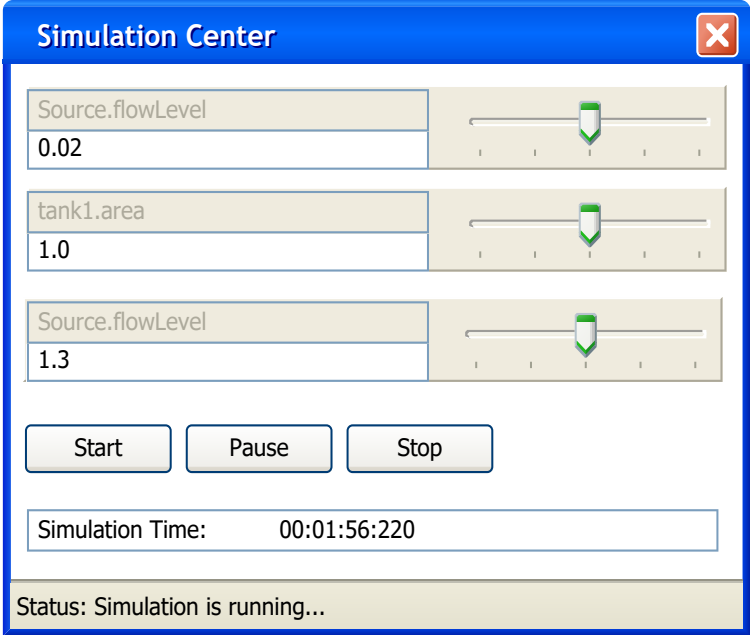
MultiBody RobotR3.FullRobot



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

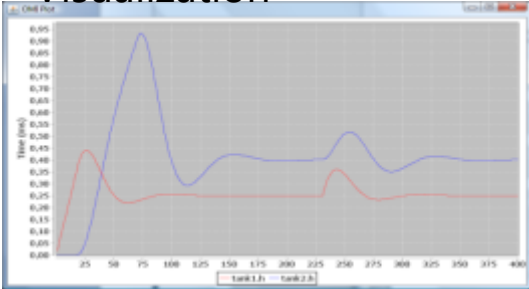
Interactive Simulation

Simulation Control

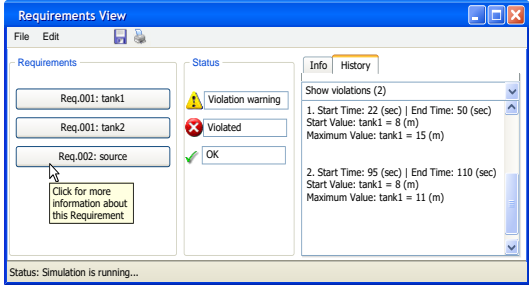


The Simulation Center window features three parameter sliders: Source.flowLevel (0.02), tank1.area (1.0), and Source.flowLevel (1.3). Below the sliders are Start, Pause, and Stop buttons. At the bottom, the Simulation Time is 00:01:56:220 and the status is 'Simulation is running...'.

Examples of Simulation Visualization

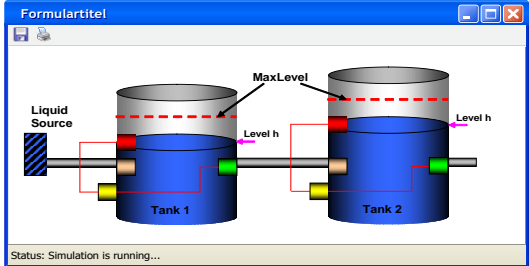


Plot View



The Requirements View window shows a list of requirements: Req.001: tank1 (Violation warning), Req.001: tank2 (Violated), and Req.002: source (OK). The Status section shows a 'Violation warning' icon. The Info/History section displays two violations with their start and end times and maximum values for tank1.

Requirements Evaluation View in ModelicaML



Domain-Specific Visualization View

OMPython – Python Scripting with OpenModelica

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

The screenshot shows a Python script named `test_execute_mode.py` and its execution output. The script uses the `OMPython` library to load a Modelica model, simulate it, and plot the results.

```
import OMPython

OMPython.execute("loadFile('c:/OpenModelica.8.1/testmodels/BouncingBall.mo')")
result=OMPython.execute("simulate(BouncingBall, stopTime=2, method='Euler')")
print result
OMPython.execute("plot(h)")

OMPython.execute("@quit()")
```

The execution output shows the OMC server starting and running the simulation. The output includes simulation options, simulation results, and a message that OMC has been shutdown.

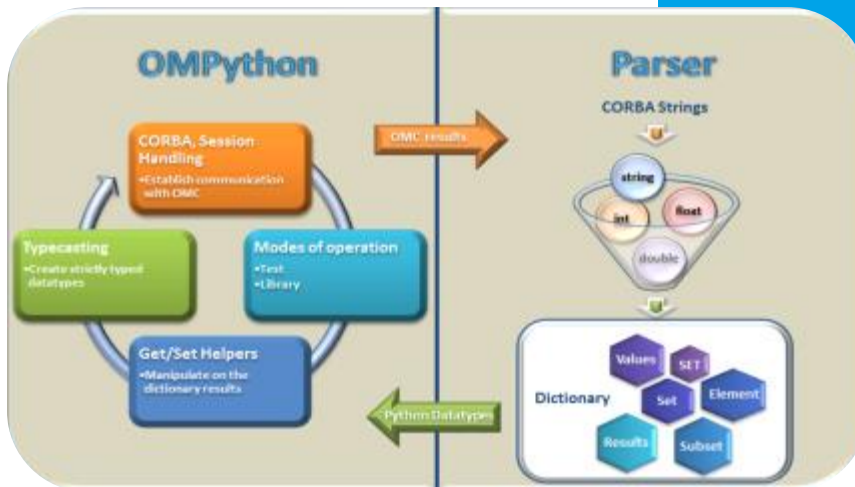
```
C:\Users\ganan642>python test_execute_mode.py
OMC Server is up and running at file:///c:/users/ganan642/appdata
\local\temp\openmodelica.objid.20120825120756188000

{'simulationoptions': {'options': {'storeInTemp': False, 'cf
lags': '...', 'simflags': '...', 'variableFilter': '...', 'noClean
': False, 'outputFormat': 'mat', 'method': 'dassl', 'measureT
ime': False, 'stopTime': 2.0, 'startTime': 0.0, 'numberOfInterv
als': 500, 'tolerance': 1e-06, 'filenamePrefix': 'BouncingBall'},
'simulationresults': {'timecompile': 6.89815662792063, 'timeBack
end': 0.0229111689831523, 'messages': '...', 'timeFrontend': 0.024
5992104508437, 'timeSimulation': 0.131418166559841, 'timeTemplat
e': 0.0206379911344139, 'timeSimCode': 0.00999736303670383, 'time
Total': 7.1078098383753, 'resultFile': 'c:/users/ganan642/bounci
ngBall_res.mat'}}

OMC has been shutdown

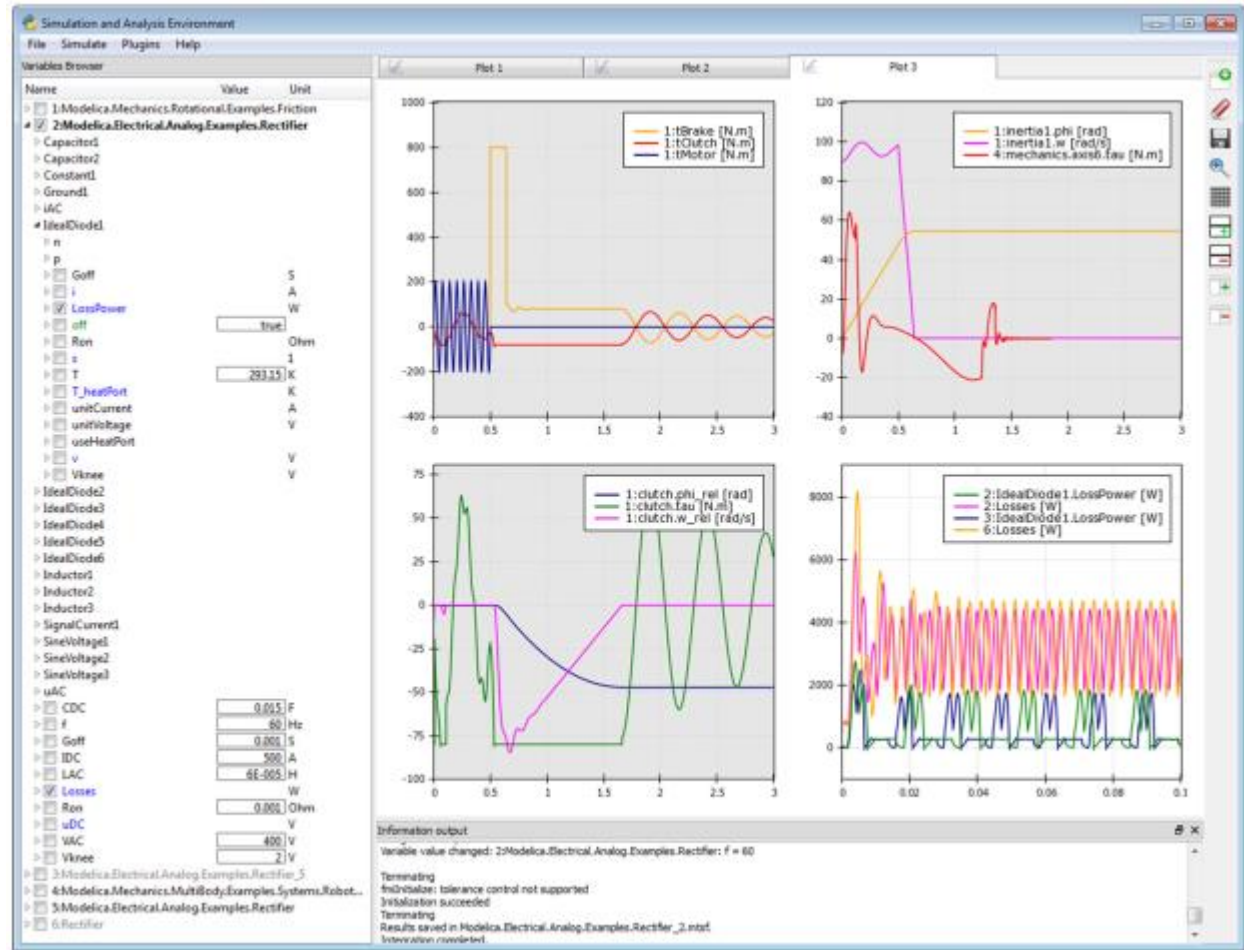
C:\users\ganan642>
```

The output also includes a plot titled "Plot by OpenModelica" showing the height of the ball over time.



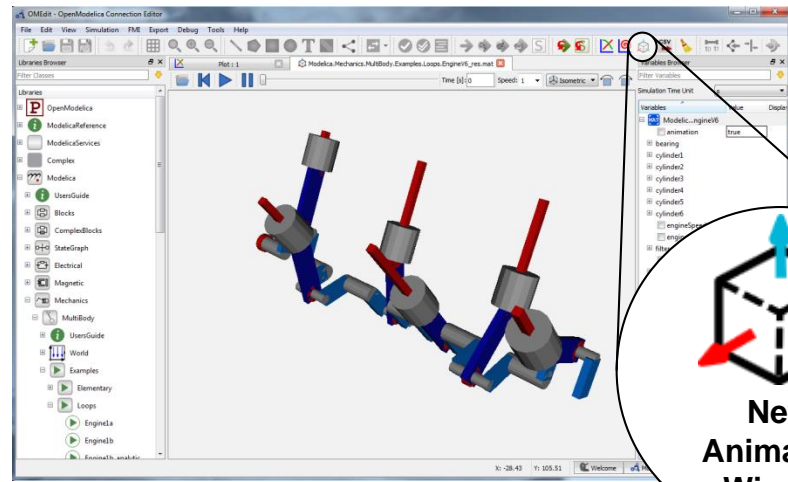
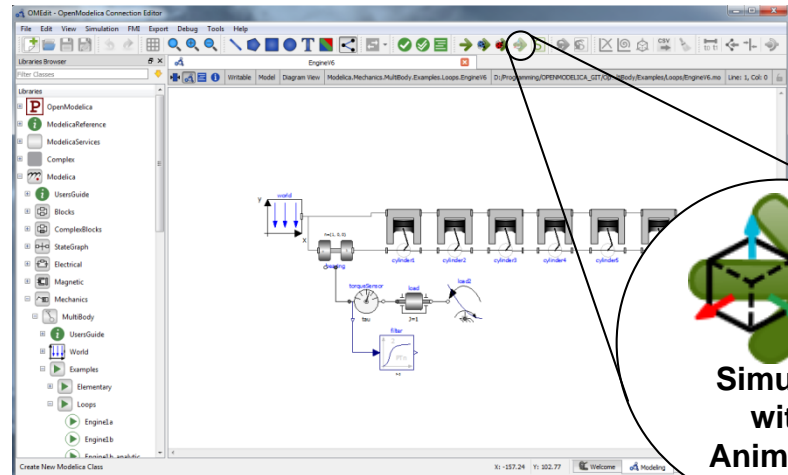
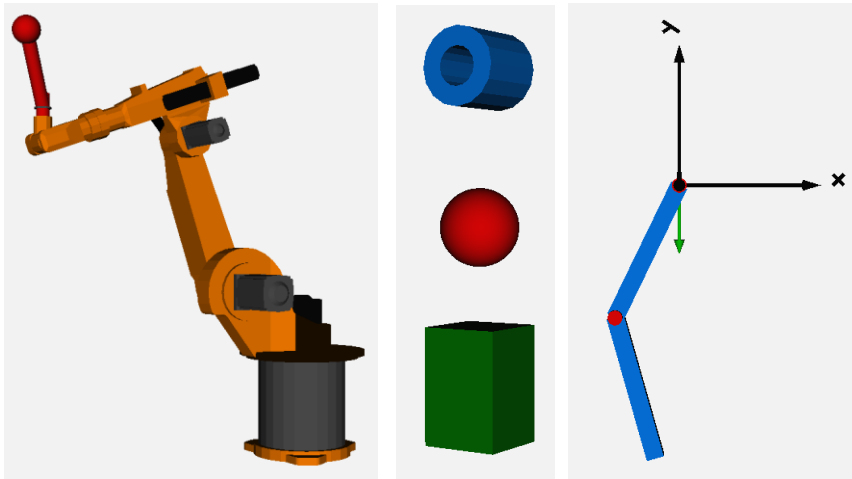
PySimulator Package

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



OMEdit 3D Visualization of Multi-Body Systems

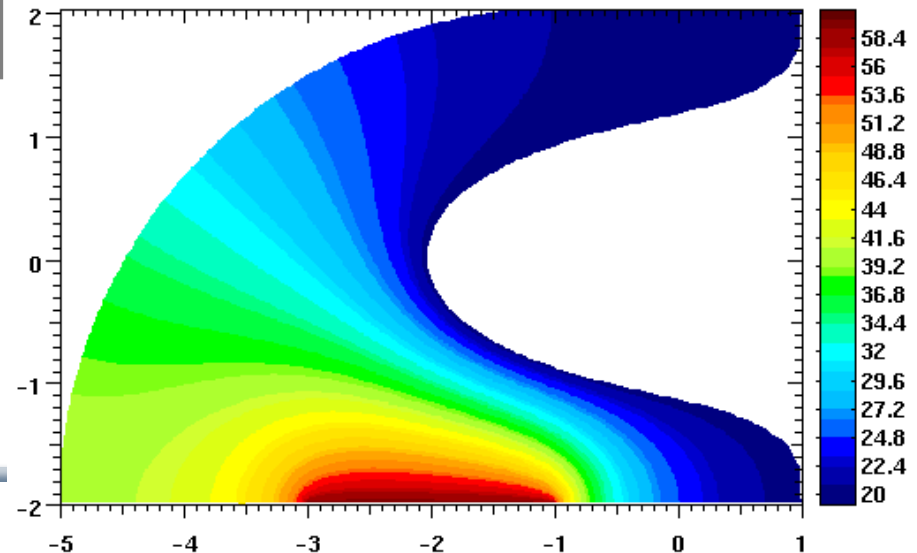
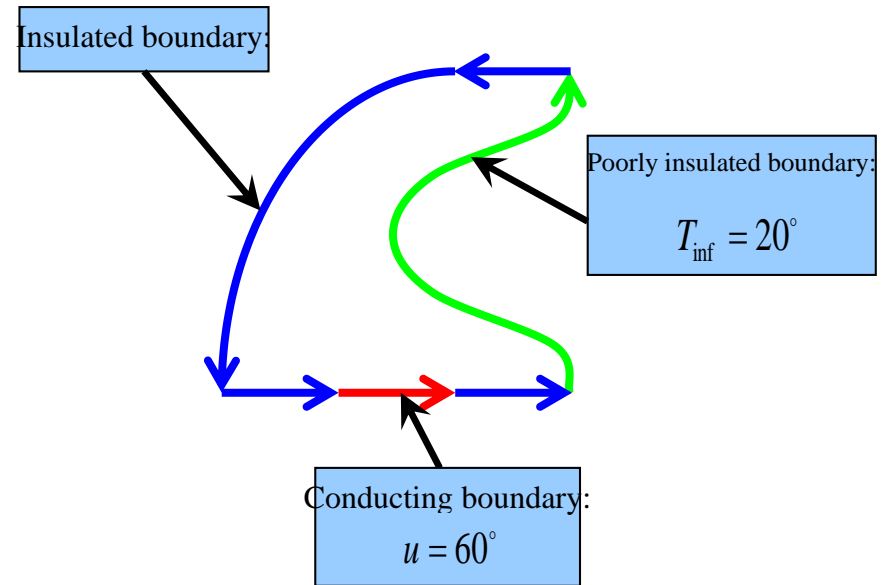
- Built-in feature of OMEdit to animate MSL-Multi-Body shapes
- Visualization of simulation results
- Animation of geometric primitives and CAD-Files



Extending Modelica with PDEs for 2D, 3D flow problems – Research

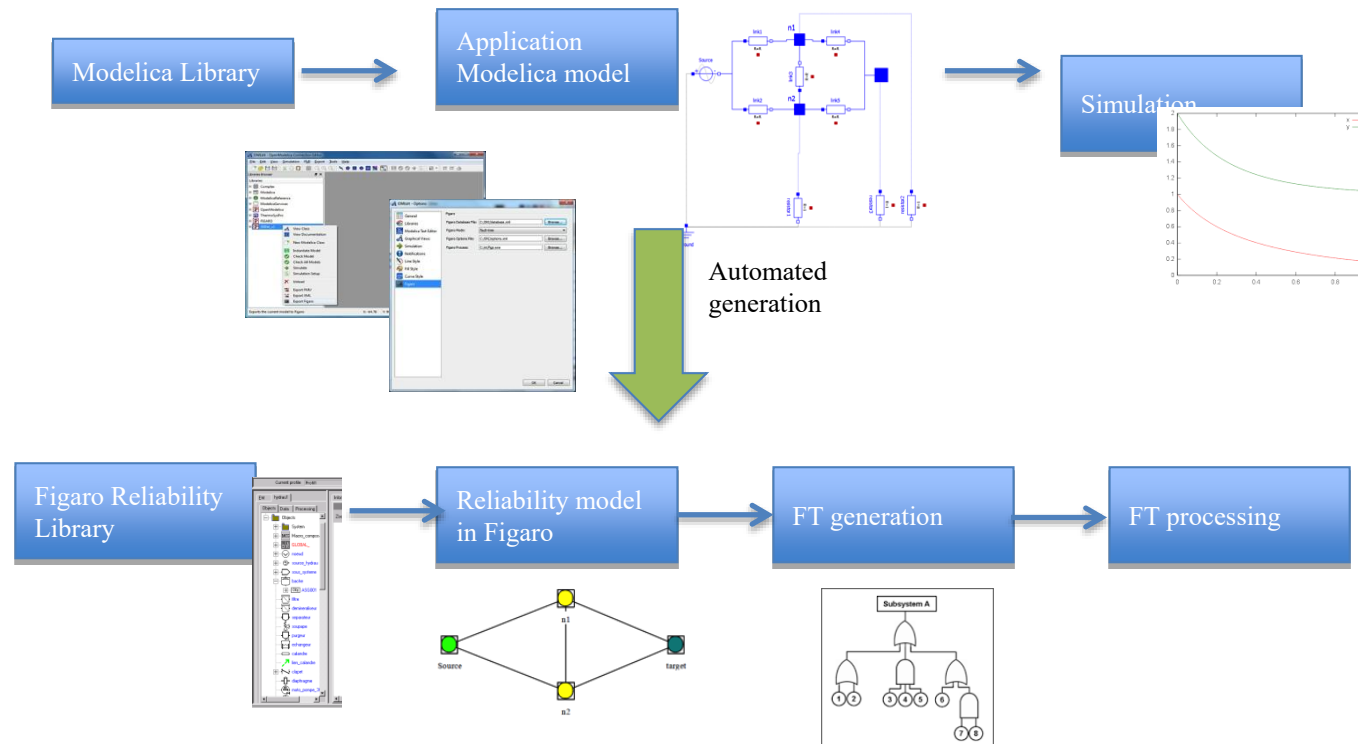
```
class PDEModel
  HeatNeumann h_iso;
  Dirichlet h_heated(g=50);
  HeatRobin h_glass(h_heat=30000);
  HeatTransfer ht;
  Rectangle2D dom;
equation
  dom.eq=ht;
  dom.left.bc=h_glass;
  dom.top.bc=h_iso;
  dom.right.bc=h_iso;
  dom.bottom.bc=h_heated;
end PDEModel;
```

Prototype in OpenModelica 2005
PhD Thesis by Levon Saldamli
www.openmodelica.org
Currently not operational



Failure Mode and Effects Analysis (FMEA) in OM

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



OMOptim – Optimization (1)

Model structure

Model Variables

Optimized parameters

Optimized Objectives

The screenshot displays the MinEIT software interface, which is used for optimization. The interface is divided into several sections:

- Model structure:** A list of model components on the left side, including variables like Pc, Va, Vb, Ia, Ib, Ic, Ea, Eb, Ec, and various investment and operational gain parameters.
- Model Variables:** A table showing the current values and descriptions of the model variables. The table is highlighted with a green border.
- Optimized parameters:** A table showing the optimized values for selected variables, highlighted with a blue border.
- Optimized Objectives:** A table showing the optimized values for the objective functions, highlighted with a yellow border.

Variables Table:

Name	Value	Description
global.sourceeaudville.h	1,18294e+06	[J/kg]
global.sourceeaudville.flowPort.p	100000	
global.sourceInEchColdB.h	1,41347e+06	[J/kg]
global.sourceInEchColdB.flowPort.p	100000	
global.sourceInEchColdB.debit	12,78	[kg/s]
global.sourceEffluentsECS.h	1,35495e+06	[J/kg]
global.sourceEffluentsECS.flowPort.p	100000	
global.sourceEffluentsECS.etat	1	
global.sourceEffluentsECS.debit1	0	
global.sourceEffluentsECS.debit	1	[kg/s]
global.sourceEffluentsB.h	1,35495e+06	[J/kg]
global.sourceEffluentsB.flowPort.p	100000	
global.sourceEffluentsB.etat	1	
global.sourceEffluentsB.debit	1,22612	[kg/s]
global.sourceEffluentsA.h	1,35495e+06	[J/kg]
global.sourceEffluentsA.flowPort.p	100000	
global.sourceEffluentsA.etat	1	
global.sourceEffluentsA.debit	0,601234	[kg/s]
global.scenariosourceEaudville.debit	0,940001	[kg/s]
global.scenariodepartB.z	0	

Optimized variables Table:

Name	Description	Opt Minimum
global.sourceEffluentsB.debit	[kg/s]	0
global.sourceEffluentsA.debit	[kg/s]	0
global.scenarioPACB.MySpecPcomp		0
global.scenarioPACA.MvSpecPcomp		0

Optimization objectives Table:

Name	Description	Direction	M
global.gaincoutoperationnel		Maximize	0
global.coutdinvestissement		Minimize	0

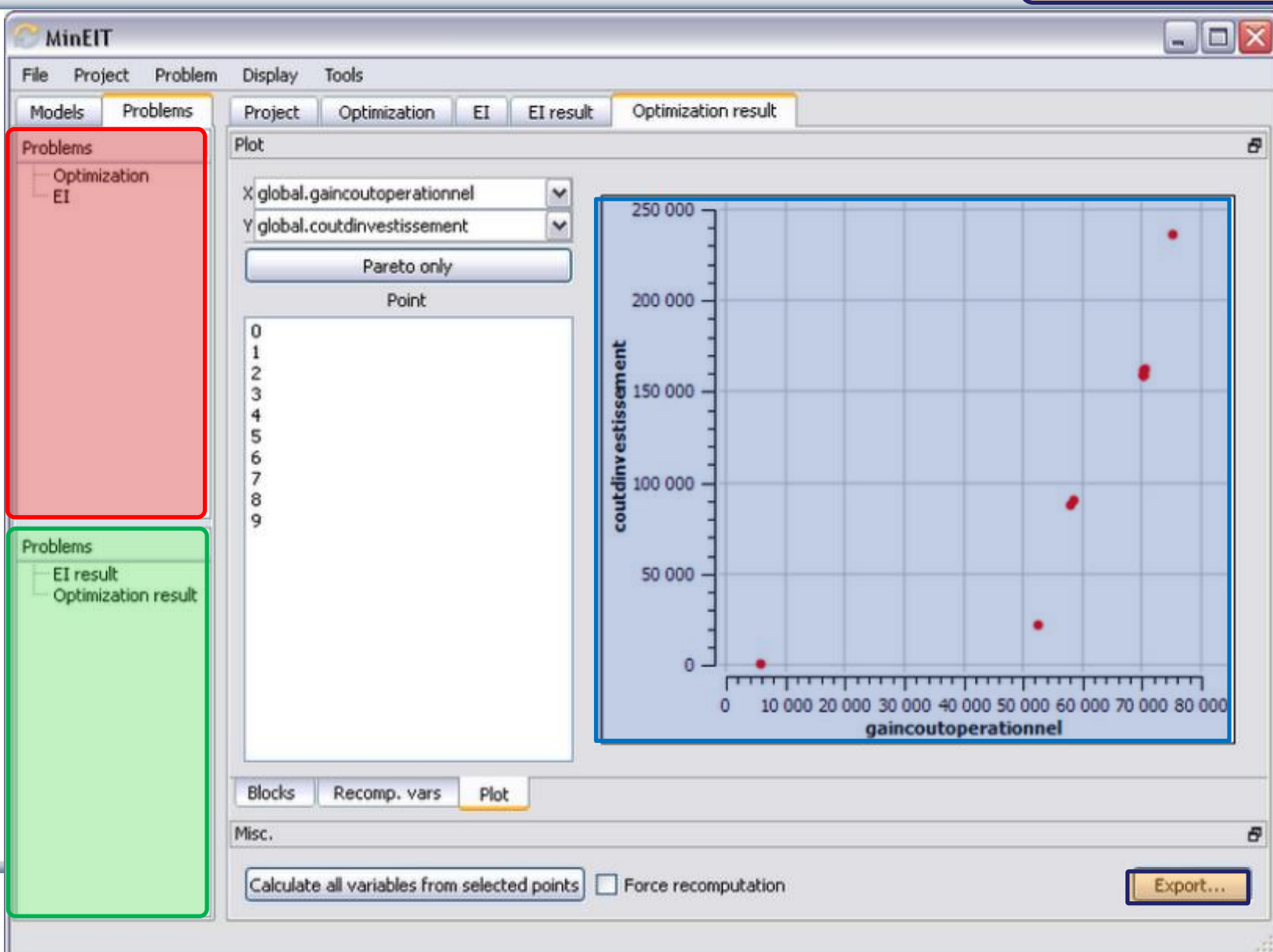
Problems

OMOptim – Optimization (2)

Solved problems

Result plot

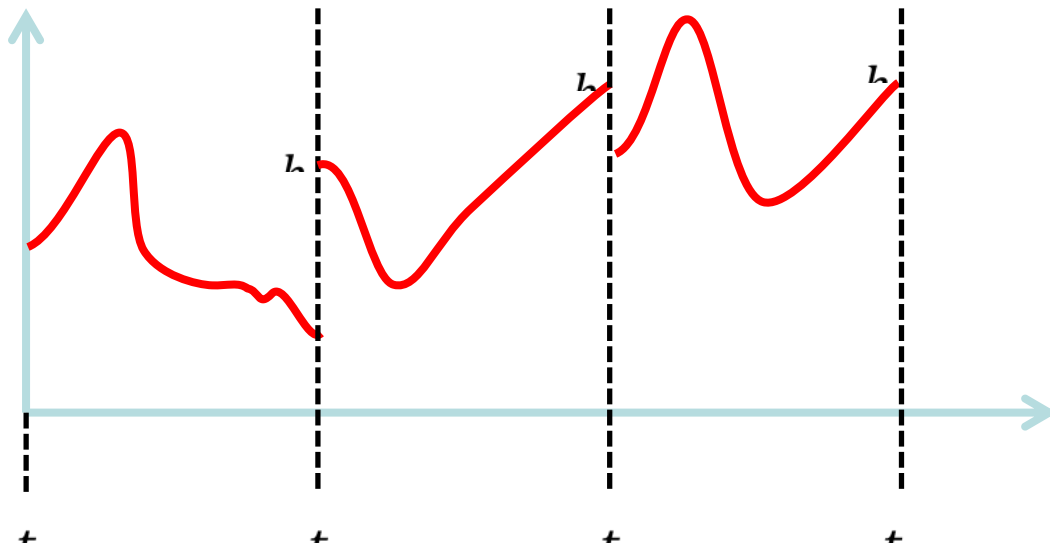
Export result data .csv



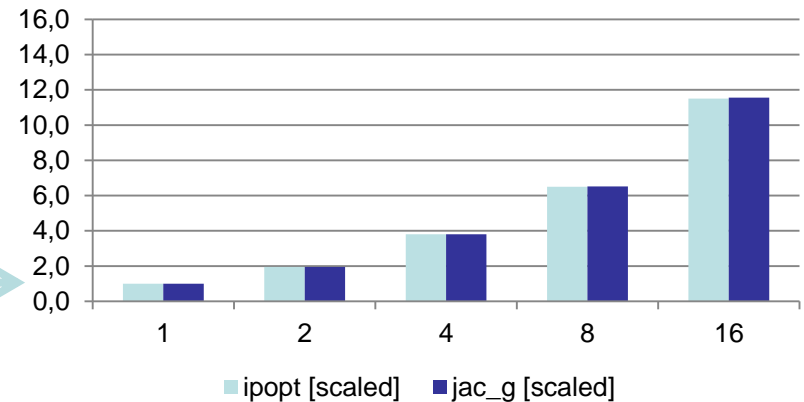
Multiple-Shooting and Collocation Dynamic Trajectory Optimization

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

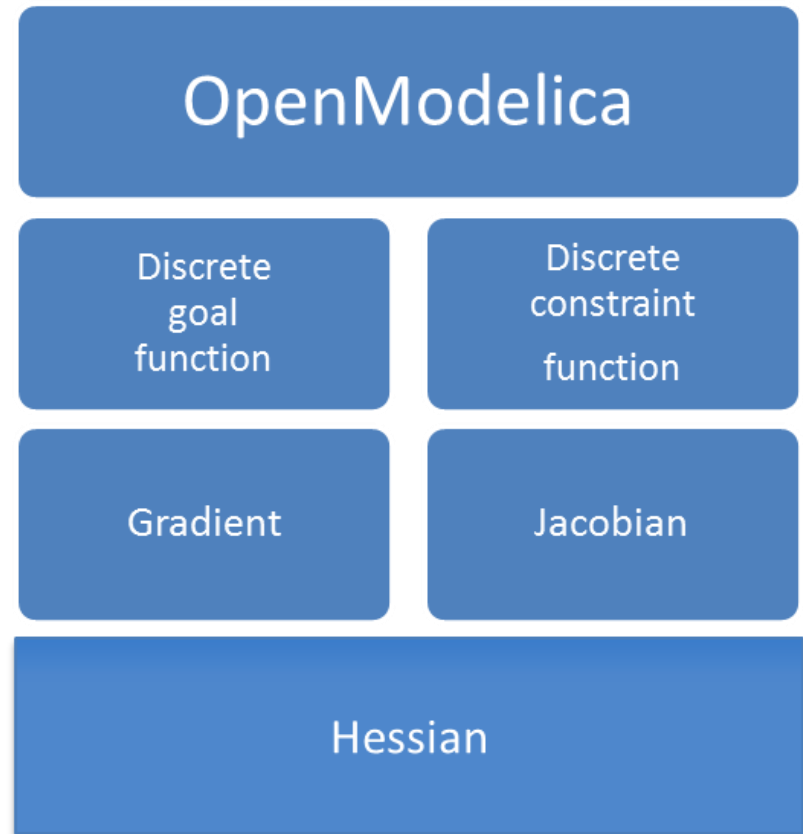
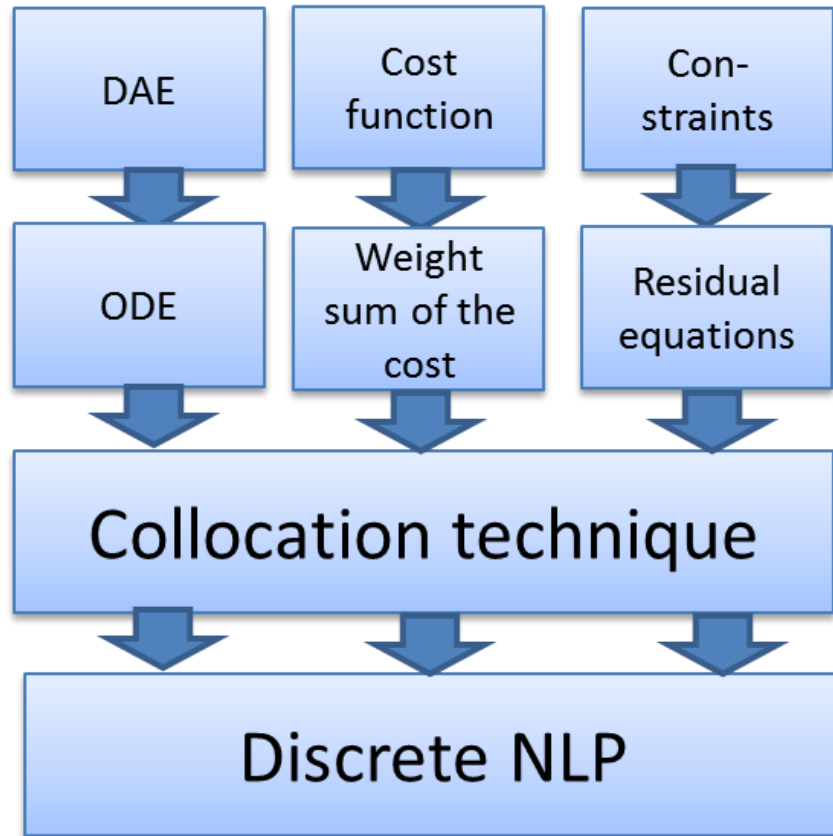
$$x_i(t_{i+1}) = h_i + \int_{t_i}^{t_{i+1}} f(x_i(t), u(t), t) dt \approx F(t_i, t_{i+1}, h_i, u_i), \quad x_i(t_i) = h_i$$



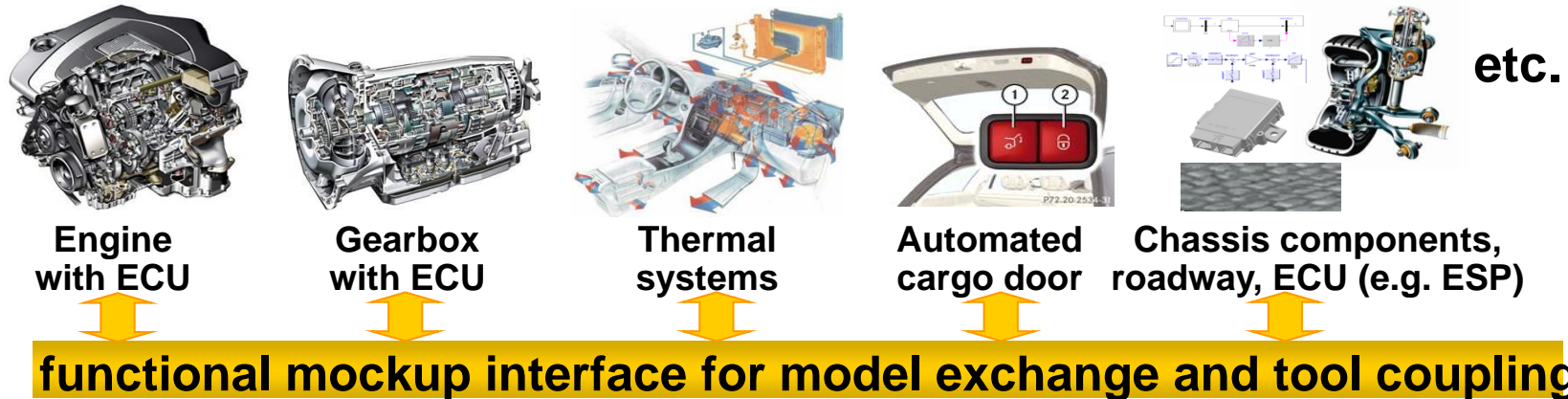
Example speedup, 16 cores:
MULTIPLE_COLLOCATION



OpenModelica Dynamic Optimization Collocation



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

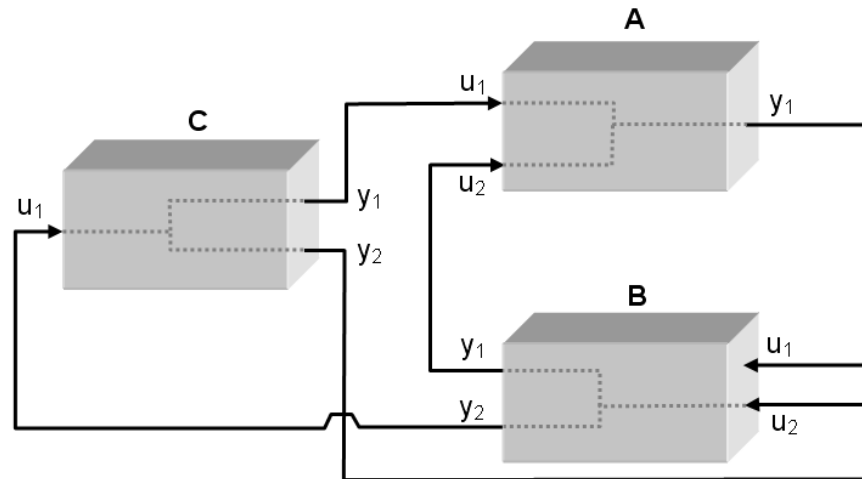


courtesy Daimler

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

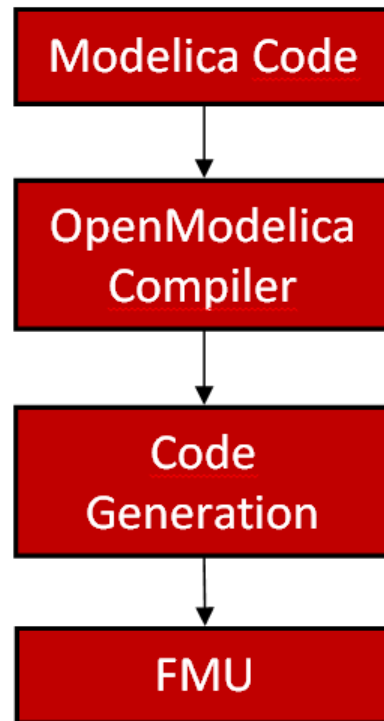
Functional Mockup Units

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



OpenModelica Functional Mockup Interface (FMI)

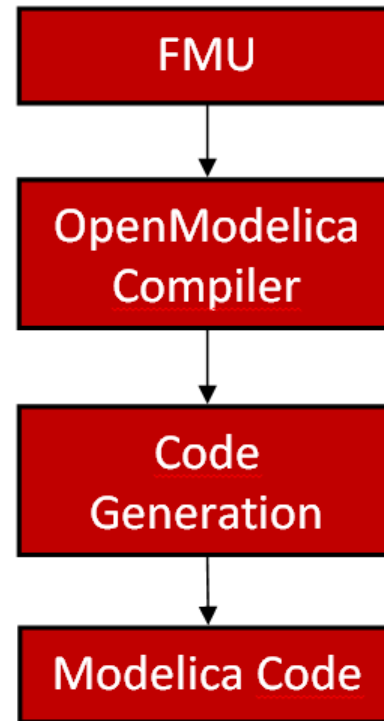
FMI Export



Translator, Analyzer & Optimizer

Model Description, DLL & FMI interface functions

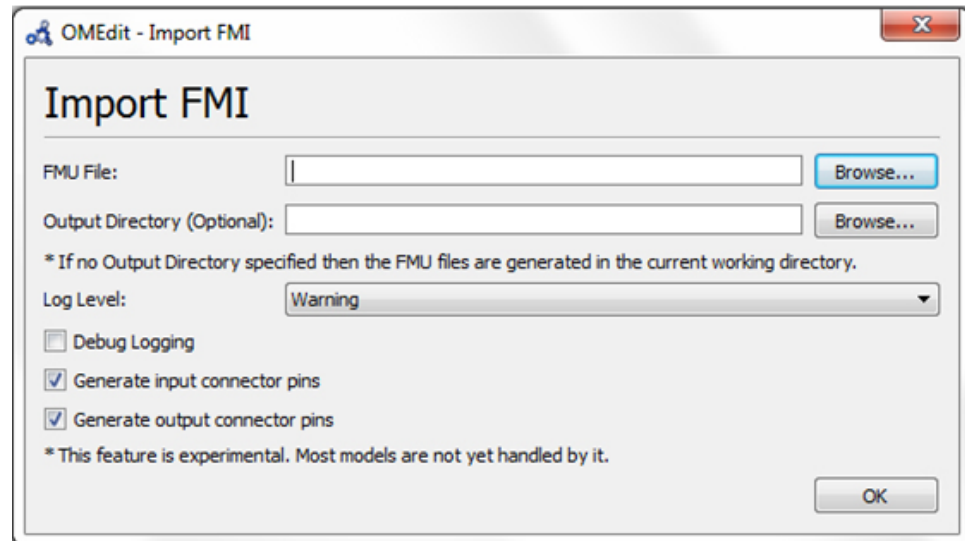
FMI Import



FMU parsing, reading states & events

FMI in OpenModelica

- Model Exchange implemented (FMI 1.0 and FMI 2.0)
- FMI 2.0 Co-simulation available
- The FMI interface is accessible via the **OpenModelica scripting environment** and the **OpenModelica connection editor**



OpenModelica Code Generators for Embedded Real-time Code

- A **full-fledged** OpenModelica-generated source-code FMU (Functional Mockup Unit) code generator
 - Can be used to **cross-compile FMUs** for platforms with more available memory.
 - These platforms can **map** FMI inputs/outputs to analog/digital I/O in the importing FMI master.
- A very **simple code generator** generating a **small footprint** statically linked executable.
 - Not an FMU because there is no OS, filesystem, or shared objects in microcontrollers.

Code Generator Comparison, Full vs Simple

	Full Source-code FMU targeting 8-bit AVR proc	Simple code generator targeting 8-bit AVR proc
Hello World (0 equations)	43 kB flash memory 23 kB variables (RAM)	130 B flash memory 0 B variables (RAM)
SBHS Board (real-time PID controller, LCD, etc)	68 kB flash memory 25 kB variables (RAM)	4090 B flash memory 151 B variables (RAM)

The largest 8-bit AVR processor MCUs (Micro Controller Units) have 16 kB SRAM.

One of the more (ATmega328p; Arduino Uno) has 2 kB SRAM.

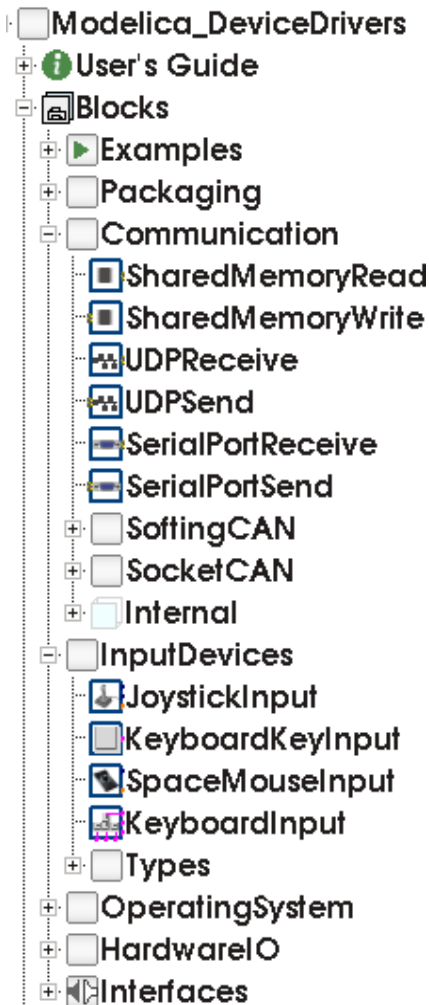
The ATmega16 we target has **1 kB SRAM available** (stack, heap, and global variables)

The Simple Code Generator

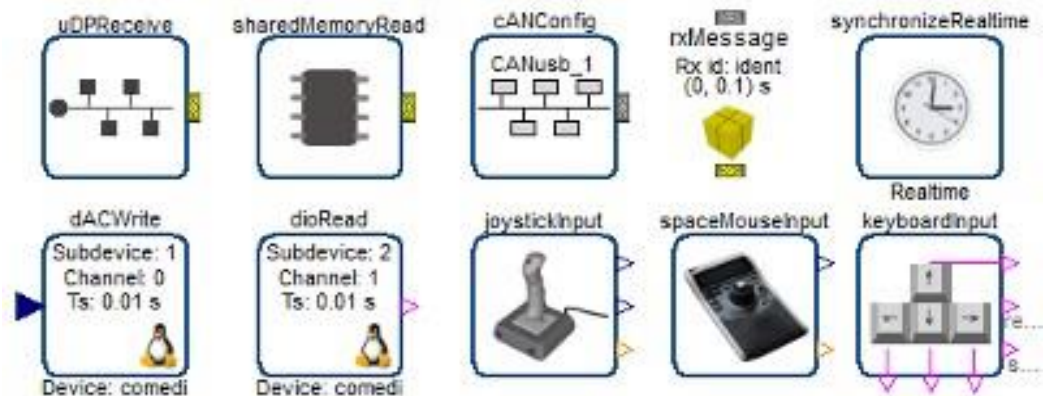
Supports only a limited Modelica subset

- No initialization (yet)
- No strongly connected components
- No events
- No functions (except external C and built-in)
- Only parts that OpenModelica can generate good and efficient code for right now (extensions might need changes in the intermediate code)
 - Unused variables are not accepted (OM usually duplicates all variables for pre() operators, non-linear system guesses, etc... but only a few of them are actually used)
- FMU-like interface (but statically linked)

Communication & I/O Devices: MODELICA_DEVICEDRIVERS Library



- **Free library** for interfacing hardware drivers
- **Cross-platform** (Windows and Linux)
- UDP, SharedMemory, CAN, Keyboard, Joystick/Gamepad
- DAQ cards for digital and analog IO (only Linux)
- Developed for **interactive real-time** simulations



OpenModelica and Device Drivers Library

AVR Processor Support

- No direct Atmel AVR or Arduino support in the OpenModelica compiler
- **Everything is done by the Modelica DeviceDrivers library**
- **All I/O is modeled explicitly in Modelica**, which makes code generation very simple

Modelica Device Drivers Library - AVR processor sub-packages:

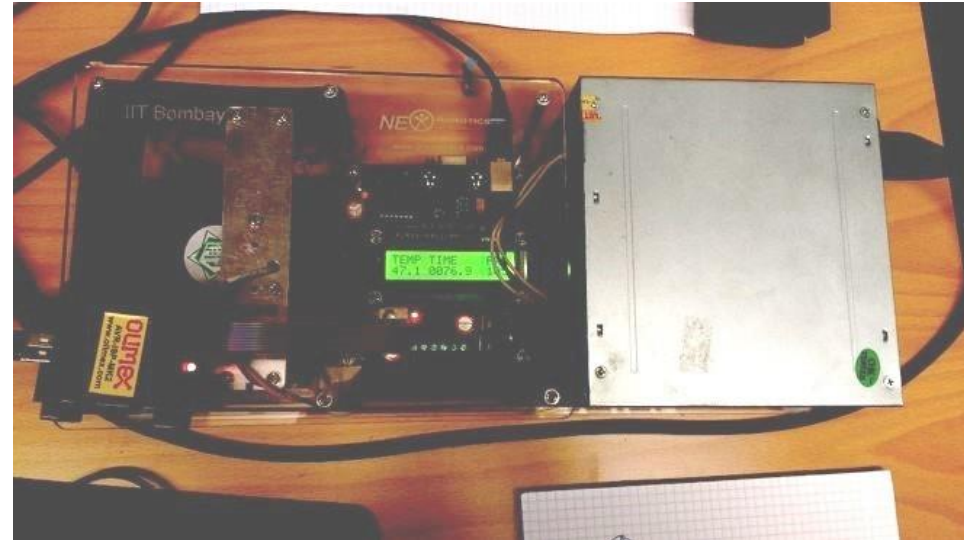
- IO.AVR.Analog (ADC – Analog Input)
- IO.AVR.PWM (PWM output)
- IO.AVR.Digital.LCD (HD44780 LCD driver on a single 8-pin digital port)
- OS.AVR.Timers (Hardware timer setup, used by real-time and PWM packages)
- OS.AVR.RealTime (very simple real-time synchronization; one interrupt per clock cycle; works for single-step solvers)

Use Case: SBHS (Single Board Heating System)

Single board heating system (IIT Bombay)

- Use for teaching basic control theory
- Usually controlled by serial port (set fan value, read temperature, etc)
- OpenModelica can generate code targeting the ATmega16 on the board (AVR-ISP programmer in the lower left).

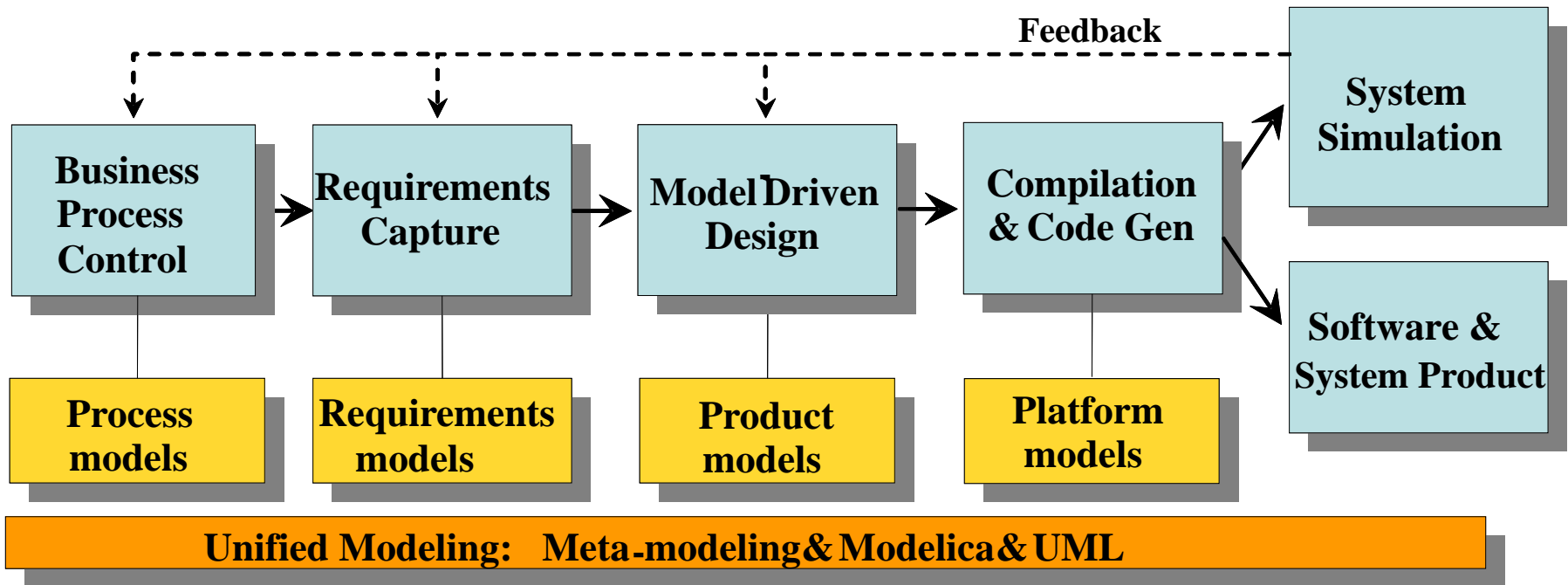
Program size is 4090 bytes including LCD driver and PID-controller (out of 16 kB flash memory available).



Movie Demo!

OPENPROD – Large 28-partner European Project, 2009-2012

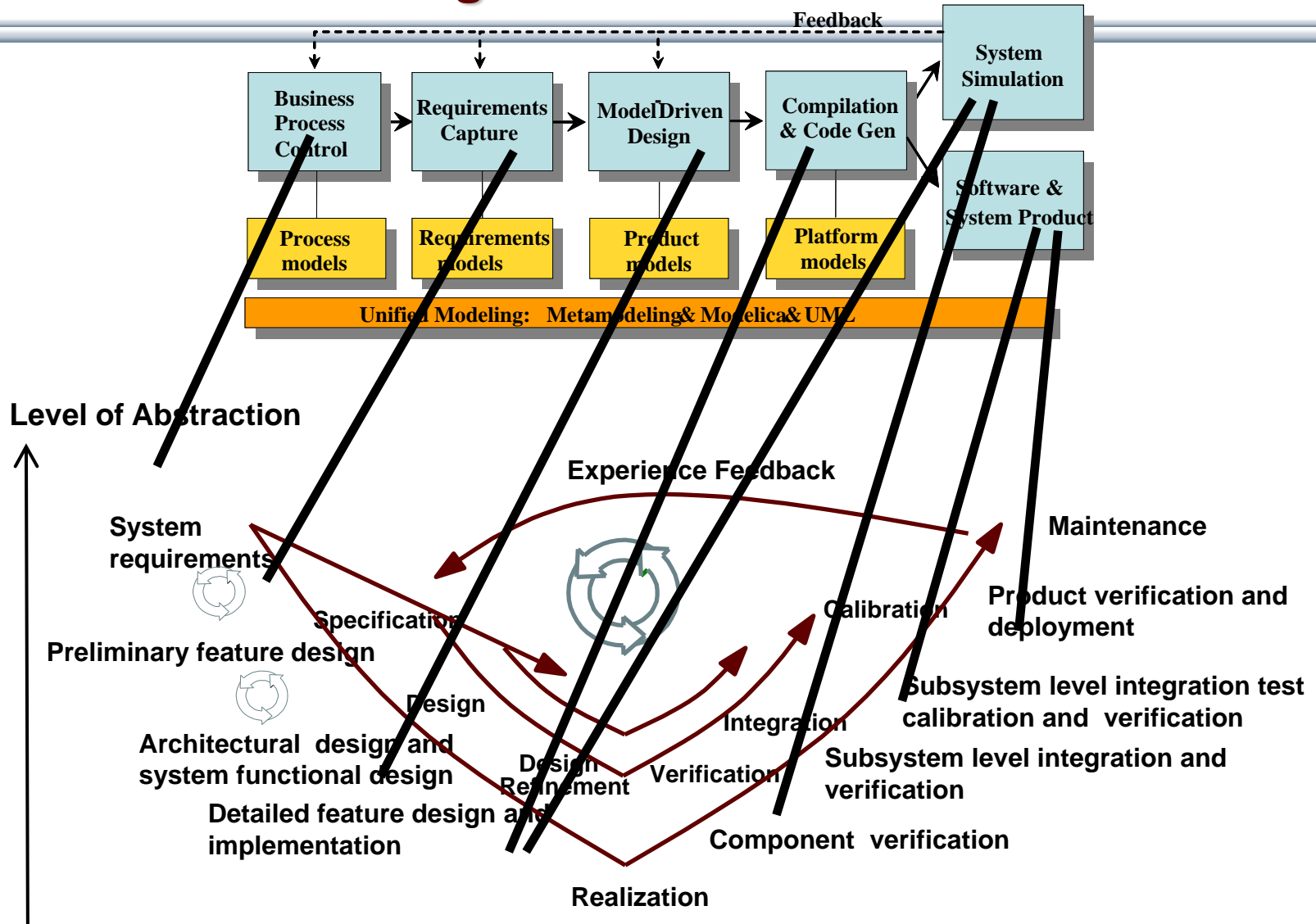
Vision of Cyber-Physical Model-Based Product Development



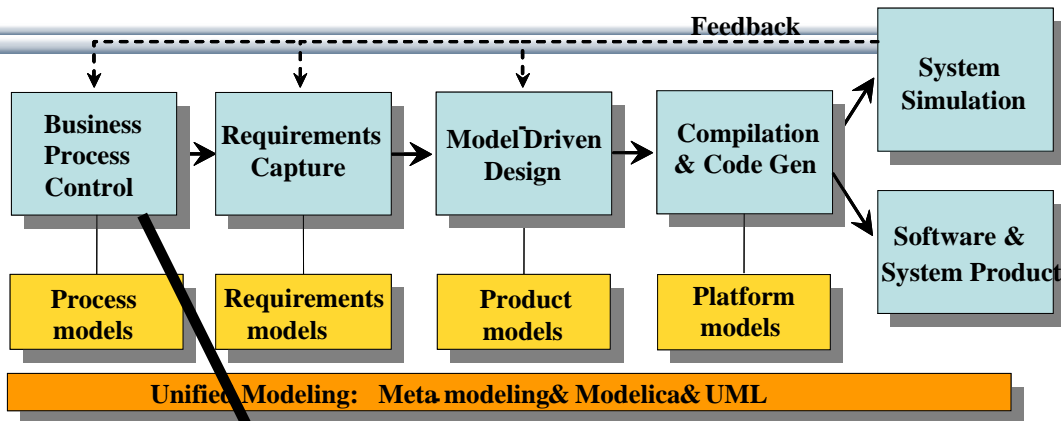
OPENPROD Vision of unified modeling framework for model-based product development.

Open Standards – Modelica (HW, SW) and UML (SW)

OPENPROD Model-Based Development Environment Covers Product-Design V



Business Process Control and Modeling



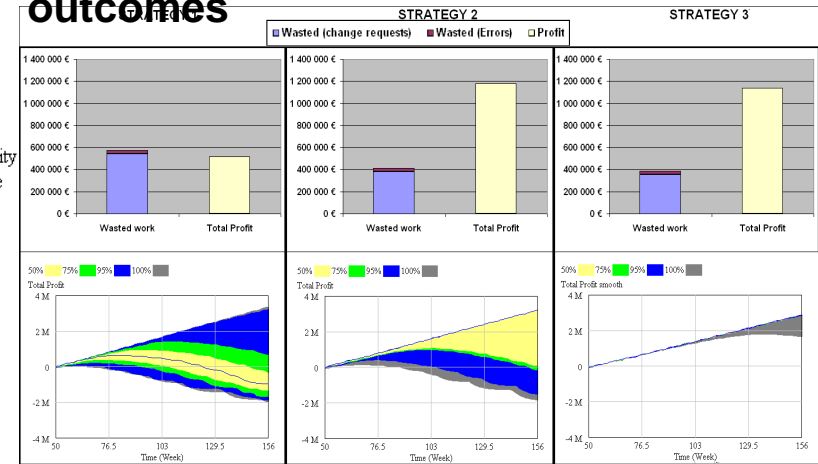
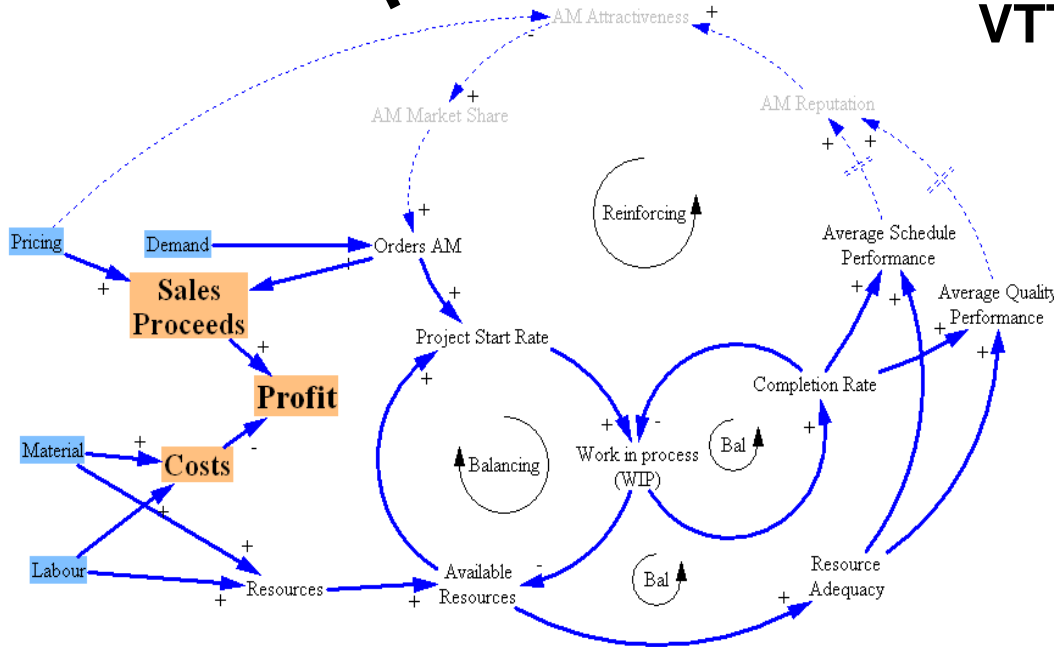
VTT Simantics
Business process modeler

OpenModelica
compiler & simulator

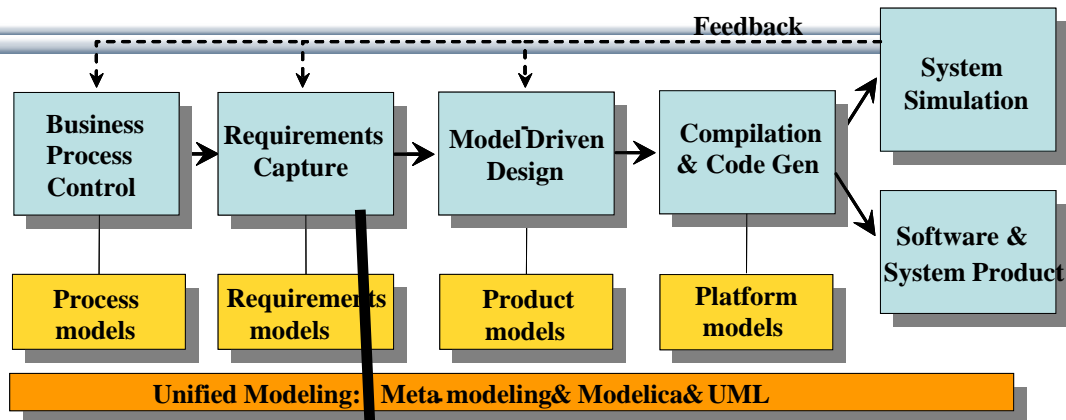
OpenModelica based simulation

Metso Business model & simulation VTT Simantics Graphic Modeling To

Simulation of 3 strategies with outcomes



Requirement Capture



OpenModelica based simulation

vVDR (virtual Verification of Designs against Requirements)

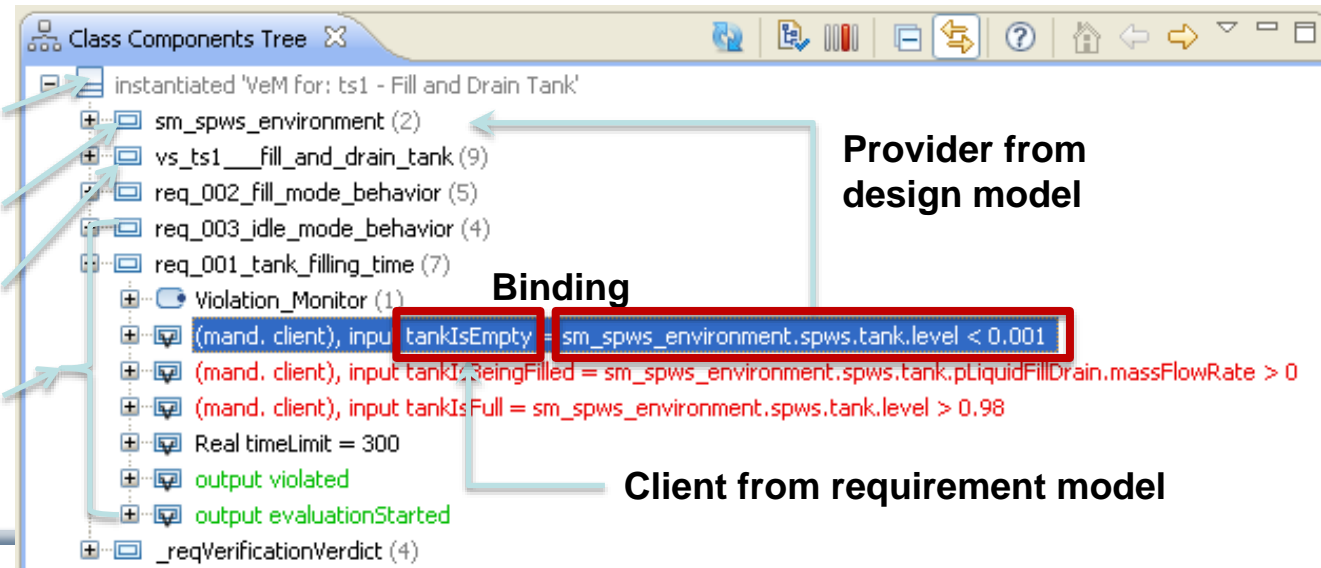
in ModelicaML UML/Modelica Profile, part of OpenModelica

Verification Model

Design Model

Scenario Model

Requirement Models

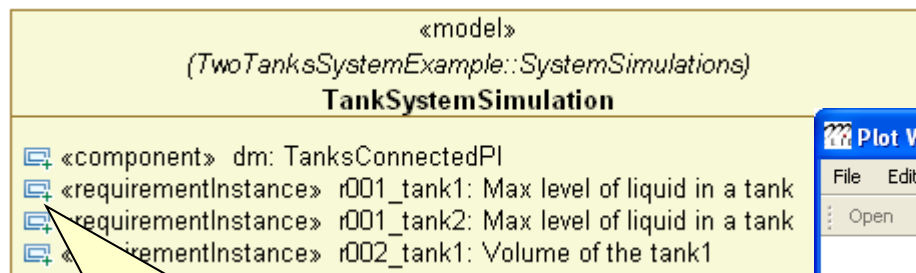


OpenModelica – ModelicaML UML Profile

SysML/UML to Modelica OMG Standardization

- ModelicaML is a UML Profile for SW/HW modeling
 - Applicable to “pure” UML or to other UML profiles, e.g. SysML
- Standardized Mapping UML/SysML to Modelica
 - Defines transformation/mapping for **executable** models
 - Being **standardized** by OMG
- ModelicaML
 - Defines graphical concrete syntax (graphical notation for diagram) for representing Modelica constructs integrated with UML
 - Includes graphical formalisms (e.g. State Machines, Activities, Requirements)
 - Which do not exist in Modelica language
 - Which are translated into executable Modelica code
 - Is defined towards generation of executable Modelica code
 - Current implementation based on the Papyrus UML tool + OpenModelica

Example: Simulation and Requirements Evaluation

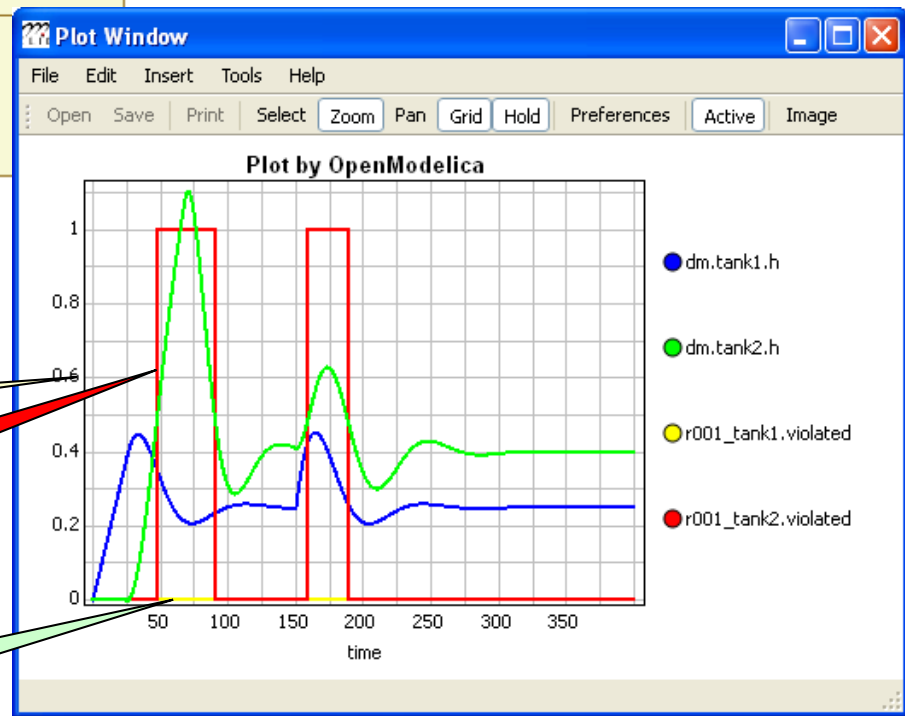


**Req. 001 is instantiated 2 times
(there are 2 tanks in the system)**

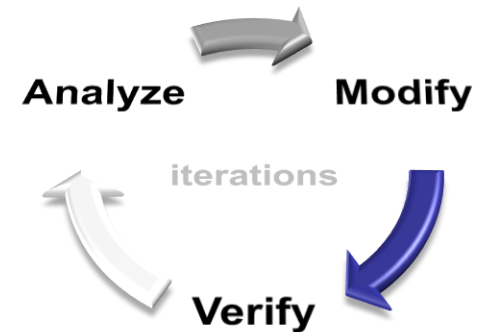
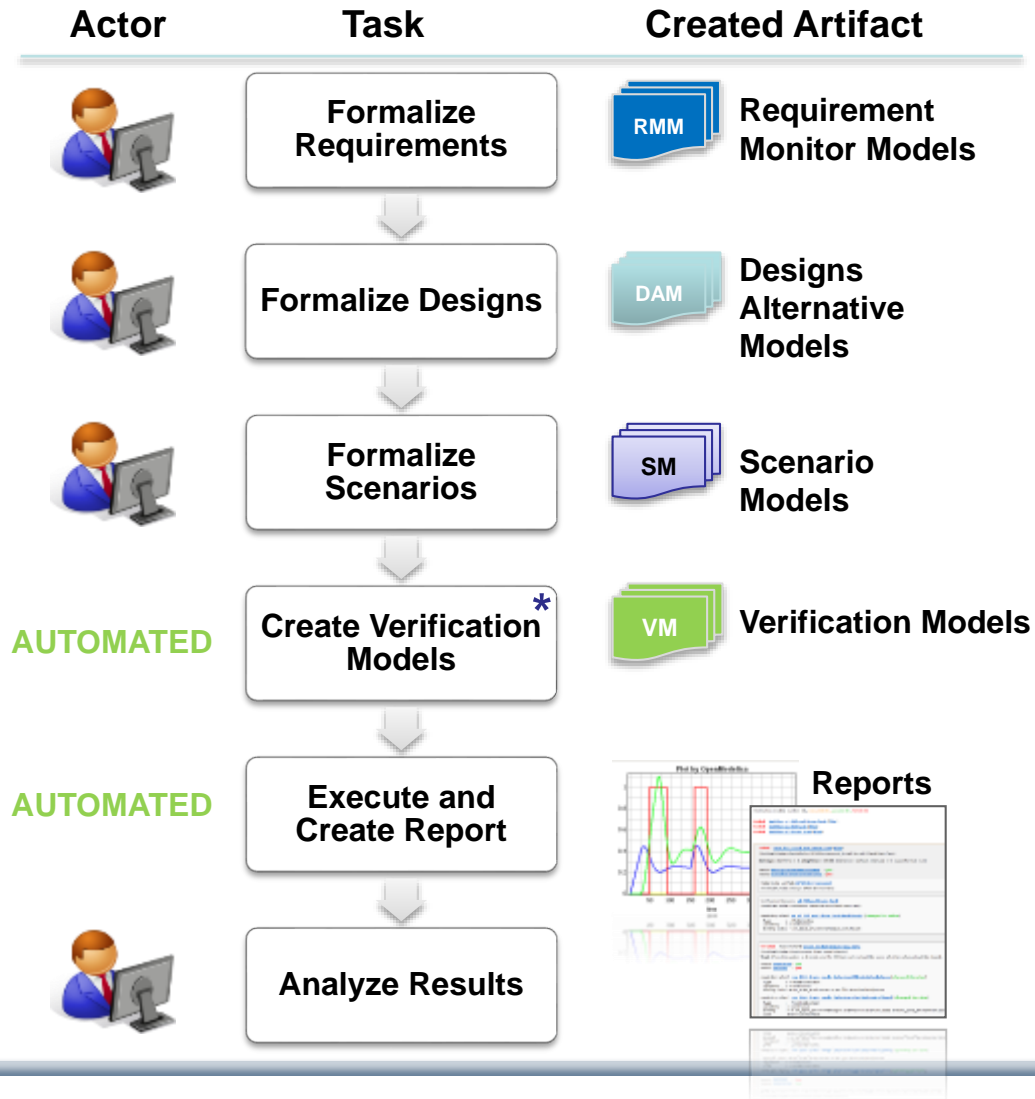
tank-height is 0.6m

**Req. 001 for the tank2 is
violated**

**Req. 001 for the tank1 is
not violated**



vVDR Method – virtual Verification of Designs vs Requirements



Goal: Enable on-demand verification of designs against requirements using automated model composition at any time during development.

ABB Industry Use of OpenModelica FMI 2.0 and Debugger

- ABB OPTIMAX® provides advanced model based control products for power generation and water utilities



- ABB: “ABB uses several compatible Modelica tools, including OpenModelica, depending on specific application needs.”
- ABB: “OpenModelica provides outstanding debugging features that help to save a lot of time during model development.”

Recent Large-scale ABB OpenModelica Application

Generate code for controlling 7.5 to 10% of German Power Production



ABB OPTIMAX PowerFit

- Real-time optimizing control of large-scale virtual power plant for system integration
- **Software including OpenModelica** now used in managing more than 2500 renewable plants, total up to 1.5 GW

High scalability supporting growth

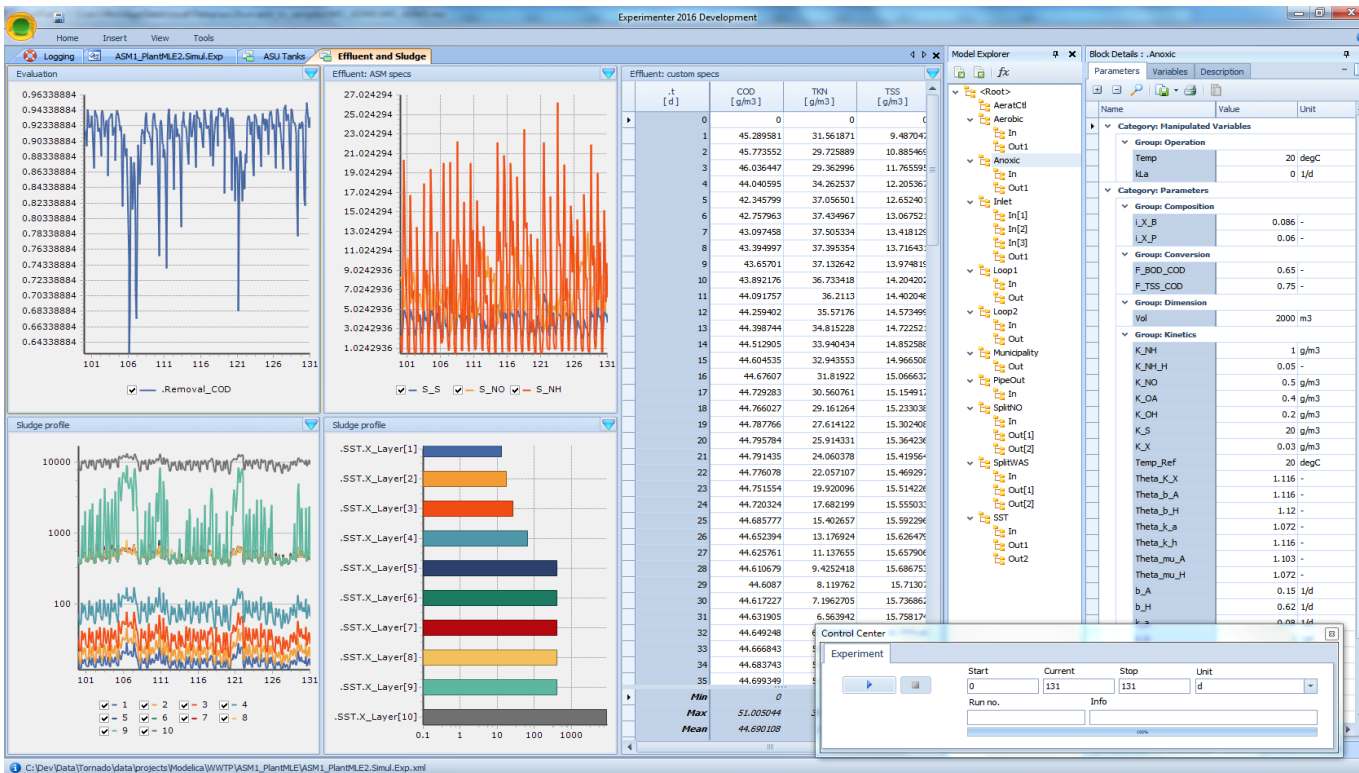
- 2012: initial delivery (for 50 plants)
- 2013: SW extension (500 plants)
- 2014: HW+SW extension (> 2000)
- 2015: HW+SW extension, incl. OpenModelica generating optimizing controller code in FMI 2.0 form

Manage 7.5% - 10% of German Power

- 2015, Aug: OpenModelica Exports FMUs for real-time optimizing control (seconds) of about **5.000 MW (7.5%) of power in Germany**

Industrial Product with OEM Usage of OpenModelica – MIKE by DHI, WEST Water Quality

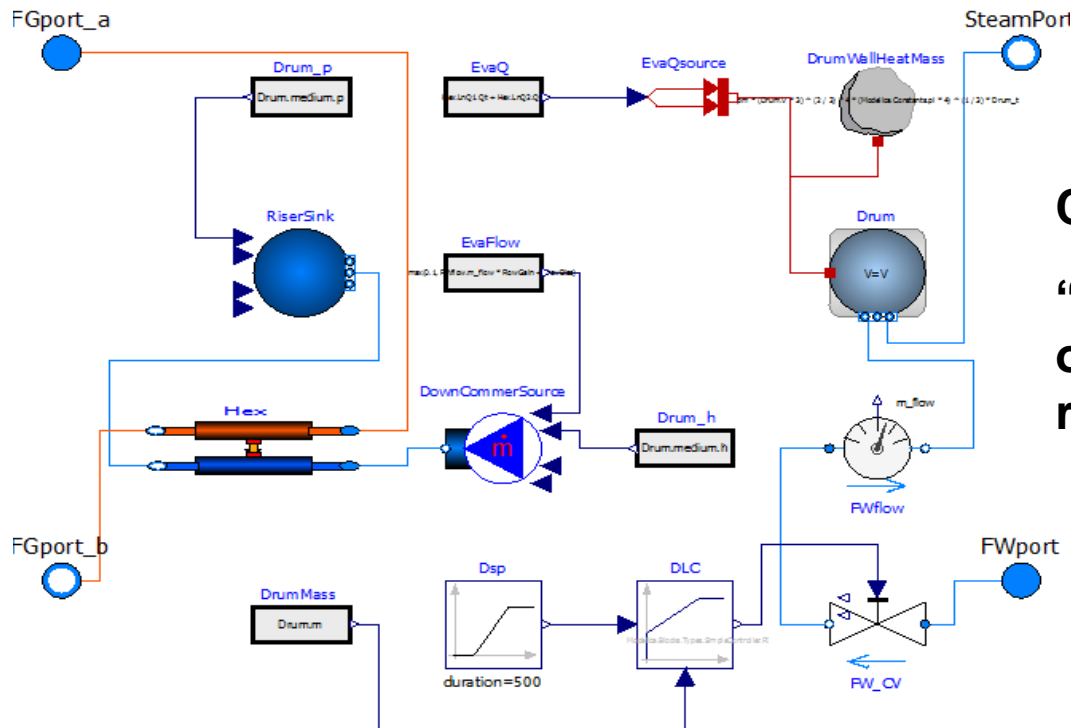
- MIKE by DHI, www.mikebydhi.com, WEST Water Quality modeling and simulation environment
- Includes a large part of the OpenModelica compiler using the OEM license.
- Here a water treatment effluent and sludge simulation.



Performance Profiling for faster Simulation

(Here: Profiling equations of Siemens Drum boiler model with evaporator)

- Measuring **performance** of equation blocks to find bottlenecks
 - Useful as input before model simplification for real-time applications
- Integrated with the debugger to **point out the slow equations**
- Suitable for **real-time profiling** (collect less information), or a complete view of all equation blocks and function calls



Conclusion from the evaluation:

“...the profiler makes the process of performance optimization radically shorter.”

OpenModelica MDT Algorithmic Code Debugger

The screenshot displays the Eclipse IDE interface for debugging a Modelica simulation. The main window is titled "Debug - HelloWorld/SimulationModel.mo - Eclipse SDK".

List of Stack Frames: A red box highlights the "Debug" view, which shows the current execution context. It includes the "Simulation Model [Modelica Development Tooling (MDT) GDB]" and the "Main Thread (stepping)". The stack frames are:

- getValueMultipliedByTwo at simulationmodel.mo:13
- eqFunction_3 at simulationmodel.mo:5

Variables View: A red box highlights the "Variables" view, which displays the current state of variables. It shows a table with the following data:

Name	Declared Type	Value	Actual Type
inValue	Real	1	double
outValue	Real	6.9453280720608359e-308	double

Code View: The "SimulationModel.mo" editor shows the source code. The function `getValueMultipliedByTwo` is highlighted, with the line `outValue := inValue * 2;` selected. The code is as follows:

```
model SimulationModel
  Real x(start = 1);
  Real y(start = 1);
  algorithm
    x := getValueMultipliedByTwo(x);
    y := x;
end SimulationModel;

function getValueMultipliedByTwo
  input Real inValue;
  output Real outValue;
  algorithm
    outValue := inValue * 2;
end getValueMultipliedByTwo;
```

Output View: A red box highlights the "Console" view, which is currently empty, indicating that no output has been captured yet.

Outline View: The "Outline" view shows the project structure, including the `getValueMultipliedByTwo` function and the `SimulationModel` model, with sub-elements `x` and `y`.

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

Debug - trunk/testsuite/mosfiles-nosim/QuotedFunction.mo - Eclipse SDK

File Edit Navigate Search Run Project Window Help

Correct Indentation Build project

Debug MDT

MDT GDB [Modelica Development Tooling (MDT) GDB]

MDT

Main Thread (stepping)

- オープンモデリッカー・ロックス at quotedfunction.mo:5
- Ceval_cevalCallFunction at Ceval.mo:1294
- Ceval_ceval at Ceval.mo:318
- Interactive_evaluateExpr at Interactive.mo:935
- Interactive_evaluateExprToStr at Interactive.mo:985
- Interactive_evaluate2 at Interactive.mo:507
- Interactive_evaluateToStdOut at Interactive.mo:329
- Interactive_evaluateToStdOut at Interactive.mo:333
- Interactive_evaluateToStdOut at Interactive.mo:333

Variables Breakpoints

Name	Declared Type	Value
◆ キャン・ザー・デバガー・シー・ミー	Real	1.5
◆ イエス・イット・キャン	Real	-4.836697827222

moGenerator.c Main.mo Util.mo System_omc.cpp systemimpl.c QuotedFunction.mo 31

```
function 'オープンモデリッカー・ロックス'  
  input Real 'キャン・ザー・デバガー・シー・ミー';  
  output Real 'イエス・イット・キャン';  
algorithm  
  'イエス・イット・キャン' := sin('キャン・ザー・デバガー・シー・ミー');  
end 'オープンモデリッカー・ロックス';
```

Console Tasks Problems Executables

MDT GDB [Modelica Development Tooling (MDT) GDB] C:\OpenModelica\trunk\testsuite\bootstrapping\main.exe

```
true  
***
```

OpenModelica Equation Model Debugger

The screenshot displays the OMEdit - Transformational Debugger interface with three main panels:

- Variables View:** Shows a tree structure of variables (frame, boxBody1, body, frame_a, R, T) and their definitions in equations. It includes a search bar and expand/collapse buttons.
- Equations View:** Shows a list of equations with their indices and types. Below it, the 'Equation Operations' section displays the transformation steps for a specific equation, such as 'solve', 'scalarize', 'simplify', 'inline', and 'substitute'.
- Source View:** Shows the original source code of the model, with line numbers and comments. The code includes relationships between quantities of frame_a and frame_b, and uses the 'Frames' library for planar rotations and absolute rotations.

Showing equation transformations of a model:

`0 = y + der(x * time * z); z = 1.0;`

(1) substitution:

`y + der(x * (time * z))
=>
y + der(x * (time * 1.0))`

(2) simplify:

`y + der(x * (time * 1.0))
=>
y + der(x * time)`

(3) expand derivative (symbolic diff):

`y + der(x * time)
=>y + (x + der(x) * time)`

(4) solve:

`0.0 = y + (x + der(x) * time)
=>
der(x) = ((-y) - x) / time
time <> 0`

Mapping run-time error to source model position

Transformations Browser – EngineV6 Overview (11 116 equations in model)

The screenshot displays the OMEdit - Transformational Debugger interface. The top bar shows the file path: /tmp/OpenModelica_marsj/OMEdit/Modelica.Mechanics.MultiBody.Examples.Loops.EngineV6_info.xml. The interface is divided into several panes:

- Variables Browser:** Shows a list of variables including phi, phi_offset, Crank1, and body. The phi variable is selected.
- Defined in Equations:** A table showing the definition of the selected variable. For phi, it is defined in two equations: an initial condition at index 587 and a regular assignment at index 5016.
- Used In Equations:** A table showing where the selected variable is used. It is used in several regular assignment equations, including the derivative of cylinder3.Rod.body.w_a[1].
- Equations Browser:** A table listing all equations in the model. The equation for the derivative of cylinder3.Rod.body.w_a[1] is selected.
- Source Browser:** Shows the source code for the selected equation, including assertions for frame connectivity and the definition of the rotation matrix R_rel.

The Source Browser pane shows the following code snippet:

```
386 Connections.branch(frame_a.R,  
387 frame_b.R);  
388 assert(cardinality(frame_a) > 0,  
389 "Connector frame a of Revolute  
390 joint is not connected");  
391 assert(cardinality(frame_b) > 0,  
392 "Connector frame b of Revolute  
393 joint is not connected");  
394  
395 angle = phi_offset + phi;  
396 w = der(phi);  
397 a = der(w);  
398  
399 // relationships between quantities  
400 of frame_a and of frame_b  
401 frame_b.r_theta = frame_a.r_theta;  
402  
403 if rooted(frame_a.R) then  
404 R_rel = Frames.planarRotation(e,  
405 phi_offset + phi, w);  
406 frame_b.R =  
407 Frames.absoluteRotation(frame_a.R,  
408 R_rel);  
409 frame_a.f = -  
410 Frames.resolve(R_rel, frame_b.f);  
411 frame_a.t = -  
412 Frames.resolve(R_rel, frame_b.t);  
413 else  
414 R_rel = Frames.planarRotation(-e,  
415 phi_offset + phi, w);  
416 frame_a.R =  
417 Frames.absoluteRotation(frame_b.R,  
418 R_rel);  
419 frame_b.f = -  
420 Frames.resolve(R_rel, frame_a.f);  
421 frame_b.t = -  
422 Frames.resolve(R_rel, frame_a.t);  
423 end if;  
424  
425 // d'Alemberts principle  
426 tau = -frame_b.t*e;  
427  
428 // Connection to internal
```


Equation Model Debugger on Siemens Model (Siemens Evaporator test model, 1100 equations)

The screenshot displays the OMEdit - Transformational Debugger interface. The main window shows the source code of a Siemens evaporator test model, which consists of 1100 equations. A callout box with a black border and white background points to a specific equation in the source code, stating: "Pointing out the buggy equation y = u1/u2; that gives division by zero". The callout box has two arrows pointing to the equation in the source code and the corresponding entry in the Equations Browser.

The interface includes several panels:

- Variables Browser:** Lists variables such as Scse1, Scse2, Scse3, Scse4, Scse5, Scse6, Evap, FGflow, FGINv, FGINVu, FGoutV, FGoutVu, FGsink, FGsource, FGtemp, FWinletV, FWinletVu, FWpump, FWpump_h, FWpump_p, SteamSink, and system.
- Defined In Equations:** A table with columns for Index, Type, Equation, and Executor. It lists equations like Evap.Hex.Ln.C.us[1], Evap.Hex.Ln.C.flowModel.ms[1], etc.
- Used In Equations:** A table with columns for Index, Type, Equation, and Executor. It lists equations like Evap.Hex.Ln.Q2.division2.u1, Evap.Hex.Ln.Q2.division2.u2, etc.
- Equations Browser:** A table with columns for Index, Type, and Equation. It lists equations like Evap.Hex.Ln.C.us[1], Evap.Hex.Ln.C.flowModel.ms[1], etc.
- Source Browser:** Shows the source code of the Siemens evaporator test model, including the equation `y = u1 / u2;` which is highlighted in blue. The code also includes comments and annotations.

Debugging Example – Detecting Source of Chattering (excessive event switching) causing bad performance

The screenshot shows the OMEdit - Transformational Debugger interface. The main window displays the source code of a model, with the following code visible:

```
1 within ;
2 package Debugging "Test
  cases for debugging of
  declarative models"
3
4 package Chattering "Models
  with chattering behaviour"
5 model ChatteringEvents1
6   "Exhibits chattering
  after t = 0.5, with
  generated events"
7   Real x(start=1,
  fixed=true);
8   Real y;
9   Real z;
  equation
  z = if x > 0 then -1
  else 1;
12  y = 2*z;
13  der(x) = y;
14  annotation
  (Documentation(info="<html>
15  <p>After t = 0.5, chattering
  takes place, due to the
  feedback loop in the right
  hand side of the
  equation.</p>
  <p>Chattering can be
  detected because lots of
  tightly spaced events are
  generated. The feedback to
  the user should allow to
  identify the equation from
  which the zero crossing
  function that generates the
  events originates.</p>
16  </html>"),
  experiment(StopTime=1));
18 end ChatteringEvents1;
19
20 model ChatteringEvents2
21   "Exhibits chattering
  after t = 0.422, with
  generated events"
```

The equation `z = if x > 0 then -1 else 1;` is highlighted in blue. A black arrow points from this equation to the text `equation z = if x > 0 then -1 else 1;` in the foreground. Below the code, the text `y = 2*z;` is also visible.

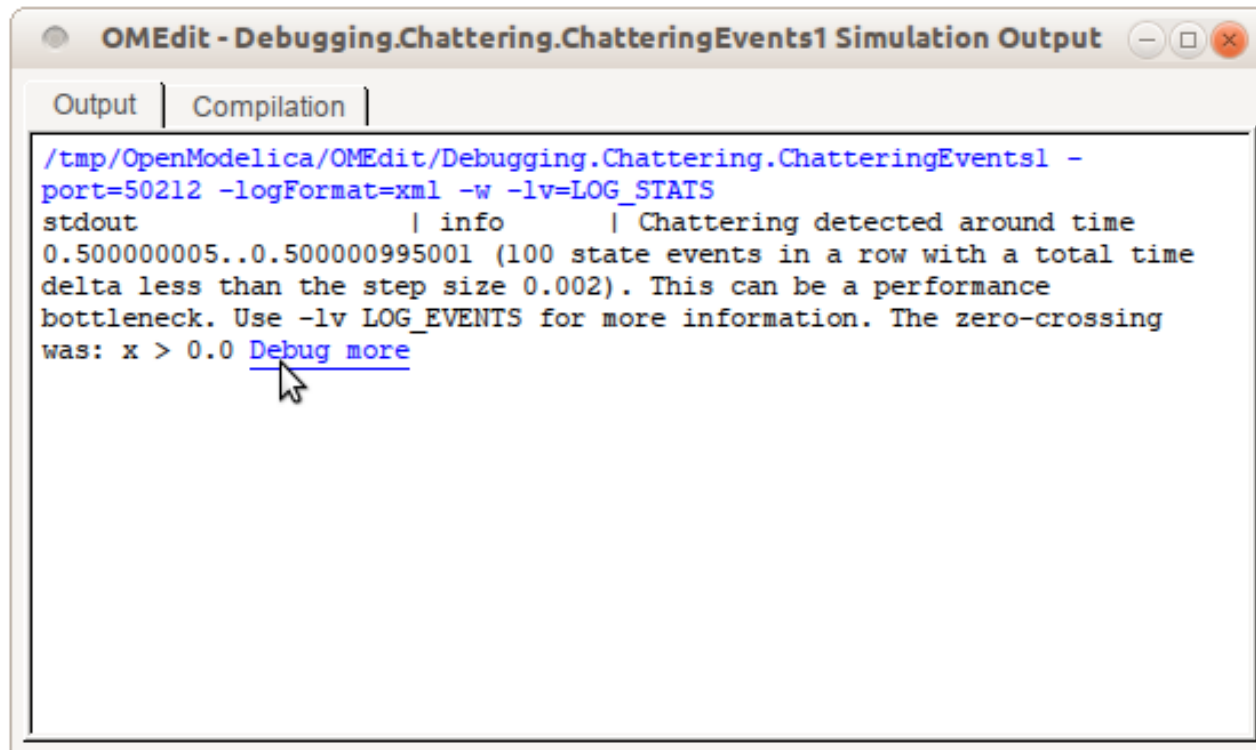
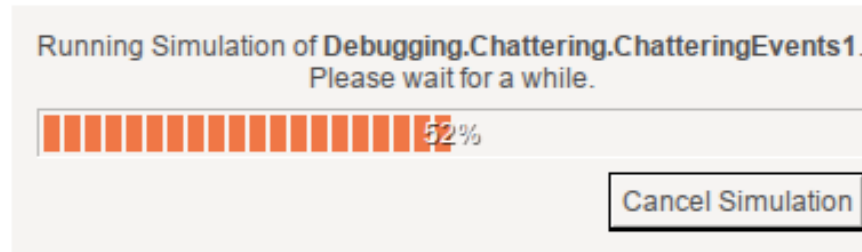
The interface also shows the Variables Browser, Equations Browser, and Variable Operations panels. The Equations Browser shows the following equations:

Inc	Type	Equation
-1	initial	(assignment) x = 1.0
-2	initial	(assignment) y = 2.0 * z
-3	initial	(assignment) der(x) = y
-4	initial	(assignment) y = 2.0 * z
-5	regular	(assignment) z = if x > 0 then -1 else 1
-6	regular	(assignment) y = 2.0 * z
-7	regular	(assignment) der(x) = y

The Variable Operations panel shows the following operations:

```
solved: z = if x > 0.0 then -1.0 else 1.0
original: z = if x > 0 then -1 else 1; => flattened: z = if x > 0.0 then -1.0 else 1.0;
```

Error Indication – Simulation Slows Down



Exercise 1.2 – Equation-based Model Debugger

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

```
model ChatteringEvents1
  Real x(start=1, fixed=true);
  Real y;
  Real z;
equation
  z = noEvent(if x > 0 then -1 else 1);
  y = 2*z;
  der(x) = y;
end ChatteringNoEvents1;
```

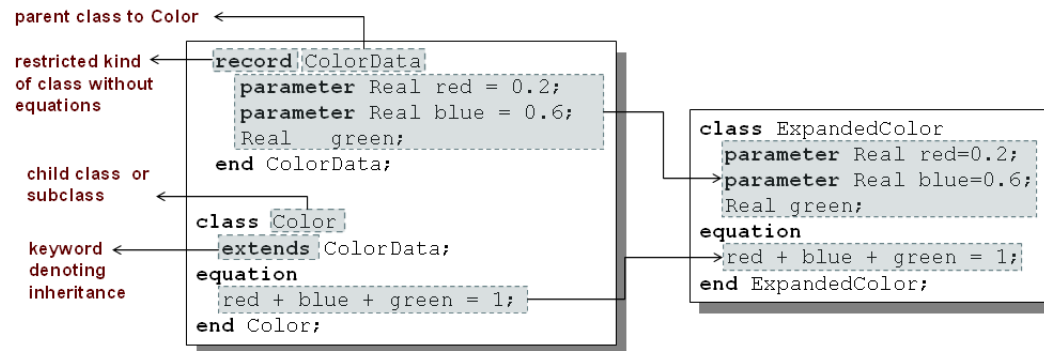
Uses 25% CPU

acrotray.exe *32	petfr27	00	976 K	A
AdobeARM.exe *32	petfr27	00	1,136 K	A
Bootcamp.exe	petfr27	00	1,448 K	B
conhost.exe	petfr27	00	1,300 K	C
csrss.exe		00	3,000 K	
DCSHelper.exe *32	petfr27	00	660 K	D
Debugging.Chattering...	petfr27	25	1,436 K	D
dllhost.exe	petfr27	00	2,224 K	C

- Switch to OMEdit text view (click on text button upper left)
- Open the Debugging.mo package file using OMEdit
- Open subpackage Chattering, then open model ChatteringEvents1
- Simulate in debug mode
- Click on the button Debug more (see prev. slide)
- Possibly start task manager and look at CPU. Then click stop simulation button

Part III

Modelica language concepts and textual modeling



Typed
Declarative
Equation-based
Textual Language

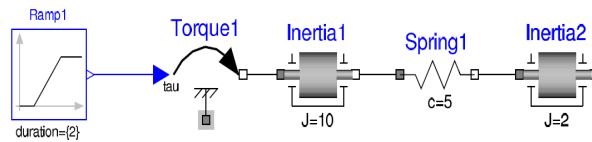
Hybrid
Modeling

Acausal Modeling

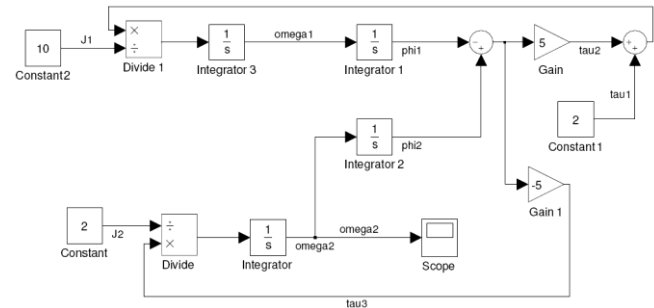
The order of computations is not decided at modeling time

Visual
Component
Level

Acausal



Causal



Equation
Level

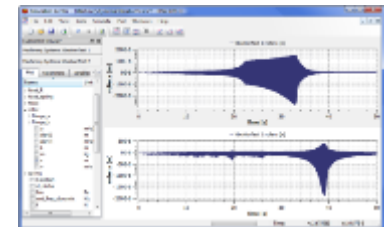
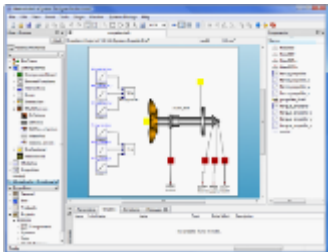
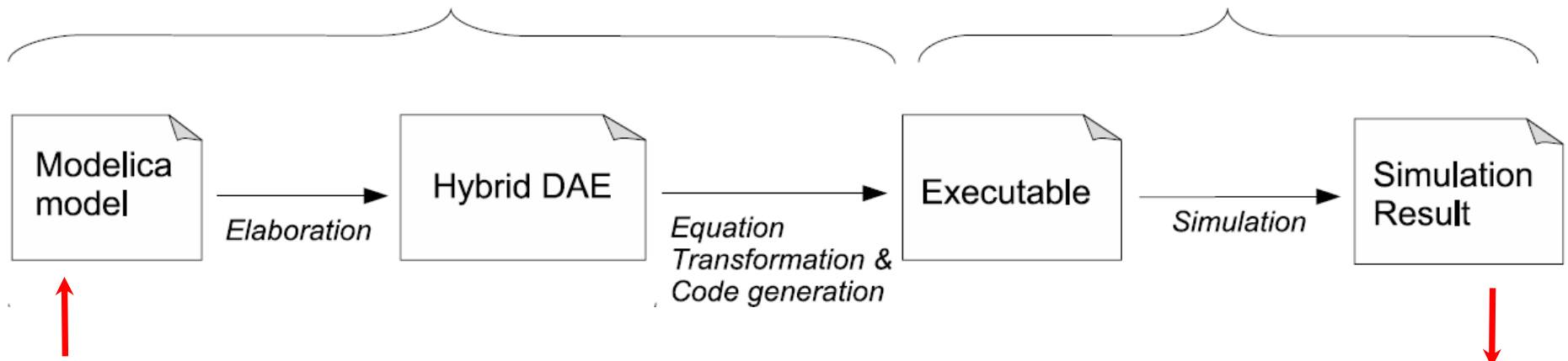
A resistor *equation*:
 $R \cdot i = v;$

Causal possibilities:
 $i := v/R;$
 $v := R \cdot i;$
 $R := v/i;$

Typical Simulation Process

“Static” semantics / compile time

“Dynamic” semantics / run time



Simple model - Hello World!

Equation: $x' = -x$

Initial condition: $x(0) = 1$

Continuous-time
variable

Parameter, constant
during simulation

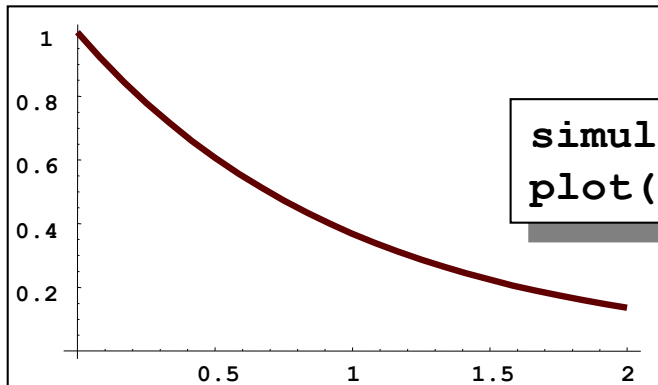
Name of model

Initial condition

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

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

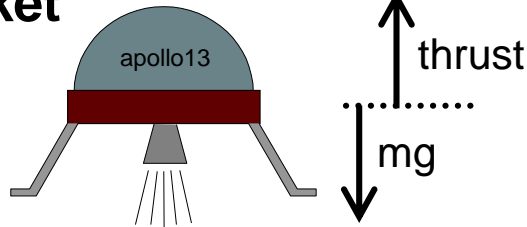
- Two types of constants in Modelica

- **constant**
- **parameter**

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

A Simple Rocket Model

Rocket



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

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

$$altitude' = velocity$$

$$velocity' = acceleration$$

new model ←

parameters (changeable before the simulation) ←

floating point type ←

differentiation with regards to time ←

```
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;
```

→ 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;
```

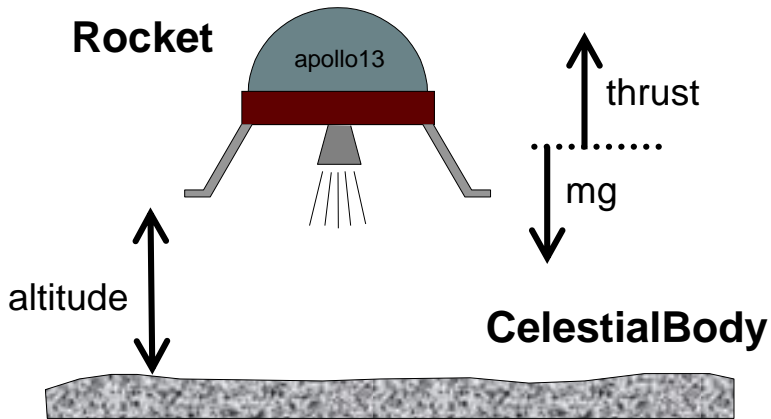


An *instance* of the class can be declared by *prefixing* the type name to a variable name

```
...
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

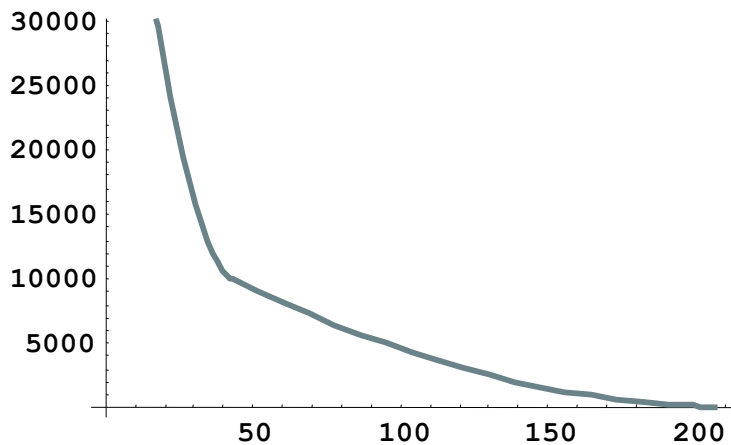
```

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;

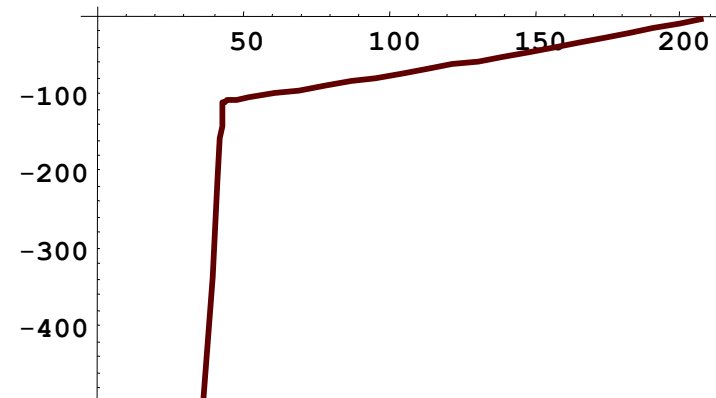
```

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

Specialized Class Keywords

- Classes can also be declared with other keywords, e.g.: `model`, `record`, `block`, `connector`, `function`, ...
- Classes declared with such keywords have specialized properties
- Restrictions and enhancements apply to contents of specialized classes
- After Modelica 3.0 the `class` keyword means the same as `model`
- Example: (Modelica 2.2). A `model` is a class that cannot be used as a connector class
- Example: A `record` is a class that only contains data, with no equations
- Example: A `block` is a class with fixed input-output causality

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

Modelica Functions

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

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

Function Call – Example Function with for-loop

Example Modelica function call:

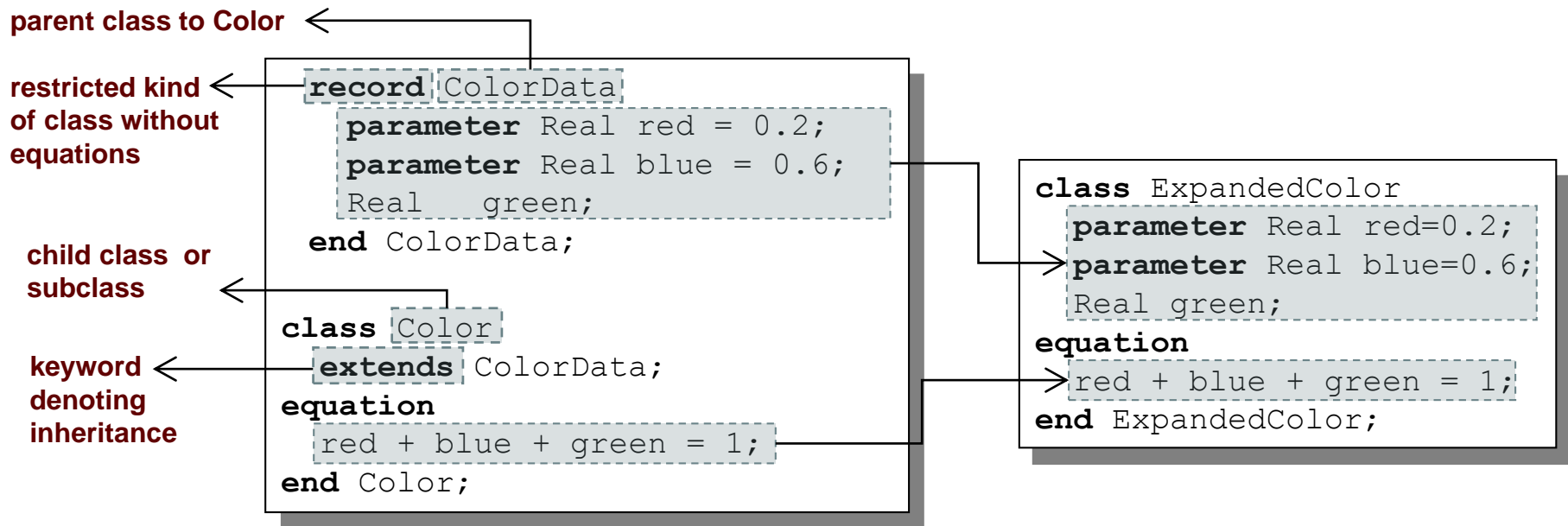
```
...  
p = polynomialEvaluator({1,2,3,4},21)
```

```
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;
```

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

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

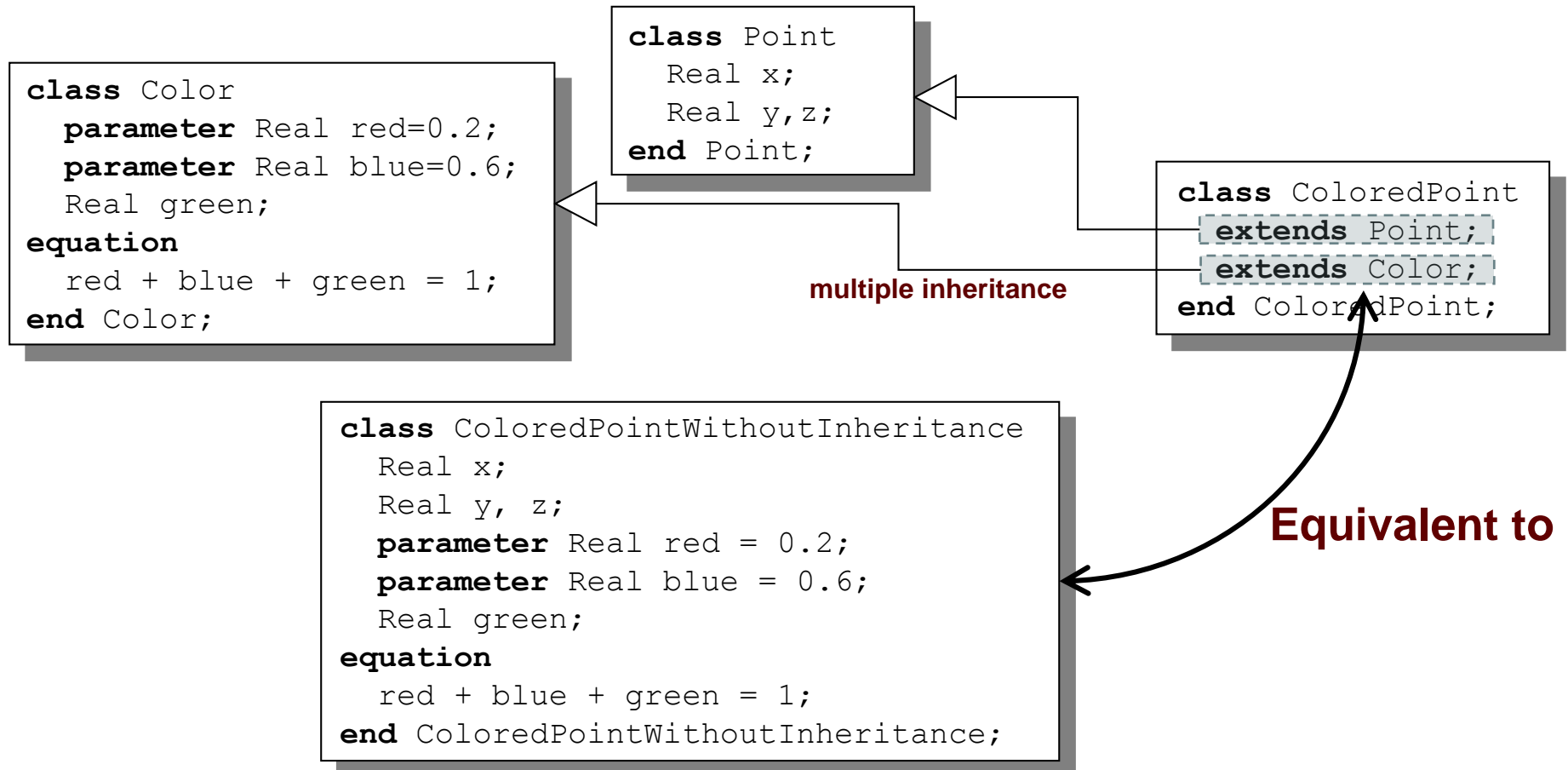
Inheritance



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

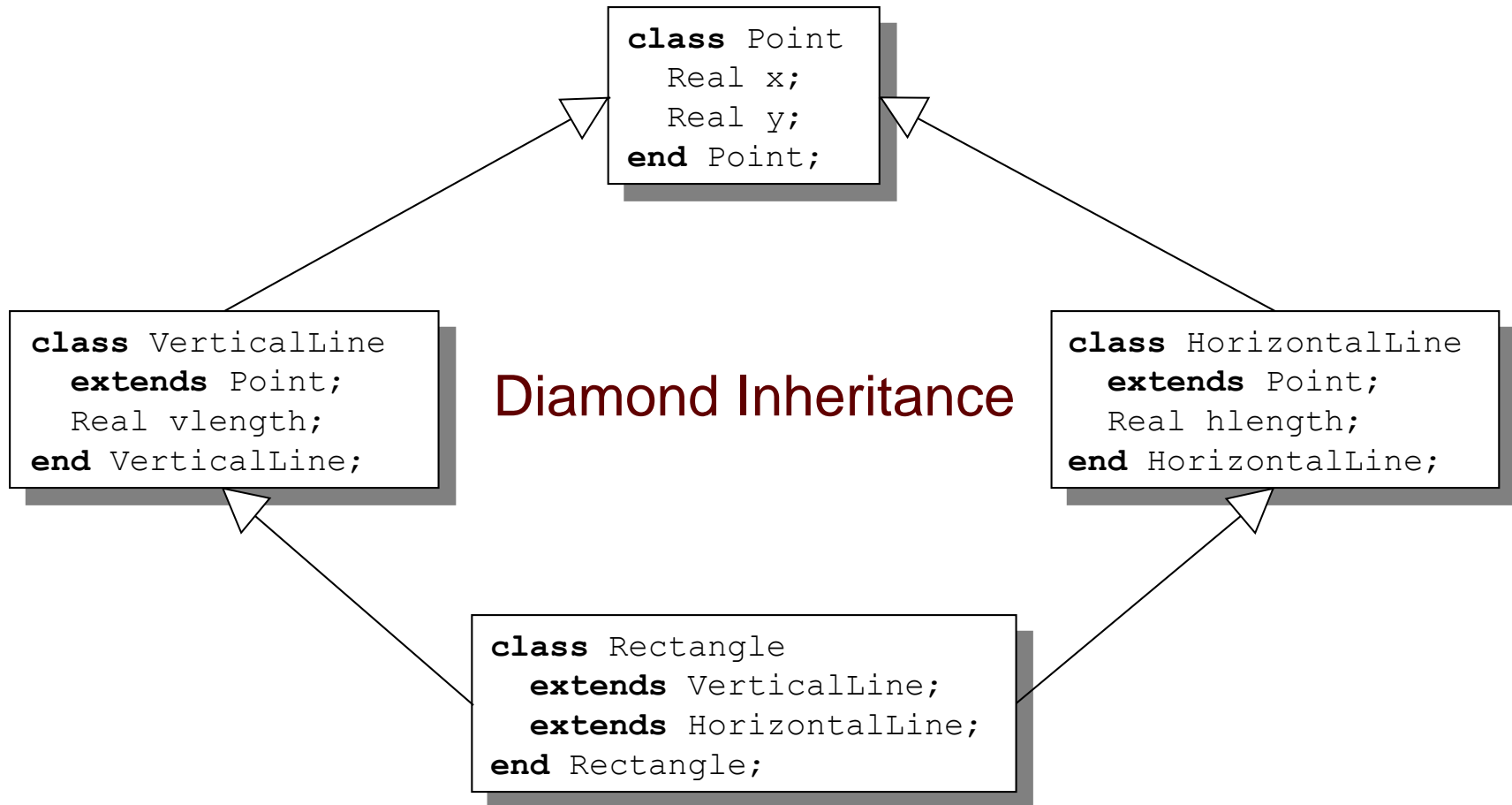
Multiple Inheritance

Multiple Inheritance is fine – inheriting both geometry and color



Multiple Inheritance cont'

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



Simple Class Definition

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

```
class SameColor = Color;
```

Equivalent to:

inheritance ←

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

- Often used for introducing new names of types:

```
type Resistor = Real;
```

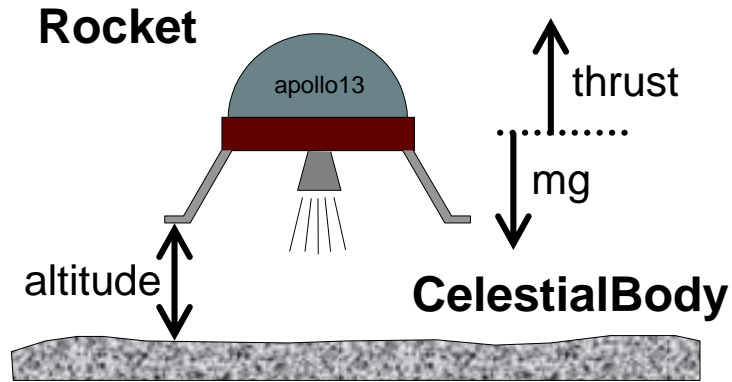
```
connector MyPin = Pin;
```

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)

```

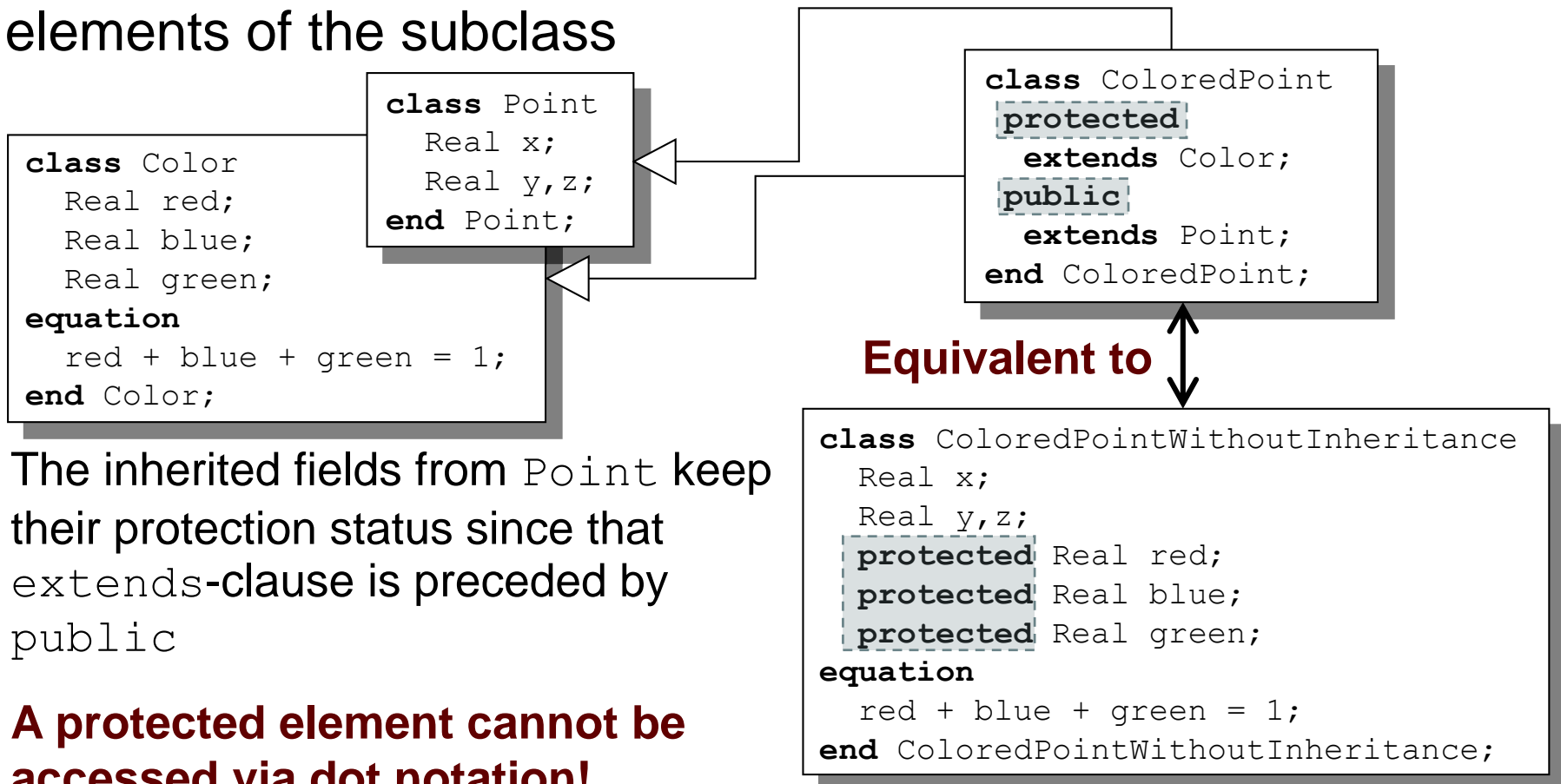
model MoonLanding
  parameter Real force1 = 36350;
  parameter Real force2 = 1308;
  parameter Real thrustEndTime = 210;
  parameter Real thrustDecreaseTime = 43.2;
  Rocket      apollo (name="apollo13", mass(start=1038.358) );
  CelestialBody moon (mass=7.382e22, radius=1.738e6, name="moon");
equation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

```

inherited
parameters

Inheritance of Protected Elements

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



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

A protected element cannot be accessed via dot notation!

Exercises Part III a (15 minutes)

Exercises Part III a

- Start OMNotebook (part of OpenModelica)
 - **Start**->Programs->OpenModelica->OMNotebook
 - **Open File**: Exercises-ModelicaTutorial.onb from the directory you copied your tutorial files to.
 - **Note**: The DrModelica electronic book has been automatically opened when you started OMNotebook.
 - **(Alternatively)**: Open the OMWeb notebook
<http://omwebbook.openmodelica.org/>
- Open Exercises-ModelicaTutorial.pdf (also available in printed handouts)

Exercises 2.1 and 2.2 (See also next two pages)

- Open the **Exercises-ModelicaTutorial.onb** found in the Tutorial directory you copied at installation.
- **Exercise 2.1.** Simulate and plot the HelloWorld example. Do a slight change in the model, re-simulate and re-plot. Try command-completion, `val()`, etc.

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

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

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

Exercise 2.1 – Hello World!

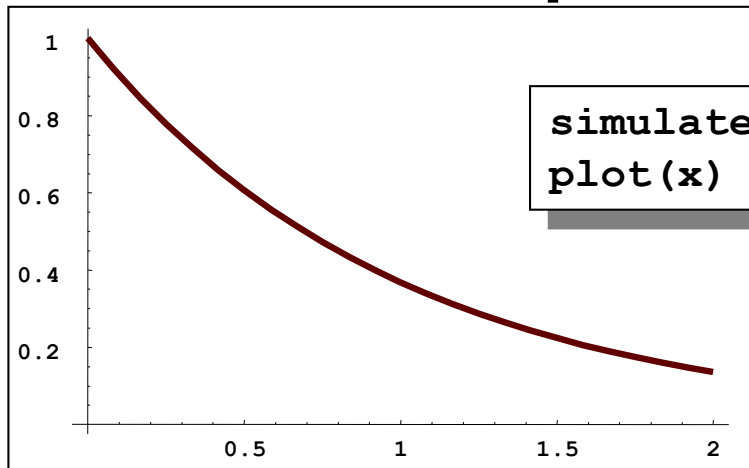
A Modelica “Hello World” model

Equation: $x' = -x$

Initial condition: $x(0) = 1$

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

Simulation in OpenModelica environment

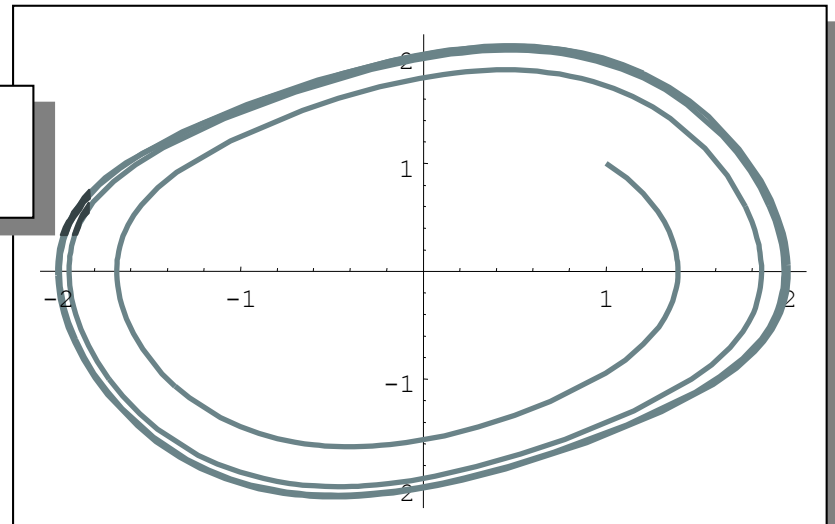


```
simulate(HelloWorld, stopTime = 2)  
plot(x)
```

(extra) 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)
```



(extra) Exercise 2.3 – DAE Example

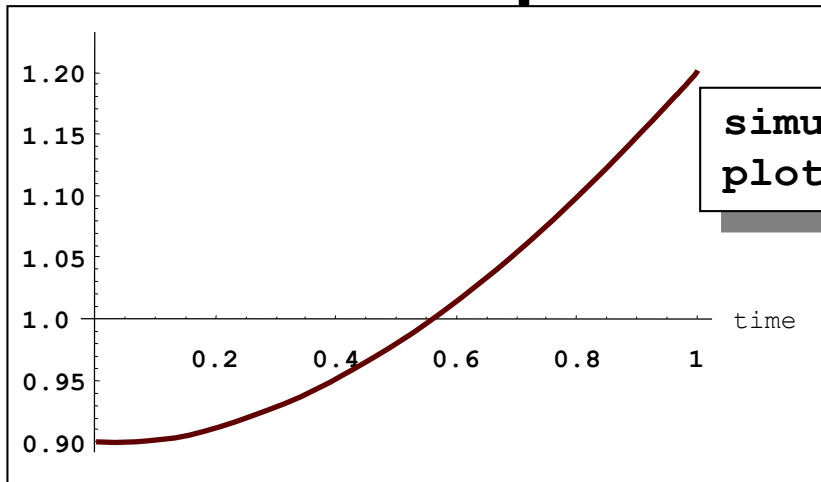
Include algebraic equation

Algebraic equations contain
no derivatives

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

```
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

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

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

$$x(0) = 2$$

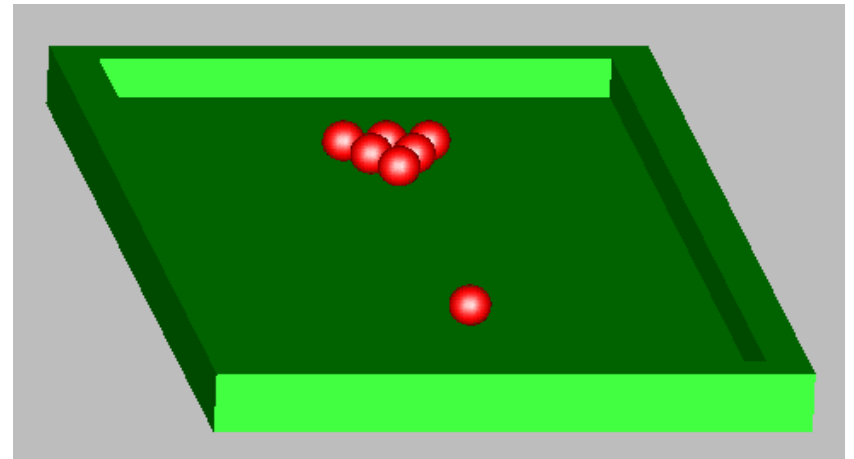
$$y(0) = 3$$

(extra) Exercise 2.5 – Functions

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

Part III b

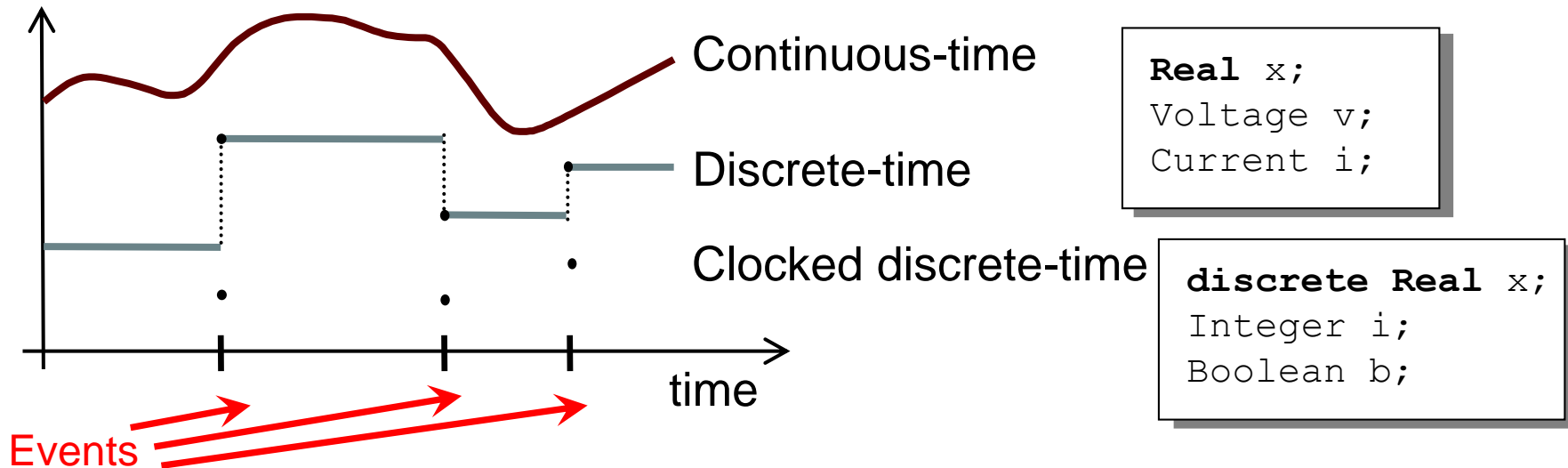
Discrete Events and Hybrid Systems



Picture: Courtesy Hilding Elmqvist

Modelica Hybrid Modeling

Hybrid modeling = continuous-time + discrete-time modeling



- A *point* in time that is instantaneous, i.e., has zero duration
- An event *condition* or *clock tick* 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 (two kinds: unclocked and clocked)

```
when <conditions> then  
  <equations>  
end when; // un-clocked version
```

```
when clock then  
  <equations>  
end when; // clocked version
```



Equations only active at event times

Time event

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

Only dependent on time, can be scheduled in advance

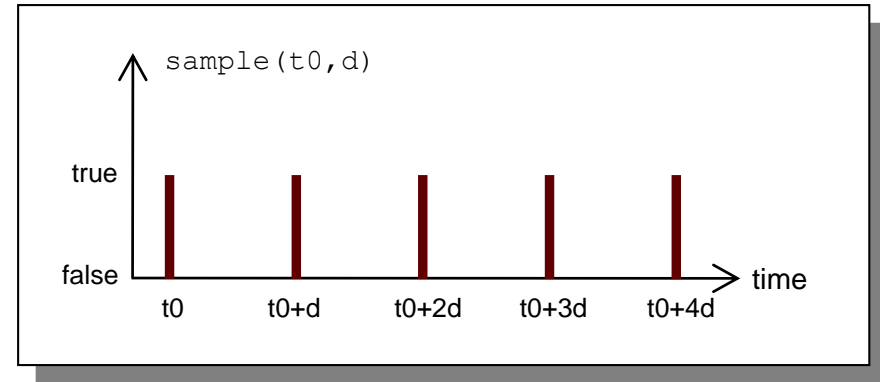
State event

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

Related to a state. Check for zero-crossing

Generating Repeated Events by unclocked sample

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

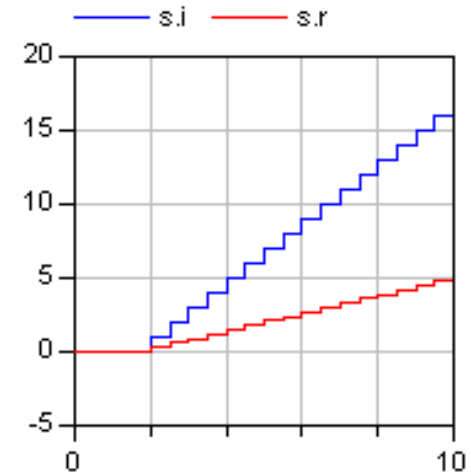


Variables need to be discrete

```
model SamplingClock
  Integer i;
  discrete Real r;
equation
  when sample(2, 0.5) then
    i = pre(i)+1;
    r = pre(r)+0.3;
  end when;
end SamplingClock;
```

Creates an event after 2 s, then each 0.5 s

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



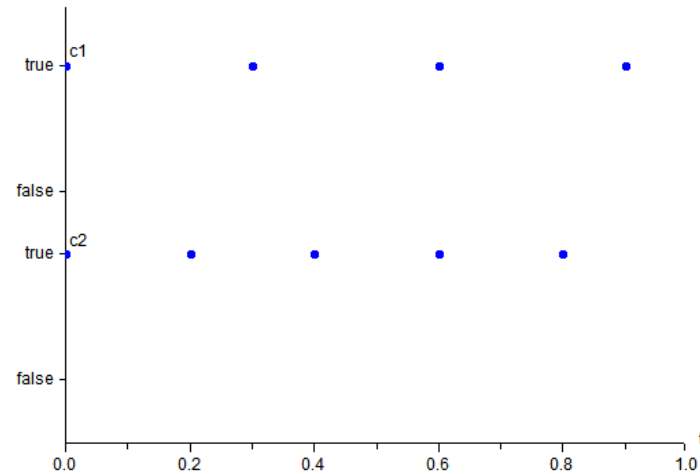
Generating Clock Tick Events using Clock()

(clocked models, Modelica 3.3)

- Clock() – inferred clock
- Clock(intervalCounter, resolution) – clock with Integer quotient (rational number) interval
- Clock(interval) – clock with a Real value interval
- Clock(condition, startInterval)
- Clock – solver clock

class ClockTicks

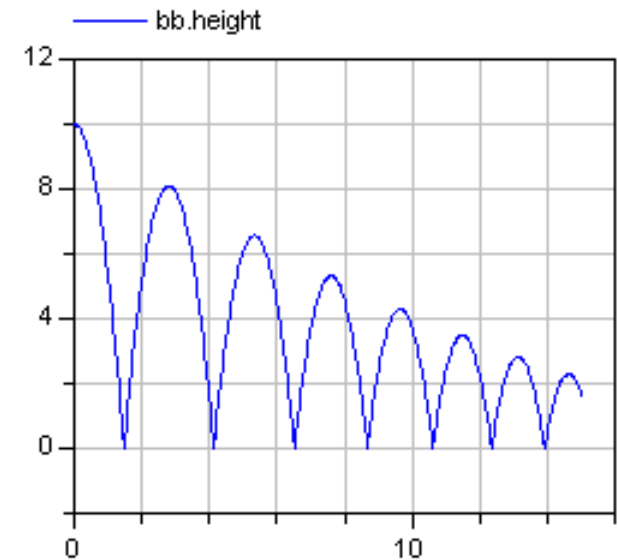
```
// Integer quotient rational number interval clock
Clock c1 = Clock(3,10);    // ticks: 0, 3/10, 6/10, ..
// Clock with real value interval between ticks
Clock c2 = Clock(0.2);    // ticks: 0.0, 0.2, 0.4, ...
end ClockTicks;
```



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;
```

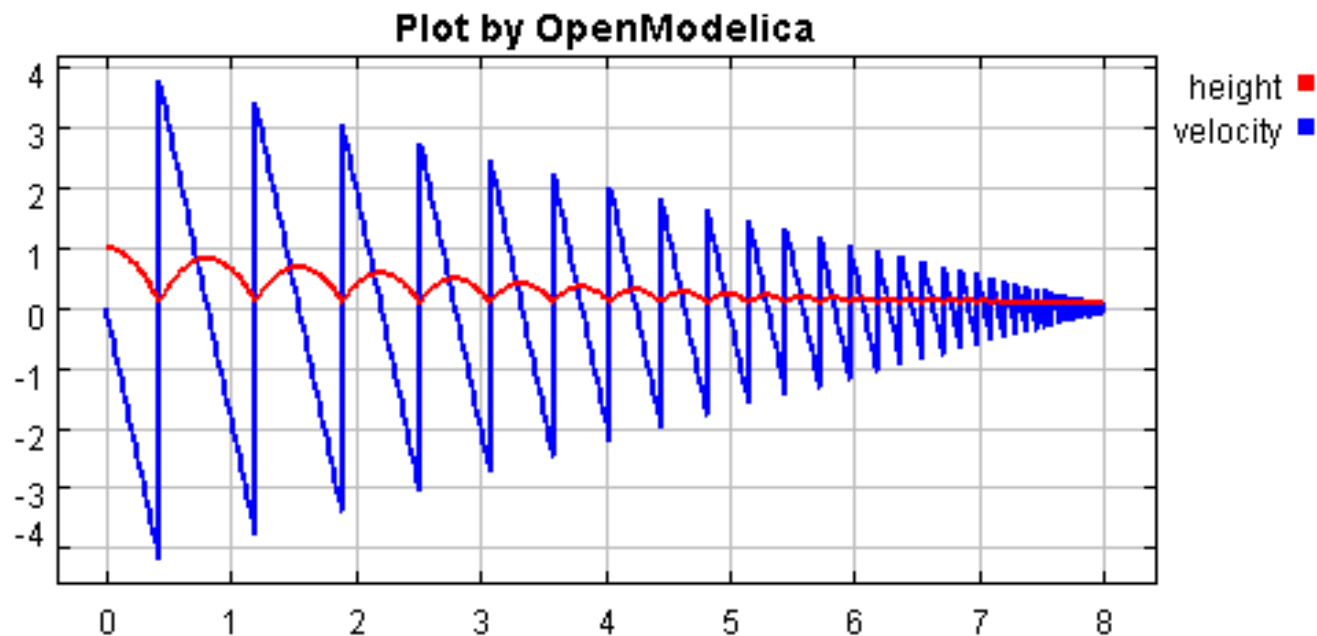


Initial conditions

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 IIIc

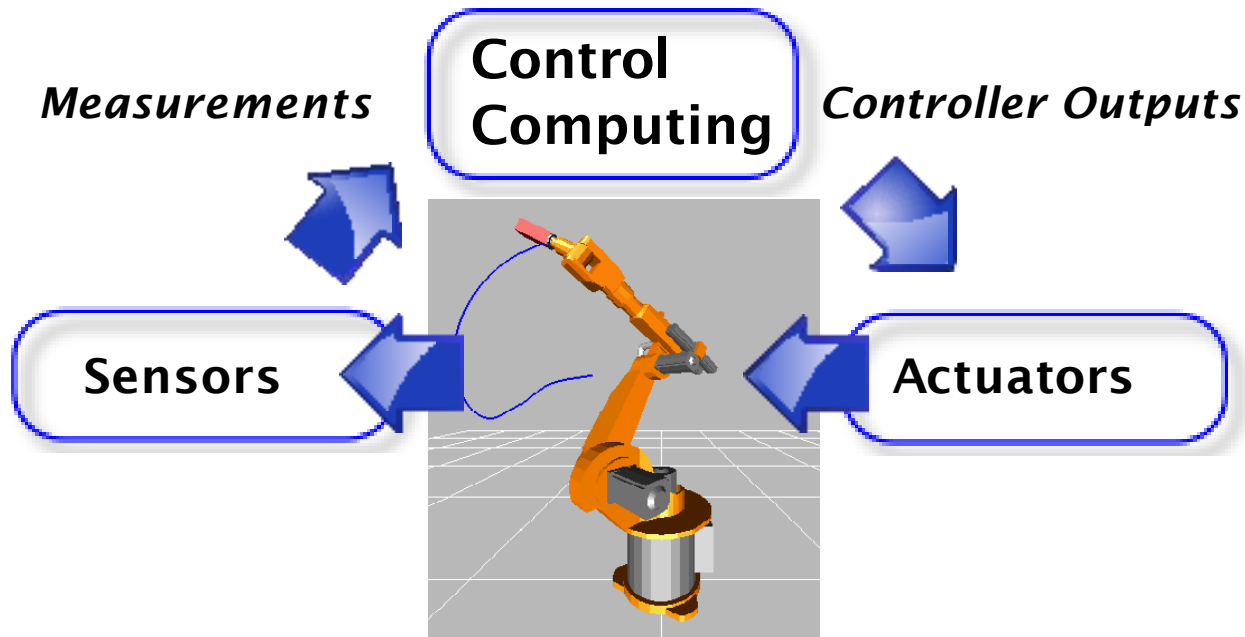
Clocked Synchronous Models and State Machines

and Applications for Digital Controllers

Control System Applications

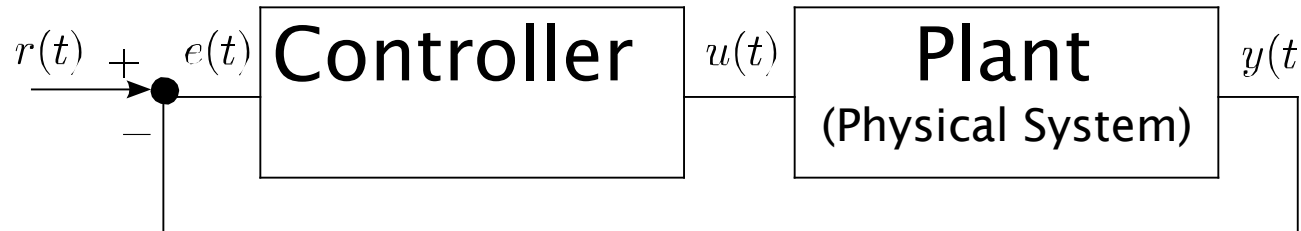
Control System

A control system is a device, or set of devices, that manages, commands, directs or regulates the behavior of other devices or systems (wikipedia).



Control Theory Perspective

Feedback Control System

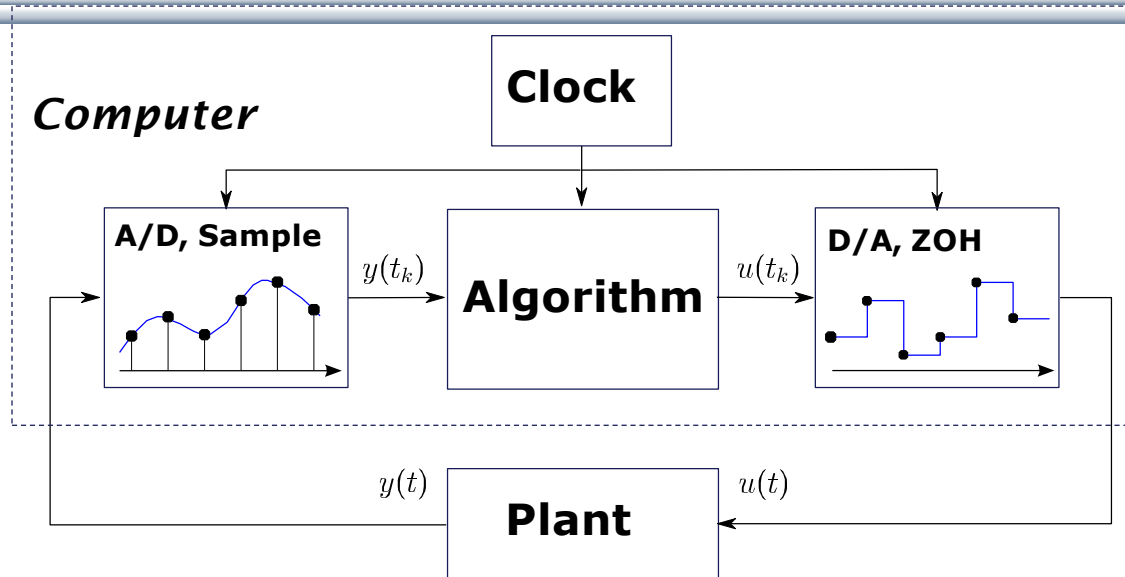


- $r(t)$ reference (setpoint)
- $e(t)$ error
- $y(t)$ measured process variable (plant output)
- $u(t)$ control output variable (plant input)

Usual Objective

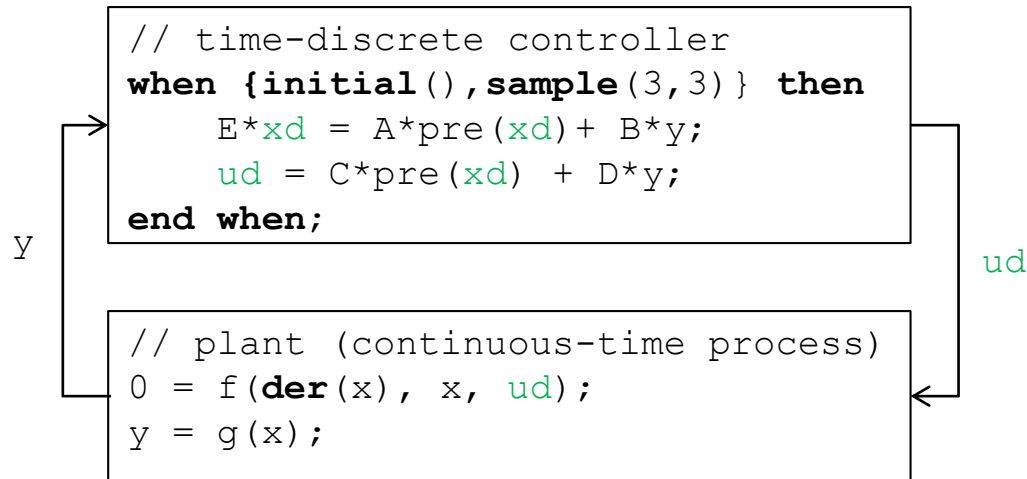
Plant output should follow the reference signal.

Embedded Real-Time Control System



1. **Discrete-time** controller + **continuous-time** plant \equiv *hybrid system* or *sampled-data system*
2. Interface between digital and analog world: Analog to Digital and Digital to Analog Converters (ADC and DAC).
3. ADC \rightarrow Algorithm \rightarrow DAC is synchronous (zero-delay model!)
4. A *clock* controls the *sampling instants*. Usually *periodic sampling*.

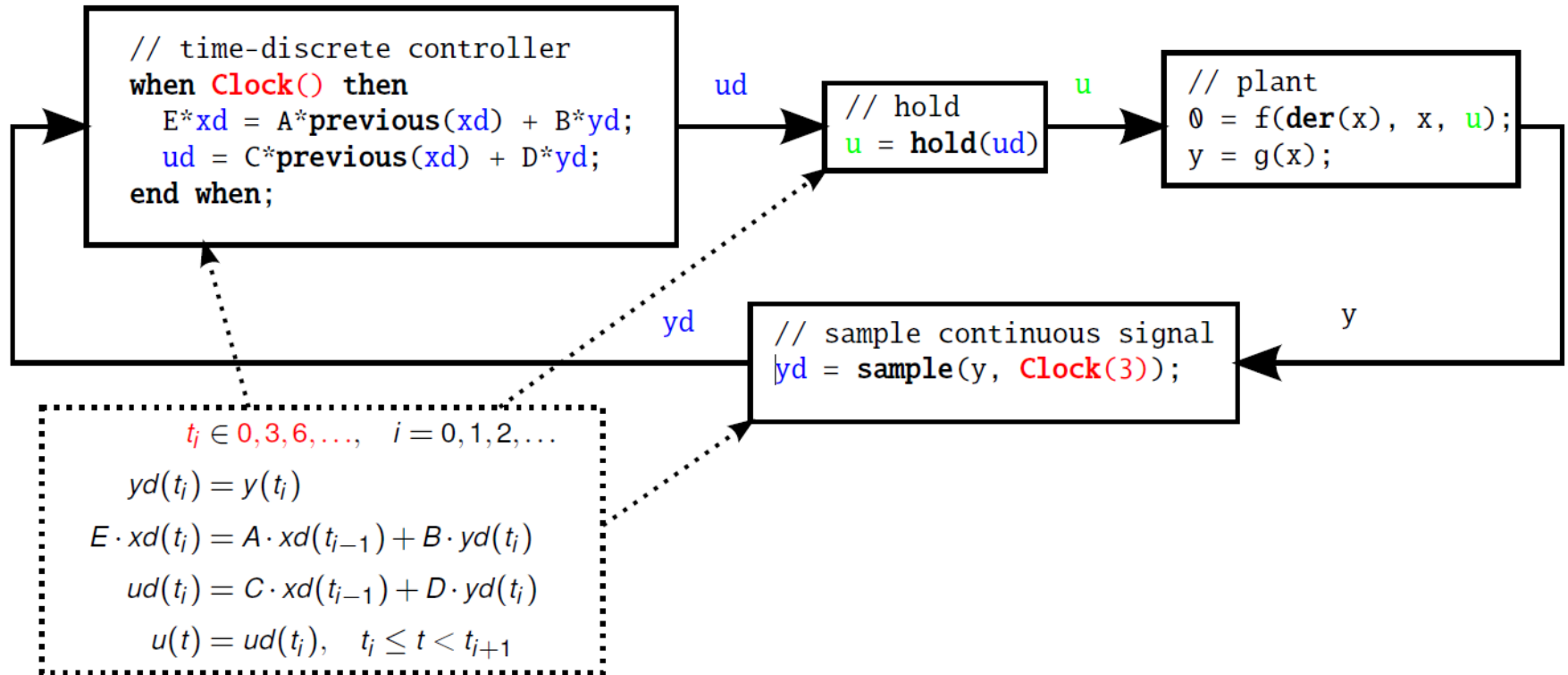
Controller with Sampled Data-Systems (unlocked models, using pre() and sample())



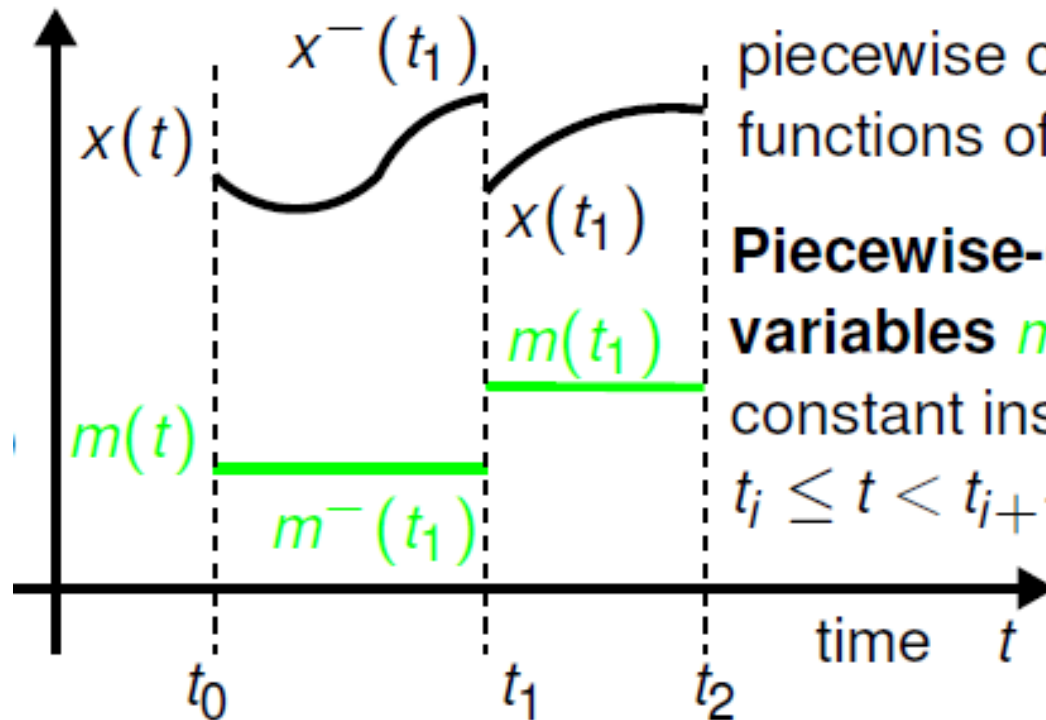
- y is automatically sampled at $t = 3, 6, 9, \dots$;
- xd , u are piecewise-constant variables that change values at sampling events (implicit zero-order hold)
- `initial()` triggers event at initialization ($t=0$)

Controller with Clocked Synchronous Constructs

clocked models using `Clock()`, `previous()`, `hold()` in Modelica 3.3



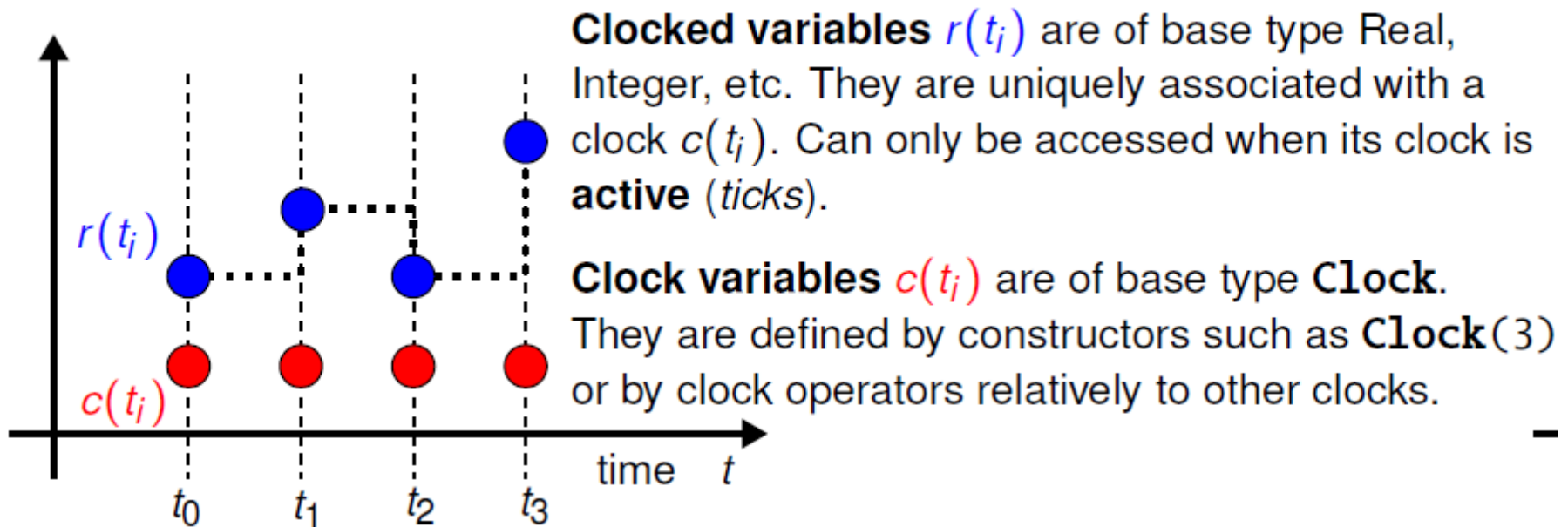
Unlocked Variables in Modelica 3.2



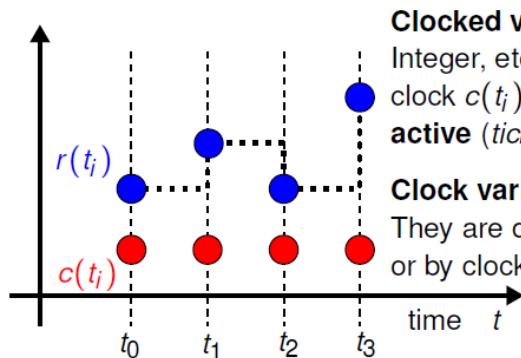
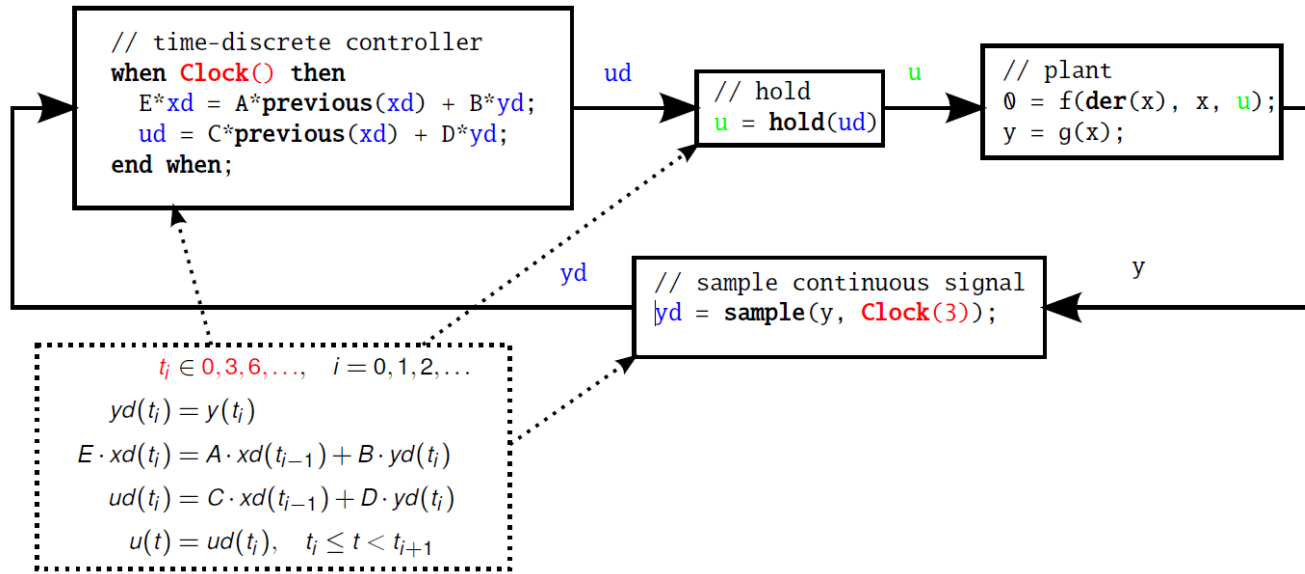
Continuous variables are Real numbers defined as piecewise continuous functions of time.

Piecewise-constant variables $m(t)$ are constant inside each $t_j \leq t < t_{j+1}$.

Clock variables (Clock) and Clocked Variables (Real) (in Modelica 3.3)

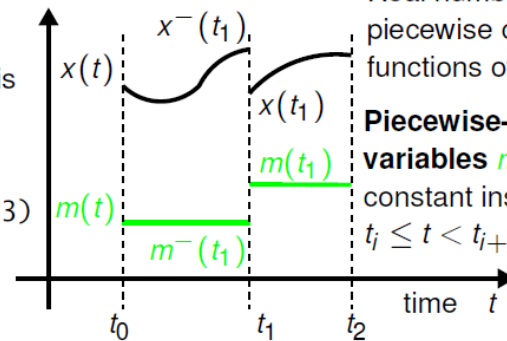


Clocked Synchronous Extension in Modelica 3.3



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

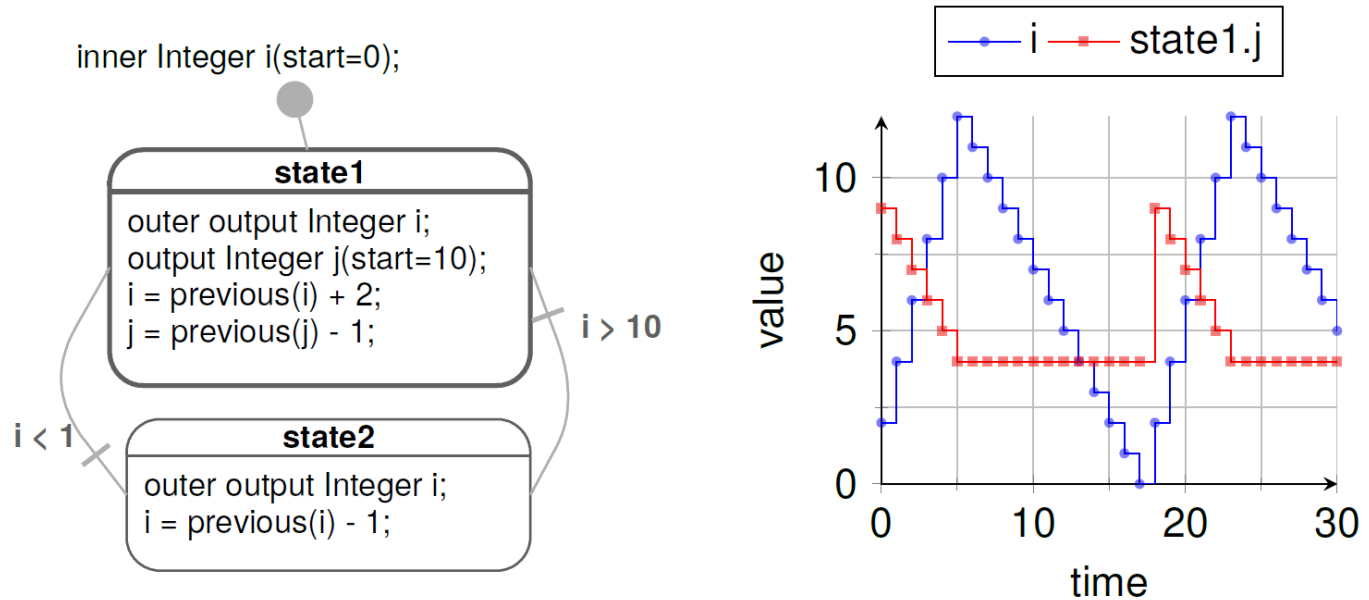
Clock variables $c(t_i)$ are of base type **Clock**. They are defined by constructors such as **Clock(3)** or by clock operators relatively to other clocks.



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

Piecewise-constant variables $m(t)$ are constant inside each $t_i \leq t < t_{i+1}$.

State Machines in Modelica 3.3: Simple Example

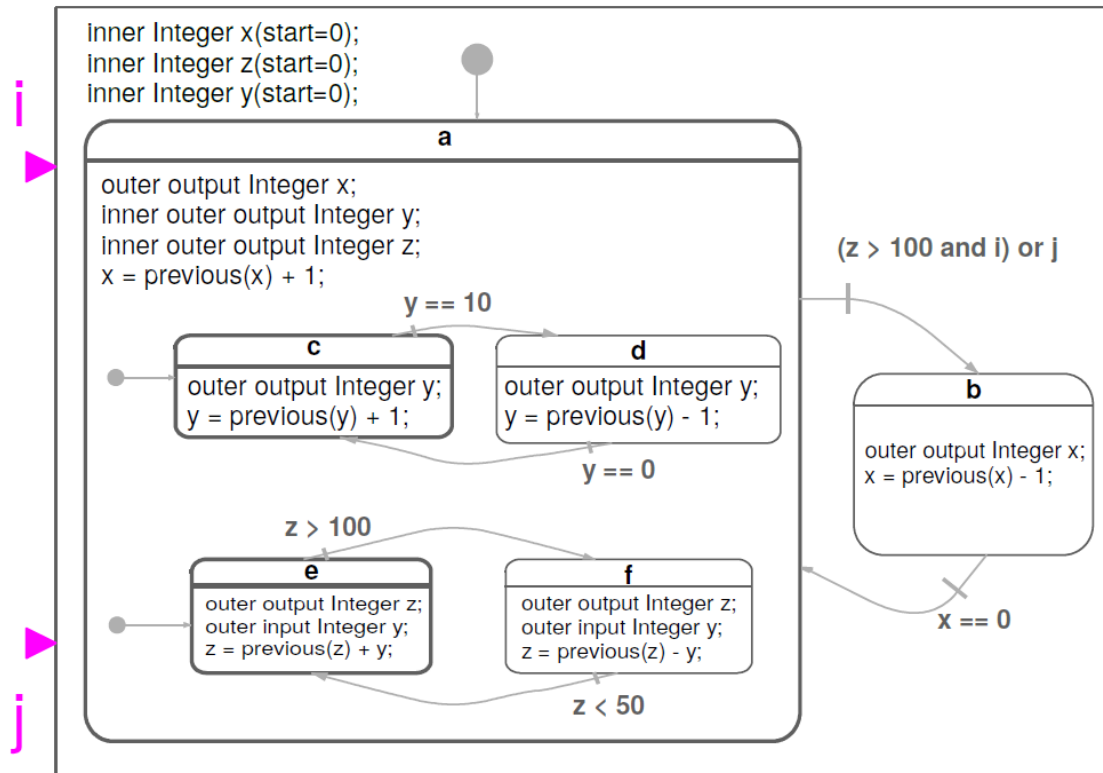


- Equations are active if corresponding *clock* ticks. Defaults to periodic clock with 1.0 s sampling period
- “ i ” is a shared variable, “ j ” is a local variable. Transitions are “*delayed*” and enter states by “*reset*”

Simple Example: Modelica Code

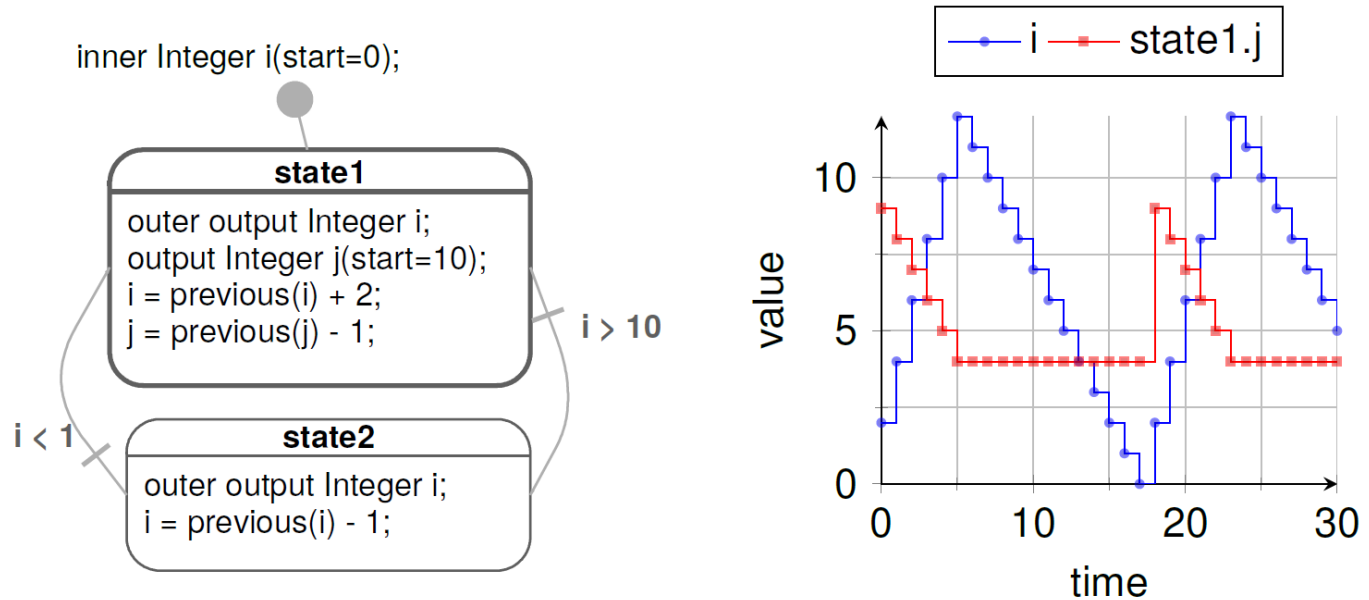
```
model Simple_NoAnnotations "Simple state machine"
  inner Integer i(start=0);
  block State1
    outer output Integer i;
    output Integer j(start=10);
  equation
    i = previous(i) + 2;
    j = previous(j) - 1;
  end State1;
  State1 state1;
  block State2
    outer output Integer i;
  equation
    i = previous(i) - 1;
  end State2;
  State2 state2;
equation
  transition(state1, state2, i > 10, immediate=false);
  transition(state2, state1, i < 1, immediate=false);
  initialState(state1);
end Simple_NoAnnotations;
```

Hierarchical and Parallel Composition of Modelica State Machine Models



Semantics of Modelica state machines (and example above) inspired by Florence Maraninchi & Yann Rémond's "Mode-Automata" and by Marc Pouzet's Lucid Synchrone 3.0.

State Machines in Modelica 3.3: Simple Example



- Equations are active if corresponding *clock* ticks. Defaults to periodic clock with 1.0 s sampling period
- “ i ” is a shared variable, “ j ” is a local variable. Transitions are “*delayed*” and enter states by “*reset*”

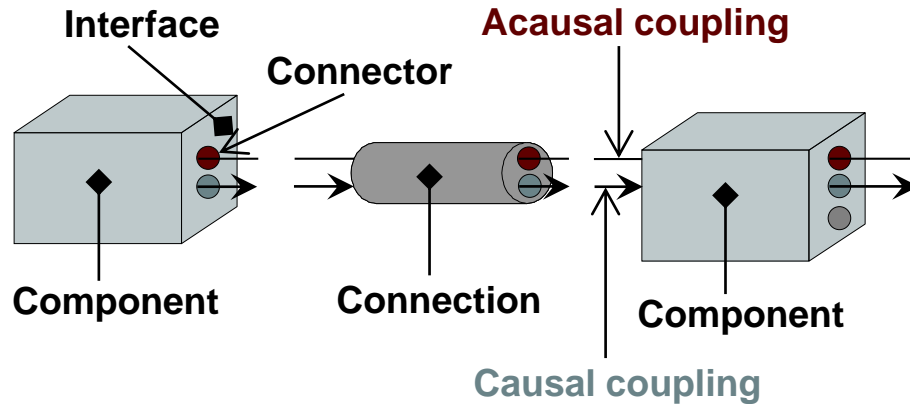
Simple Example: Modelica Code

```
model Simple_NoAnnotations "Simple state machine"
  inner Integer i(start=0);
  block State1
    outer output Integer i;
    output Integer j(start=10);
  equation
    i = previous(i) + 2;
    j = previous(j) - 1;
  end State1;
  State1 state1;
  block State2
    outer output Integer i;
  equation
    i = previous(i) - 1;
  end State2;
  State2 state2;
equation
  transition(state1, state2, i > 10, immediate=false);
  transition(state2, state1, i < 1, immediate=false);
  initialState(state1);
end Simple_NoAnnotations;
```

Part IV

Components, Connectors and Connections – Modelica Libraries and Graphical Modeling

Software Component Model



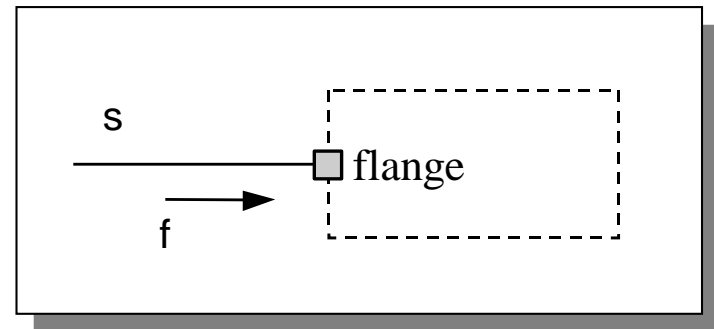
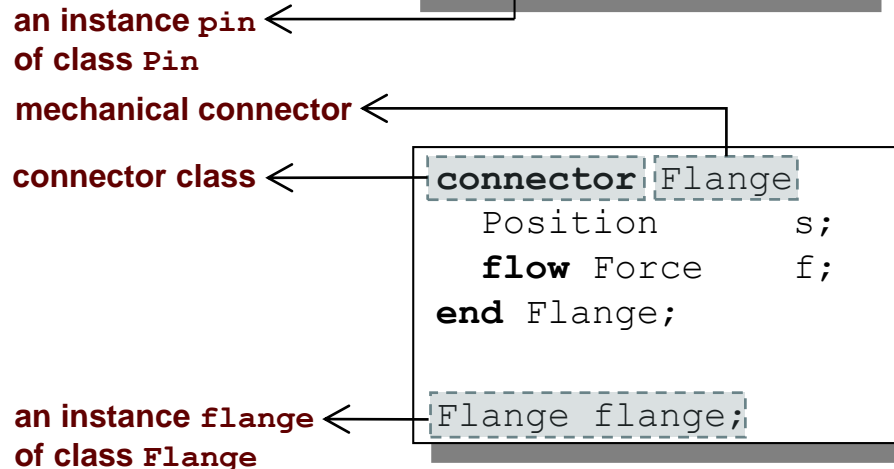
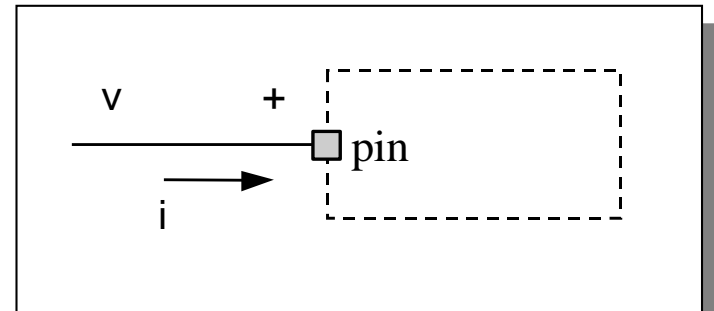
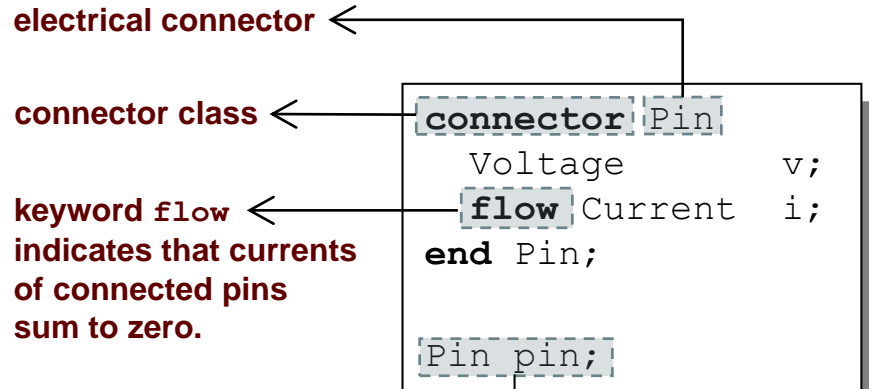
A component class should be defined *independently of the environment*, very essential for *reusability*

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

Complex systems usually consist of large numbers of *connected* components

Connectors and Connector Classes

Connectors are instances of *connector classes*



The `flow` prefix

Three possible kinds of variables in connectors:

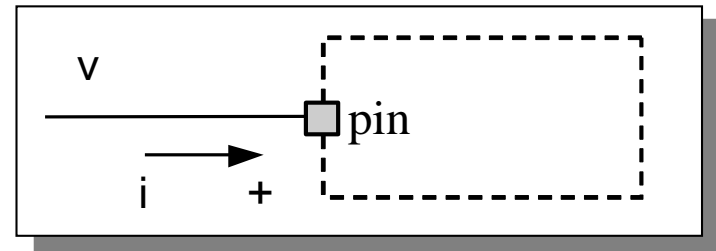
- *Potential variables* *potential* or energy level
- *Flow variables* represent some kind of flow
- *Stream variables* represent fluid flow in convective transport

Coupling

- *Equality coupling*, for potential 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

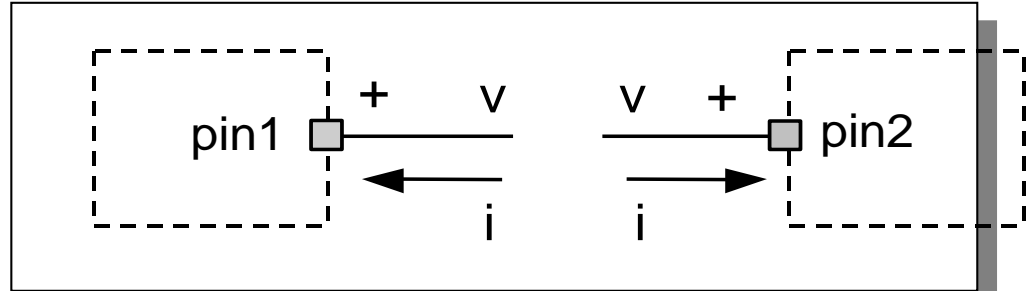
Domain Type	Potential	Flow	Carrier	Modelica Library
Electrical	Voltage	Current	Charge	Electrical. Analog
Translational	Position	Force	Linear momentum	Mechanical. Translational
Rotational	Angle	Torque	Angular momentum	Mechanical. Rotational
Magnetic	Magnetic potential	Magnetic flux rate	Magnetic flux	
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 **potential** variables is used to generate equations of the form:

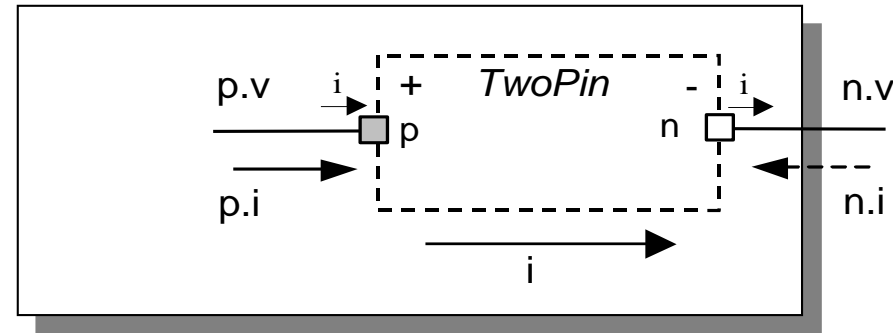
$$V_1 = V_2 = V_3 = \dots V_n$$

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

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

Common Component Structure

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



partial class
(cannot be
instantiated)

positive pin
negative pin

```
partial model TwoPin
  Voltage v
  Current i
  Pin p;
  Pin n;
equation
  v = p.v - n.v;
  0 = p.i + n.i;
  i = p.i;
end TwoPin;
// TwoPin is same as OnePort in
// Modelica.Electrical.Analog.Interfaces
```

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

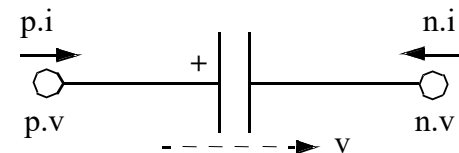
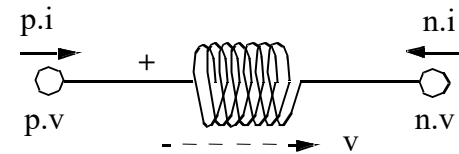
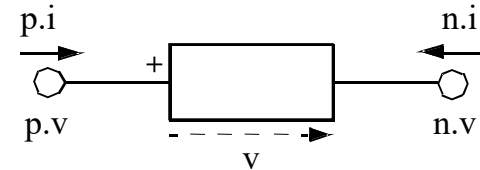
electrical connector class

Electrical Components

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

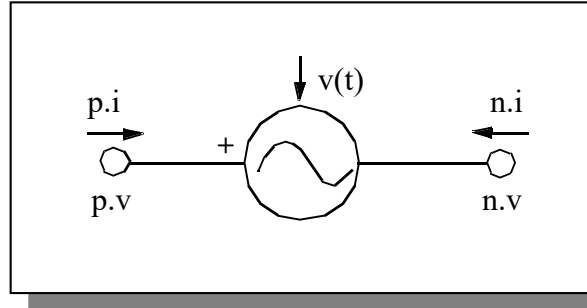
```
model Inductor "Ideal electrical inductor"  
  extends TwoPin;  
  parameter Real L "Inductance";  
equation  
   $L*der(i) = v$ ;  
end Inductor;
```

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

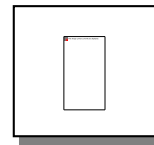


Electrical Components cont'

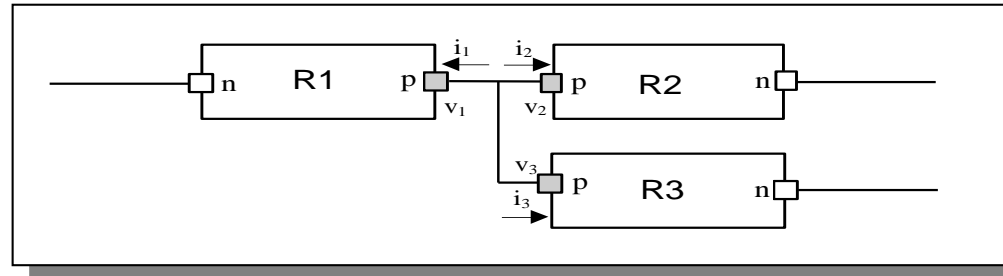
```
model Source
  extends TwoPin;
  parameter Real A,w;
equation
  v = A*sin(w*time);
end Resistor;
```



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



Resistor Circuit



```
model ResistorCircuit
  Resistor R1 (R=100);
  Resistor R2 (R=200);
  Resistor R3 (R=300);
equation
  connect (R1.p, R2.p);
  connect (R1.p, R3.p);
end ResistorCircuit;
```

Corresponds to

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

Modelica Standard Library - Graphical Modeling

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

Modelica Standard Library cont'

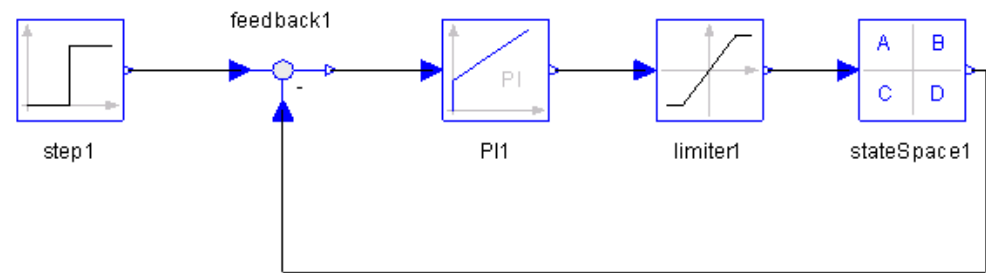
The Modelica Standard Library contains components from various application areas, including the following sublibraries:

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

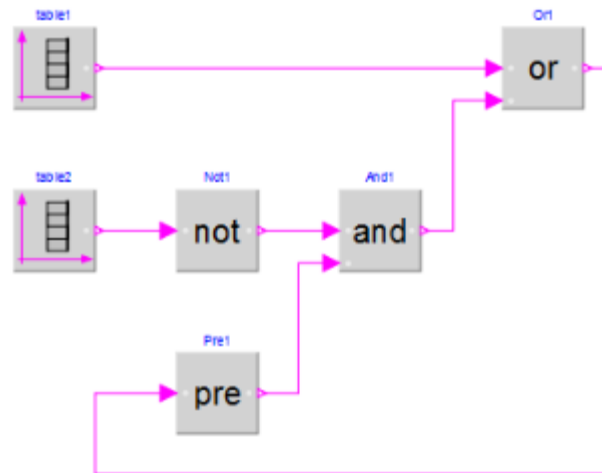
Modelica.Blocks

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

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

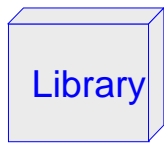


Examples:

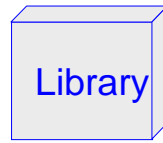


Modelica.Electrical

Electrical components for building analog, digital, and multiphase circuits



Analog



Digital

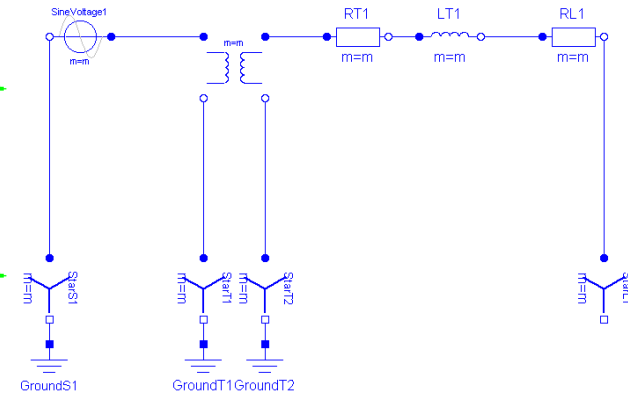
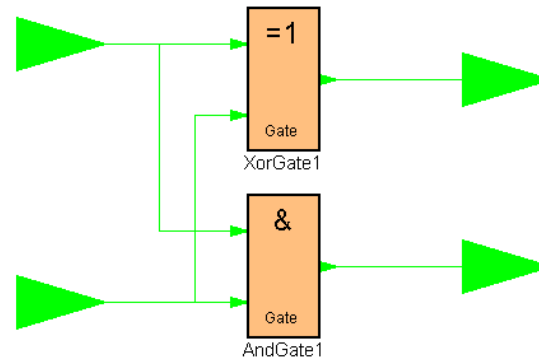
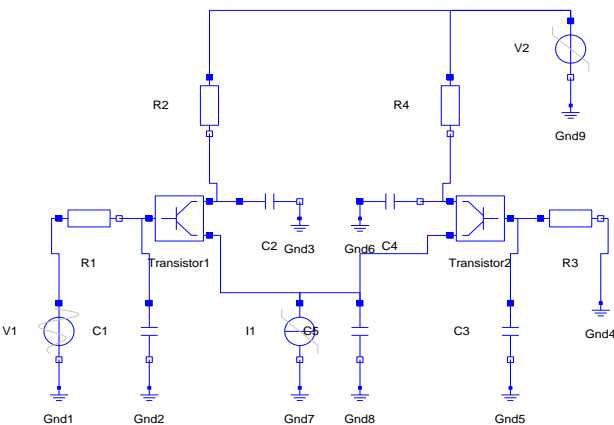


Machines



MultiPhase

Examples:

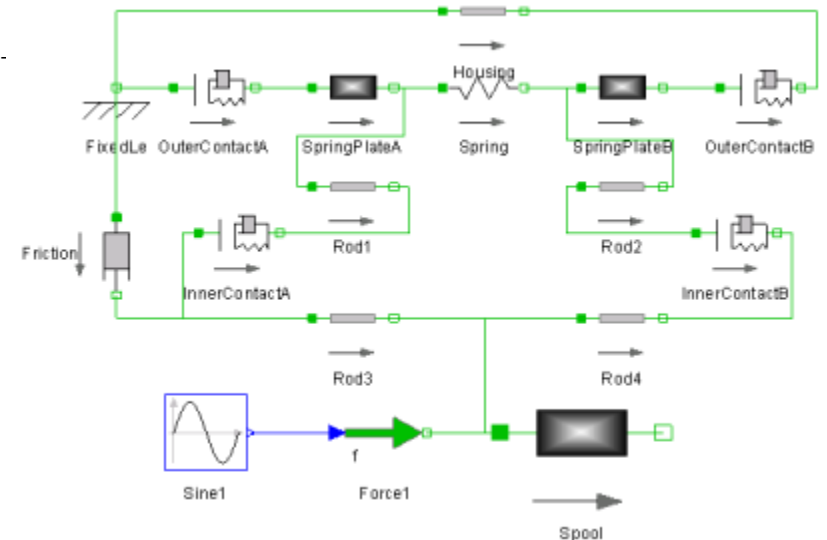
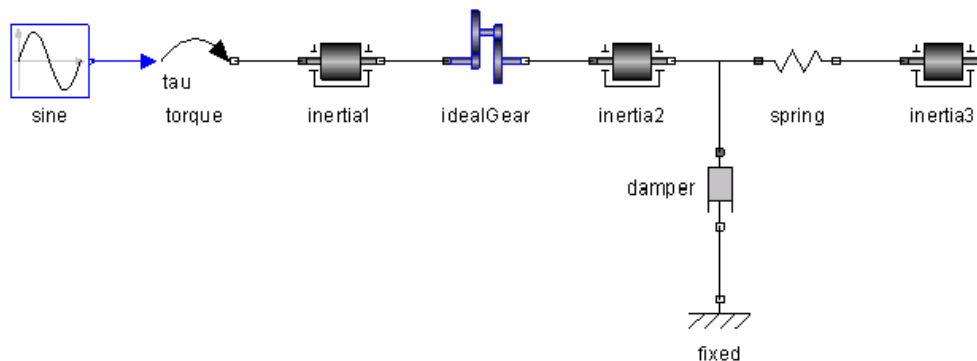


Modelica.Mechanics

Package containing components for mechanical systems

Subpackages:

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



Other Free Libraries

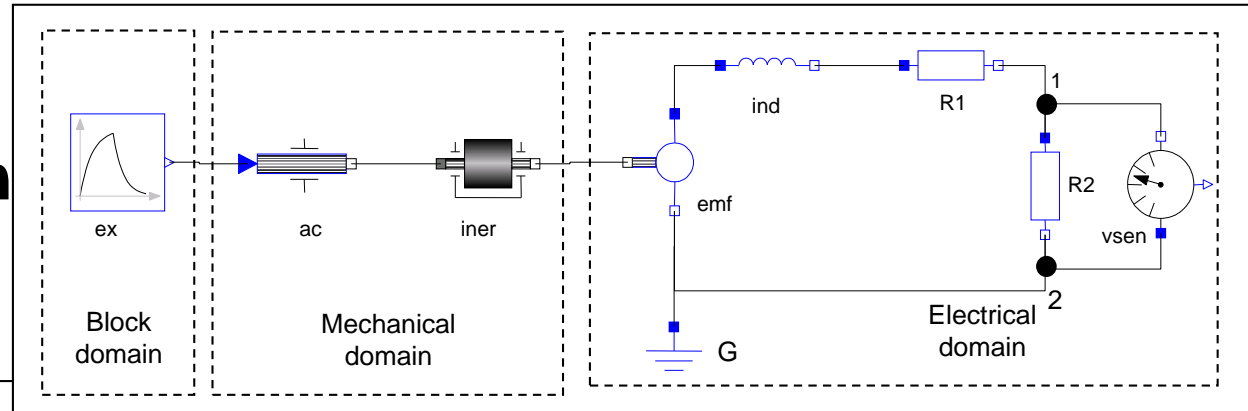
- WasteWater Wastewater treatment plants, 2003
- ATPlus Building simulation and control (fuzzy control included), 2005
- MotorCycleDynamics Dynamics and control of motorcycles, 2009
- NeuralNetwork Neural network mathematical models, 2006
- VehicleDynamics Dynamics of vehicle chassis (obsolete), 2003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

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

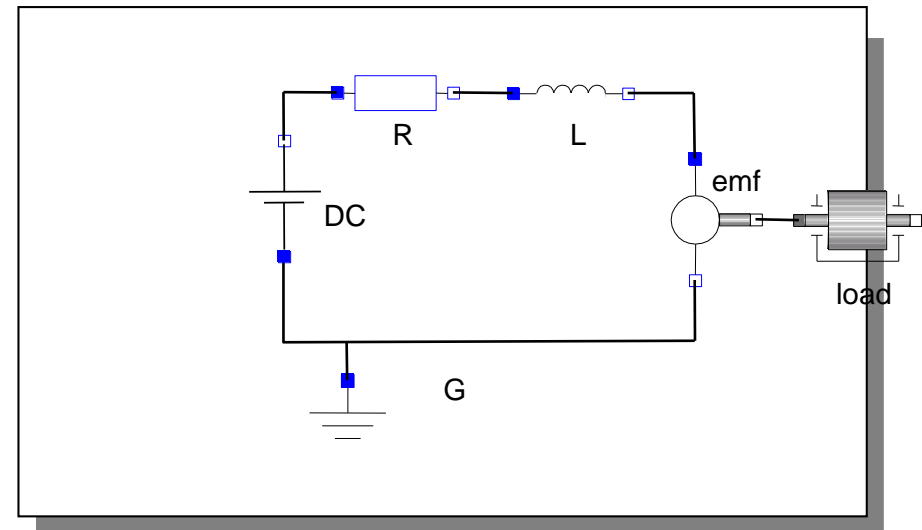
```
connect(ac.flange_b, iner.flange_a); connect(iner.flange_b, emf.flange_b);  
connect(emf.p, ind.p); connect(ind.n, R1.p); connect(emf.n, G.p);  
connect(emf.n, R2.n); connect(R1.n, R2.p); connect(R2.p, vsens.n);  
connect(R2.n, vsens.p); connect(ex.outPort, ac.inPort);
```

```
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;
```

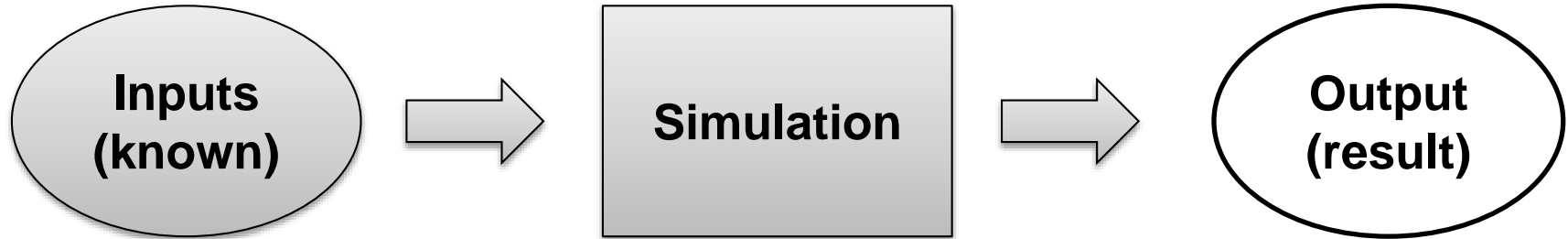


Part V
Dynamic Optimization
Theory and Exercises

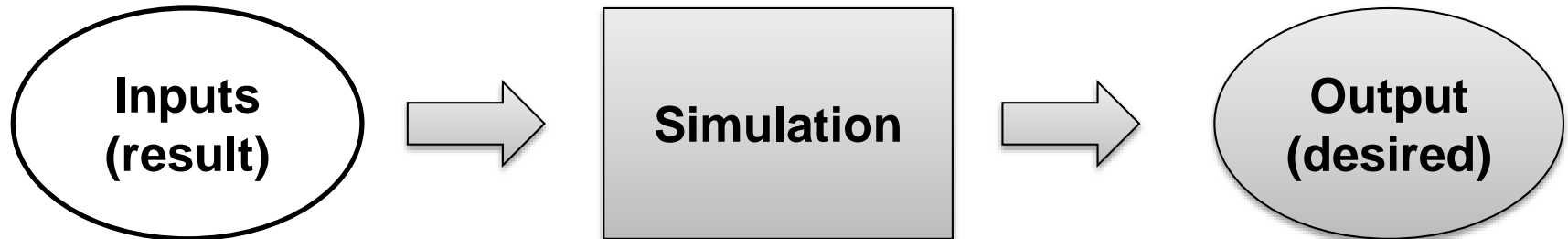
using
OpenModelica

Built-in Dynamic Optimization - Motivation

Simulation



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



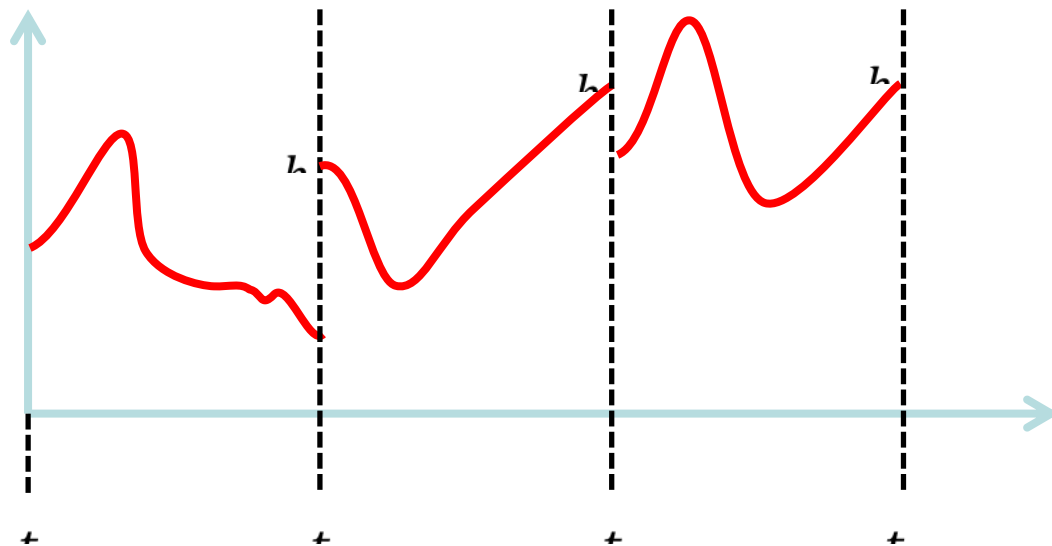
Optimization of Dynamic Trajectories Using Multiple-Shooting and Collocation

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

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

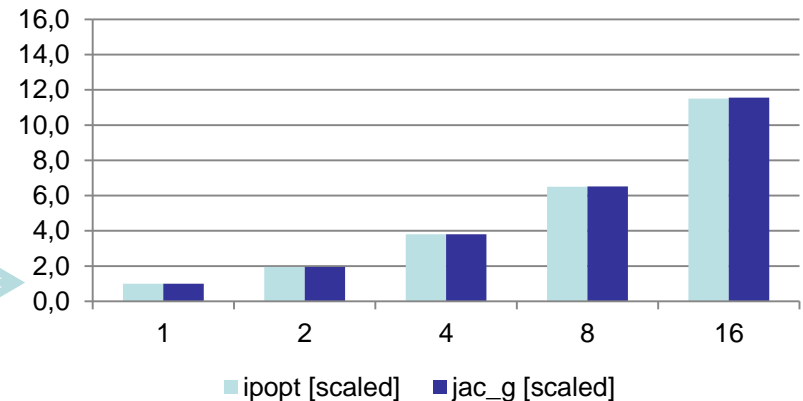
This approach uses a single optimization run and is different from classical parameter sweep optimization typically using a large number of simulations

$$x_i(t_i) = h_i$$



Example speedup, 16 cores:

MULTIPLE_COLLOCATION



Optimal Control Problem (OCP)

Cost function

$$\min_{u(t)} J(x(t), u(t), t) = \underbrace{E(x(t_f), u(t_f), t_f)}_{\text{Mayer-Term}} + \int_{t_0}^{t_f} \underbrace{L(x(t), u(t), t)}_{\text{Lagrange-Term}} dt \quad (1)$$

Subject to

Initial conditions

$$x(t_0) = x_0 \quad (2)$$

Nonlinear dynamic model

$$\dot{x} = f(x(t), u(t), t) \quad (3)$$

Path constraints

$$\hat{g}(x(t), u(t), t) \leq 0 \quad (4)$$

Terminal constraints

$$r(x(t_f)) = 0 \quad (5)$$

where

$x(t) = [x^1(t), \dots, x^{n_x}(t)]^T$ is the state vector and

$u(t) = [u^1(t), \dots, u^{n_u}(t)]^T$ is the control variable vector for

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

OCP Formulation in OpenModelica

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

$$\begin{aligned}x_{\min} &\leq x(t) \leq x_{\max} \\u_{\min} &\leq u(t) \leq u_{\max}\end{aligned}$$

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

	Annotation
Mayer-Term	<code>Real costM annotation(isMayer=true);</code>
Lagrange-Term	<code>Real costL annotation(isLagrange=true);</code>
Constraints	<code>Real x(max=0) annotation(isConstraint=true);</code>
Final constraints	<code>Real y(min=0) annotation(isFinalConstraint=true);</code>

Predator-Prey Example – The Forest Model

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

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

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

$$\text{IC: } x_r(t_0) = 700, \quad x_f(t_0) = 10$$

where

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

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

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

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

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

$$d_{fh} = 9 \cdot 10^{-2}, \text{ Death rate of foxes due to hunters}$$

Predator-Prey Example – Modelica model

```
model Forest "Predator-prey model"  
  parameter Real g_r = 4e-2 "Natural growth rate for rabbits";  
  parameter Real g_fr = 1e-1 "Efficiency in growing foxes from rabbits";  
  parameter Real d_rf = 5e-3 "Death rate of rabbits due to foxes";  
  parameter Real d_rh = 5e-2 "Death rate of rabbits due to hunters";  
  parameter Real d_f = 9e-2 "Natural deathrate for foxes";  
  parameter Real d_fh = 9e-2 "Death rate of foxes due to hunters";  
  Real x_r(start=700,fixed=true) "Rabbits with start population of 700";  
  Real x_f(start=10,fixed=true) "Foxes with start population of 10";  
  input Real u_hr "Rabbit hunters";  
  input Real u_hf "Fox hunters";  
equation  
  der(x_r) = g_r*x_r - d_rf*x_r*x_f - d_rh*u_hr;  
  der(x_f) = g_fr*d_rf*x_r*x_f - d_f*x_f - d_fh*u_hf;  
end Forest;
```

Control
variables



Predator-Prey Example – Optimal Control Problem

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

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

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

Modelica model:

Extension of the
system model

constraint

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

Cost function
Mayer-term

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

Predator-Prey Example – Using OMNotebook

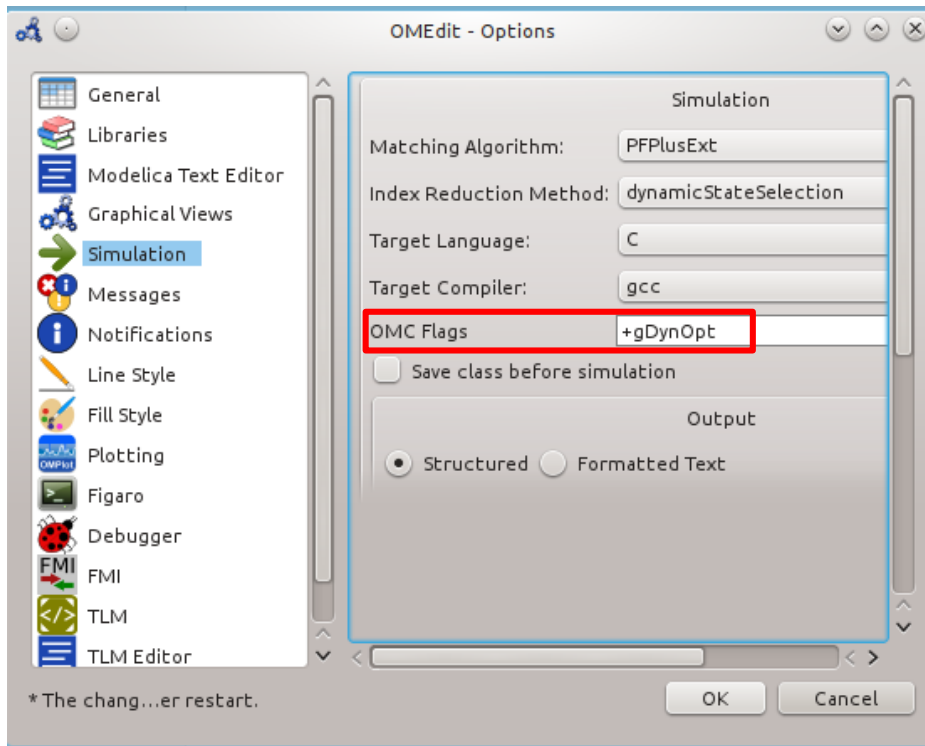
Start the optimization from OMNotebook using a time interval $[t_0, t_f] = [0, 400]$ seconds

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

Option	Example value	Description
numberOfIntervals	50	collocation intervals
startTime, stopTime	0, 400	time horizon in seconds
tolerance	1e-8	solver/optimizer tolerance
simflags	...	see documentation for details

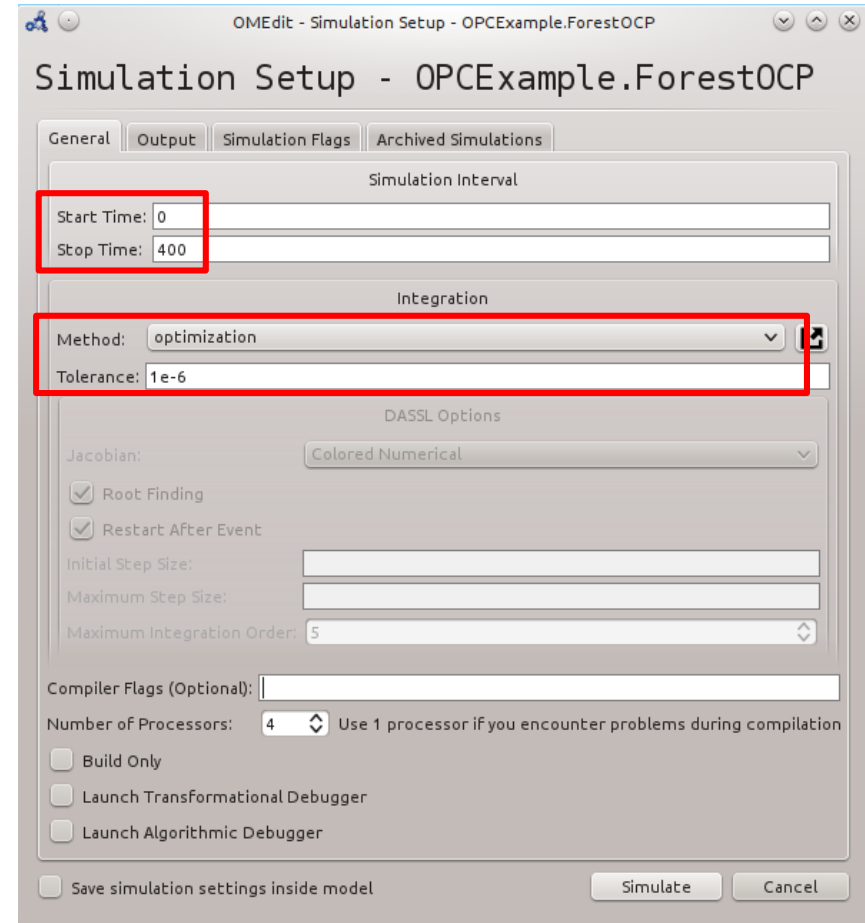
Predator-Prey Example – Using OMEdit

Tools → Options → Simulation



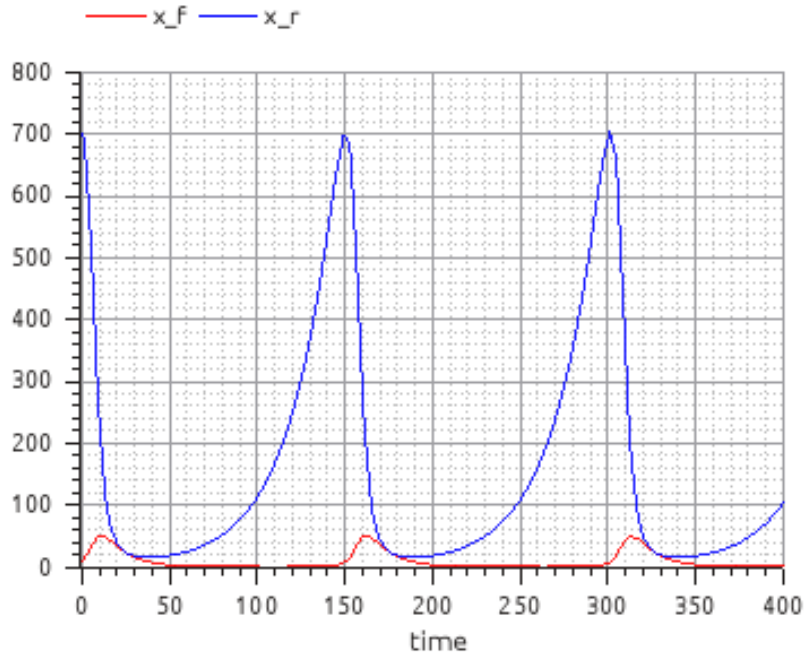
+gDynOpt

Simulation → Simulation Setup

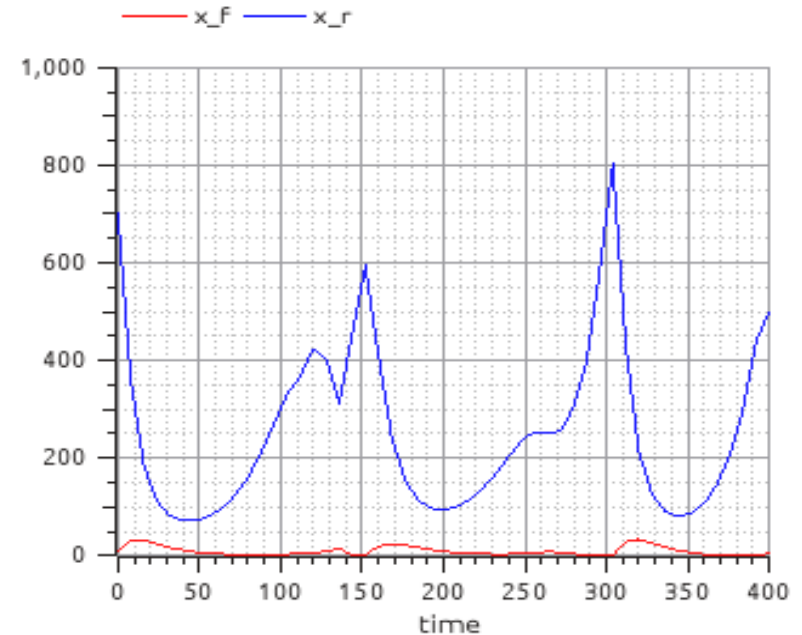


optimization

Predator-Prey Example – Plots



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



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

$$x_r(t_f) = 500, x_f(t_f) = 5$$

Exercise – Optimal Control

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

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

Part Vb
More
Graphical Modeling Exercises

using
OpenModelica

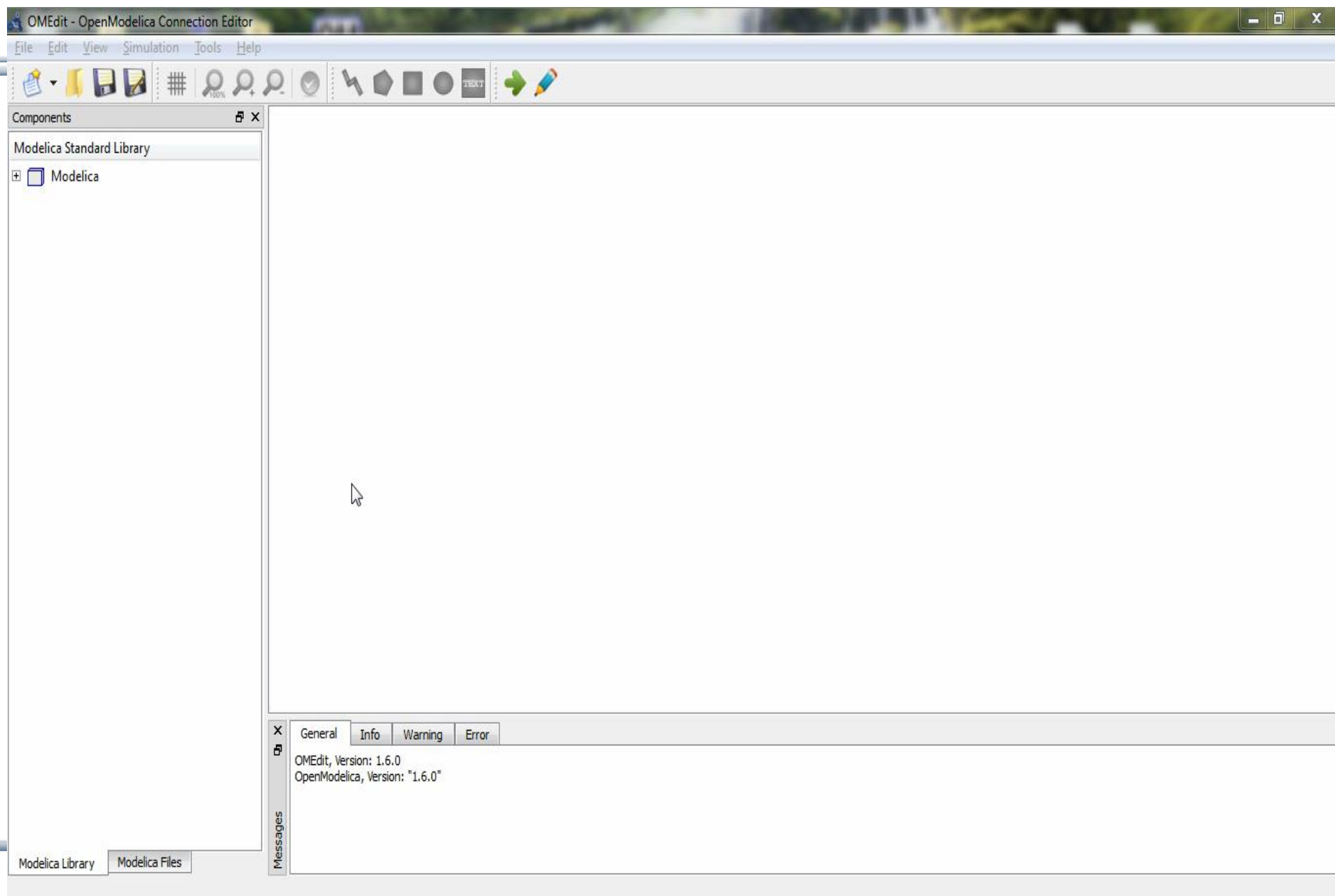
Graphical Modeling - Using Drag and Drop Composition

The screenshot displays the OMEdit - OpenModelica Connection Editor - [DCMotor*] window. The interface is divided into several panes:

- Libraries Browser:** Located on the left, it shows a tree view of libraries including Complex, Modelica, UsersGuide, Blocks, ComplexBlocks, StateGraph, Electrical, Magnetic, Mechanics, MultiBody, Rotational, and Components. The Rotational library is expanded, showing sub-components like Fixed, Inertia, Disc, Spring, Damper, and SpringDamper.
- Diagram View:** The central area shows a circuit diagram. It starts with a step voltage source (step1) with start time 0. This is connected to a resistor (resistor1) with resistance R=R, followed by an inductor (inductor1) with inductance L=L. The circuit then connects to a motor block (emf) and an inertia block (inertia1) with moment of inertia J=J. The motor block is connected to ground (ground1).
- Variables Browser:** Located on the right, it shows a table of variables and their values. The DCMotor block is expanded, showing variables like emf, der(phi), fixed, flange, i, internalSupport, k, n, phi, useSupport, v, w, ground1, and inductor1. The values are as follows:

Variable	Value	Unit
emf		
der(phi)	-0.3403	deg
fixed		
flange		
i	-0.533507	A
internalSupport		
k	1.0	N.m/
n		
phi		deg
useSupport	0	
v	-0.3403	V
w	-0.3403	rad/s
ground1		
inductor1		

Graphical Modeling Animation – DCMotor



Multi-Domain (Electro-Mechanical) Modelica Model

- A DC motor can be thought of as an electrical circuit which also contains an electromechanical component

model DCMotor

```
Resistor R(R=100);
```

```
Inductor L(L=100);
```

```
VsourceDC DC(f=10);
```

```
Ground G;
```

```
ElectroMechanicalElement EM(k=10, J=10, b=2);
```

```
Inertia load;
```

equation

```
connect (DC.p, R.n);
```

```
connect (R.p, L.n);
```

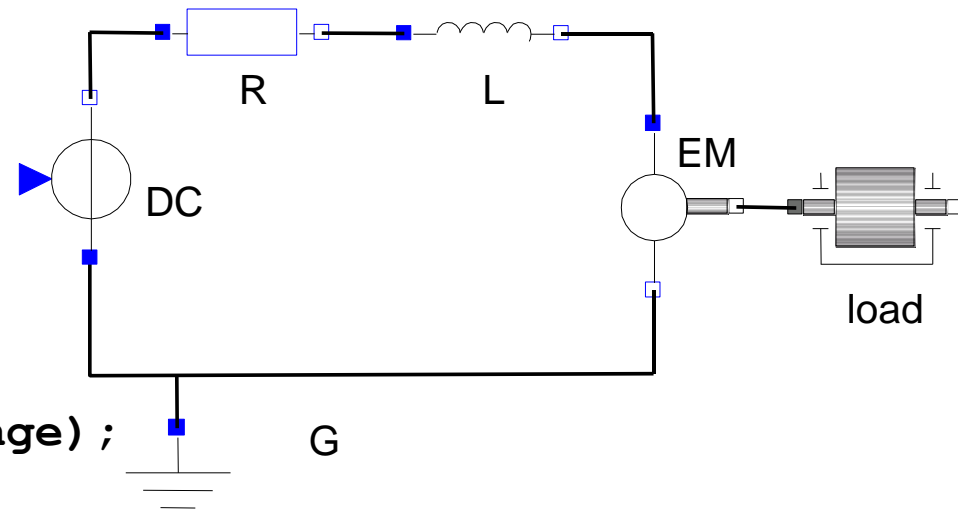
```
connect (L.p, EM.n);
```

```
connect (EM.p, DC.n);
```

```
connect (DC.n, G.p);
```

```
connect (EM.flange, load.flange);
```

end DCMotor



Corresponding DCMotor Model Equations

The following equations are automatically derived from the Modelica model:

$0 == DC.p.i + R.n.i$	$EM.u == EM.p.v - EM.n.v$	$R.u == R.p.v - R.n.v$
$DC.p.v == R.n.v$	$0 == EM.p.i + EM.n.i$	$0 == R.p.i + R.n.i$
	$EM.i == EM.p.i$	$R.i == R.p.i$
$0 == R.p.i + L.n.i$	$EM.u == EM.k * EM.\omega$	$R.u == R.R * R.i$
$R.p.v == L.n.v$	$EM.i == EM.M / EM.k$	
	$EM.J * EM.\omega == EM.M - EM.b * EM.\omega$	$L.u == L.p.v - L.n.v$
$0 == L.p.i + EM.n.i$		$0 == L.p.i + L.n.i$
$L.p.v == EM.n.v$	$DC.u == DC.p.v - DC.n.v$	$L.i == L.p.i$
	$0 == DC.p.i + DC.n.i$	$L.u == L.L * L.i'$
$0 == EM.p.i + DC.n.i$	$DC.i == DC.p.i$	
$EM.p.v == DC.n.v$	$DC.u == DC.Amp * Sin[2 \pi DC.f * t]$	
$0 == DC.n.i + G.p.i$		
$DC.n.v == G.p.v$		

(load component not included)

Automatic transformation to ODE or DAE for simulation:

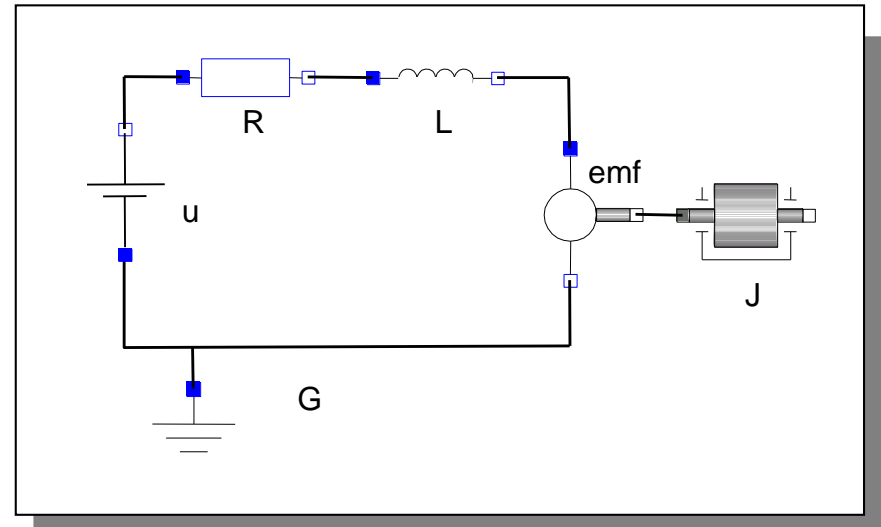
$$\frac{dx}{dt} == f[x, u, t] \quad g\left[\frac{dx}{dt}, x, u, t\right] == 0$$

Exercise 3.1

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

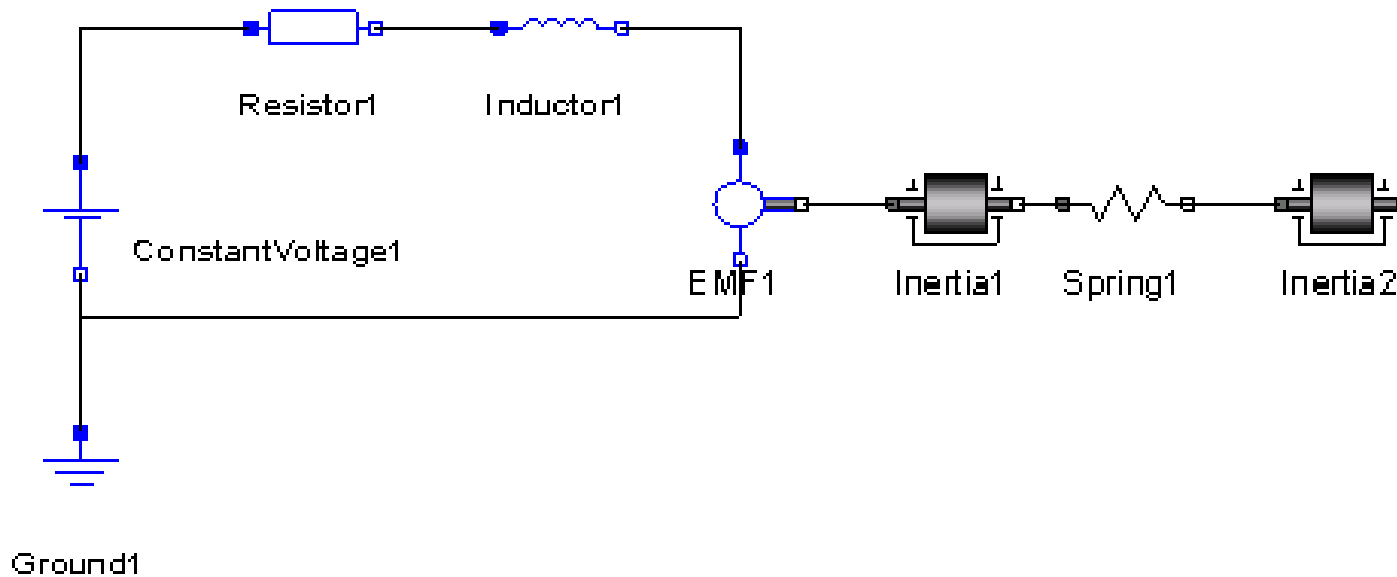
`Mechanics.Rotational.Components,`
`Electrical.Analog.Basic,`
`Electrical.Analog.Sources`

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



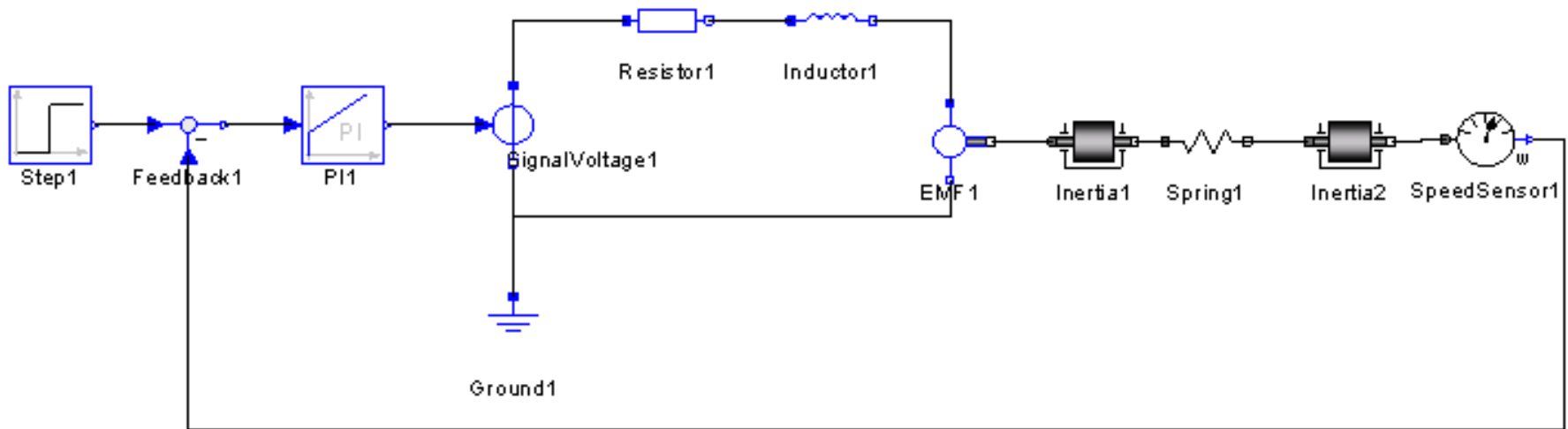
Exercise 3.2

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



Exercise 3.3

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



Exercise 3.4 – DrControl

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

Feedback

One important method in designing control systems is a feedback loop. It can be used to eliminate the influence of noise or to decrease the output error.

1 Example

Assume that we want to control the speed of a car on the road. The car has a mass m , velocity y , and aerodynamic coefficient a . The θ is the road slope, which in this case can be regarded as noise.

$$m\dot{y} = u - ay - mg \sin(\theta)$$

If we want a reference speed of 20 m/s for a car with $m=1500$ kg, $a=250$ N/s, $\theta=0$ rad, how high should the amplification factor be in the regulator? Try with $u = 250^*$.

1.1 Open Loop

```
loadModel(Modelica)
true
model nofeedback
import SI = Modelica.SIunits;
SI.Velocity y;
noise, theta = 0 -> v(t) = 0
SI.Velocity vnoise;
// output signal without
// output signal with noise
```

Transfer Function

It is sometimes practical to study the Laplace transform of the involved quantities.

$$Y(s) = \mathcal{L}\{y(t)\}(s) = \int_0^{\infty} y(t)e^{-st} dt$$

The obvious reason to why the transformed quantities are preferred is that the derivative of $y(t)$ if the initial condition $y(0)$ is zero. Now by assuming the initial value of the signal its derivative initial values are all zero and then Laplace-transforming the differential equation

$$\frac{d^n}{dt^n} y(t) + a_1 \frac{d^{n-1}}{dt^{n-1}} y(t) + \dots + a_n y(t) = b_0 \frac{d^m}{dt^m} u(t) + \dots + b_{m-1} \frac{d}{dt} u(t) + b_m u(t)$$

leads to

$$(s^n + a_1 s^{n-1} + \dots + a_n)Y(s) = (b_0 s^m + \dots + b_{m-1} s + b_m)U(s)$$

Now the differential equation with all its derivative terms is described with the help of a polynomial

$$Y(s) = \frac{(b_0 s^m + \dots + b_{m-1} s + b_m)}{(s^n + a_1 s^{n-1} + \dots + a_n)} U(s) = G(s)U(s)$$

This polynomial fraction is called the transfer function of the system, denoted as $G(s)$. The transfer function are the roots of the denominator which is the same as the roots to the char. The zeros are the roots of the numerator of the transfer function. The inverse Laplace transform called the weight function and is the impulse response of the system. In Modelica the model is formalized in a state space (differential) form. Therefore the initial conditions are important right result.

State Space Form

A state of a system is the amount of information needed for determining the future output of the system if the future inputs are known. The state space form for continuous time dependent systems can be expressed as a system of first order differential equations.

1 Linear Differential Equations in State Space Form

Higher order differential equations can be treated more practically if transferred to first order differential equations.

$$\dot{y}(t) + a_1 y(t) + a_2 y(t) = bu(t)$$

By introducing new variables, x_1 and x_2

$$\begin{cases} \dot{x}_1(t) = y(t) \\ \dot{x}_2(t) = y(t) \end{cases}$$

the differential equation can be represented as

$$\begin{cases} \dot{x}_1(t) = y(t) = x_2(t) \\ \dot{x}_2(t) = y(t) = -a_1 y(t) - a_2 y(t) + bu(t) = -a_1 x_2(t) - a_2 x_1(t) + bu(t) \end{cases}$$

With matrix notation the equations takes the form:

$$\begin{pmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ -a_2 & -a_1 \end{pmatrix} \begin{pmatrix} x_1(t) \\ x_2(t) \end{pmatrix} + \begin{pmatrix} 0 \\ b \end{pmatrix} u(t)$$

Now let

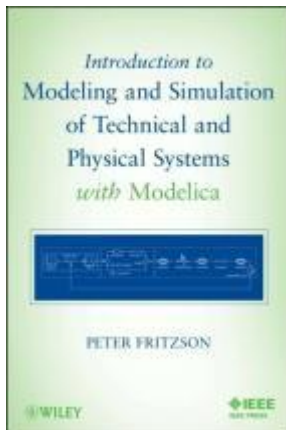
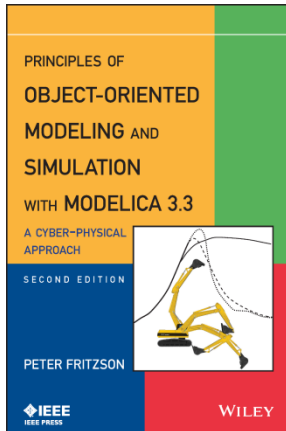
$$A = \begin{pmatrix} 0 & 1 \\ -a_2 & -a_1 \end{pmatrix} \\ B = \begin{pmatrix} 0 \\ b \end{pmatrix} \\ C = (1 \quad 0)$$

then the higher order differential equation can be written in a more compact form:

$$\begin{cases} \dot{x}(t) = Ax(t) + Bu(t) \\ y(t) = Cx(t) \end{cases}$$

2 Transferfunction to State Space Form

Learn more...

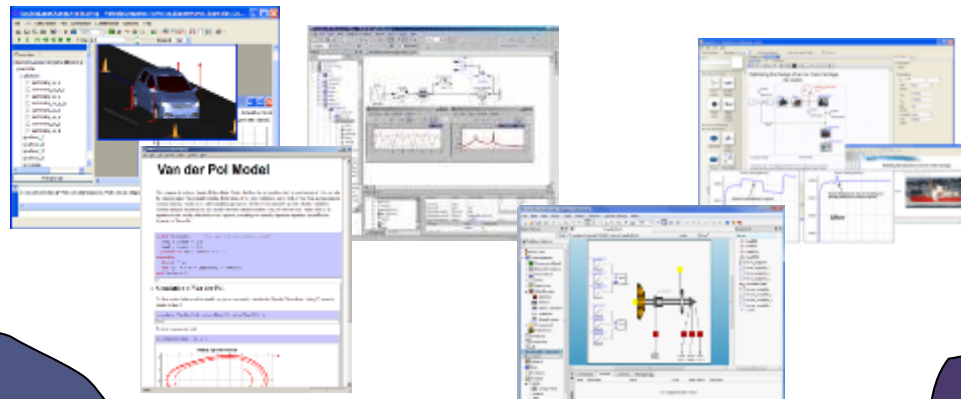


- OpenModelica
 - www.openmodelica.org
- Modelica Association
 - www.modelica.org
- Books
 - Principles of Object Oriented Modeling and Simulation with Modelica 3.3: A Cyber-Physical Approach, Peter Fritzson 2015.
 - Modeling and Simulation of Technical and Physical Systems with Modelica. Peter Fritzson., 2011
<http://eu.wiley.com/WileyCDA/WileyTitle/productCd-111801068X.html>
 - Introduction to Modelica, Michael Tiller

Summary

Multi-Domain
Modeling

Visual Acausal
Component
Modeling



Typed
Declarative
Textual Language

Thanks for listening!

Hybrid
Modeling