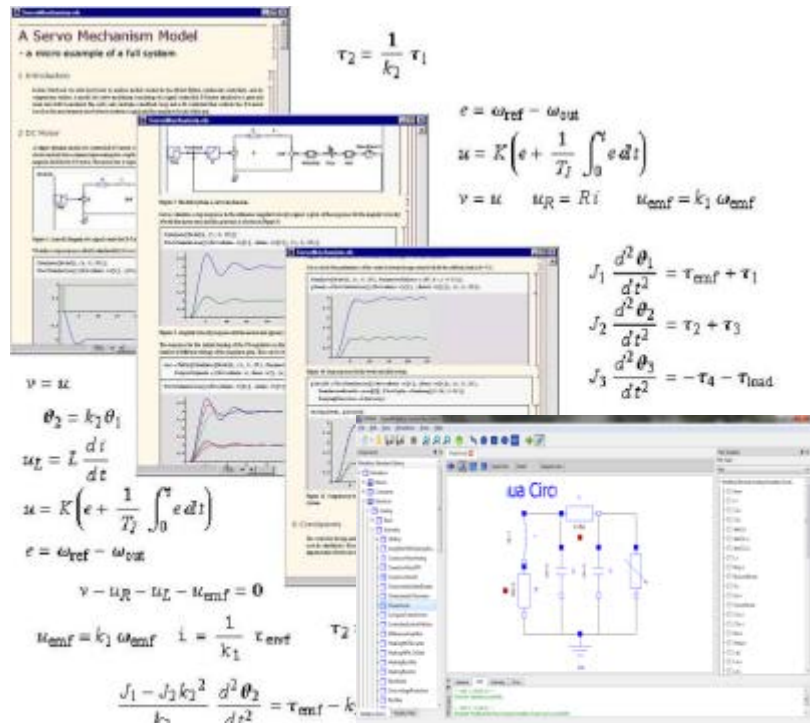


# Introduction to Object-Oriented Modeling and Simulation with Modelica and OpenModelica



**Tutorial 2022-10-28 Modelica US 2022**

**John Tinnerholm**

Linköping University, [john.tinnerholm@liu.se](mailto:john.tinnerholm@liu.se)

PhD Student

**Peter Fritzson**

Professor em. at Linköping University, [peter.fritzson@liu.se](mailto:peter.fritzson@liu.se)

Research Director at Programming Environments Lab

Vice Director of the Open Source Modelica Consortium

Vice Director of the MODPROD Center for Model-based Development

**Adrian Pop**

Linköping University, [adrian.pop@liu.se](mailto:adrian.pop@liu.se)

Technical Coordinator of the Open Source Modelica Consortium

**Slides**

Based on book and lecture notes by Peter Fritzson

Contributions 2004-2005 by Emma Larsdotter Nilsson, Peter Bunus

Contributions 2006-2018 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 2012-2018 by Peter Fritzson, Lena Buffoni, Mahder

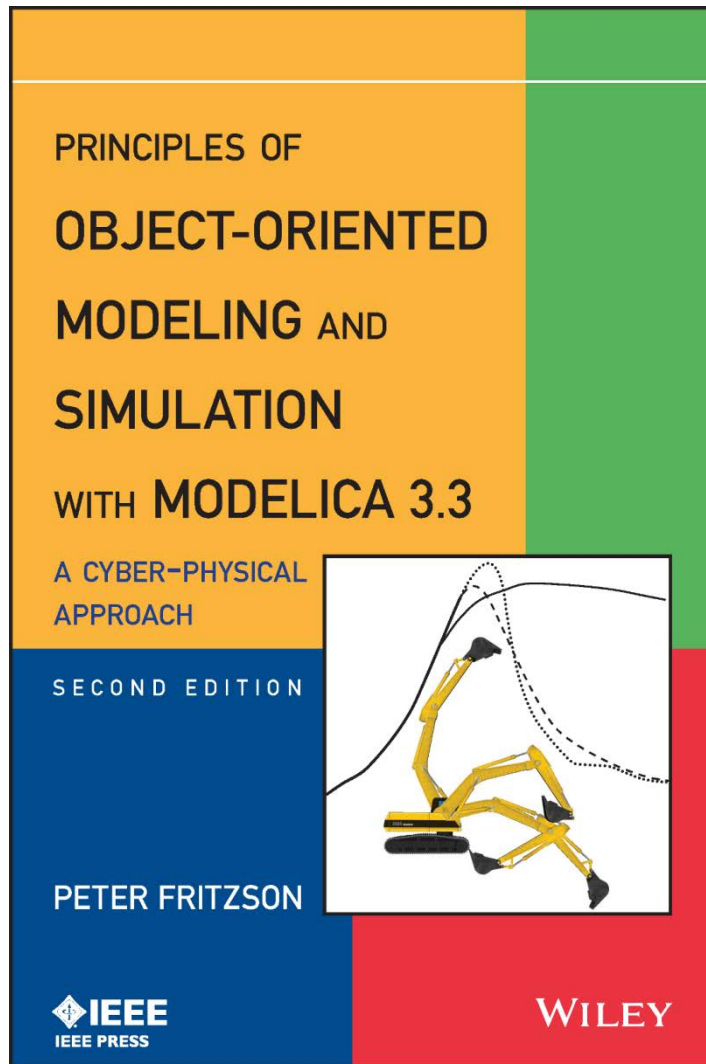
Gebremedhin, Bernhard Thiele, Lennart Ochel

Contributions 2019-2022 by Peter Fritzson, Arunkumar Palanisamy, Bernt

Lie, Adrian Pop

# 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](http://www.openmodelica.org)
- Modelica Association
  - [www.modelica.org](http://www.modelica.org)



# Introductory Modelica Book

September 2011  
232 pages

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@liu.se](mailto:peter.fritzson@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 3.3", 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](http://www.modelica.org)
- OpenModelica: [www.openmodelica.org](http://www.openmodelica.org)

# Outline

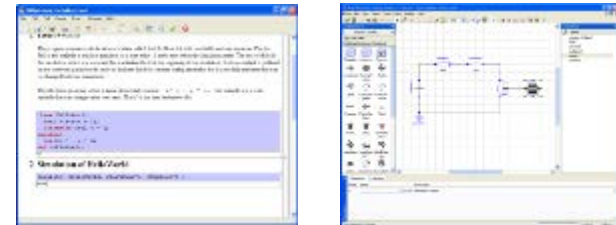
## Part I

### Introduction to Modelica and a demo example



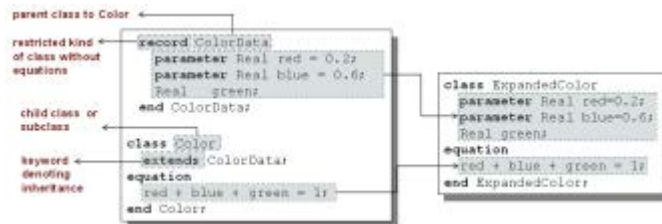
## Part II

### Modelica environments



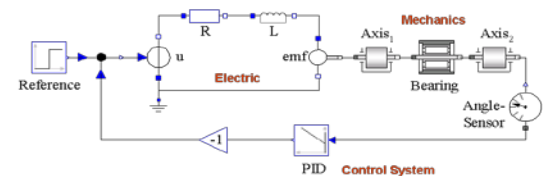
## Part III

### Modelica language concepts and textual modeling



## Part IV

### Graphical modeling and the Modelica standard library



# Software Installation - Windows

---

- Start the software installation
- Install OpenModelica-1.18.0 or later Download from [www.openmodelica.org](http://www.openmodelica.org)  
(takes about 20min)

# Software Installation – Linux (requires internet connection)

---

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



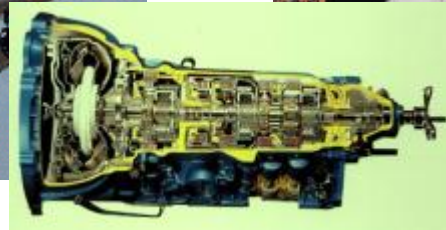
# Software Installation – MAC (requires internet connection)

---

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

# Part I

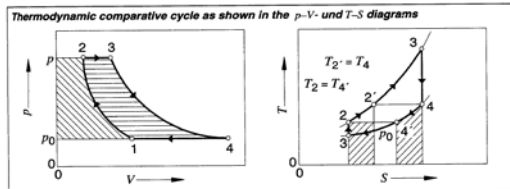
## Introduction to Modelica and a demo example



# 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:

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

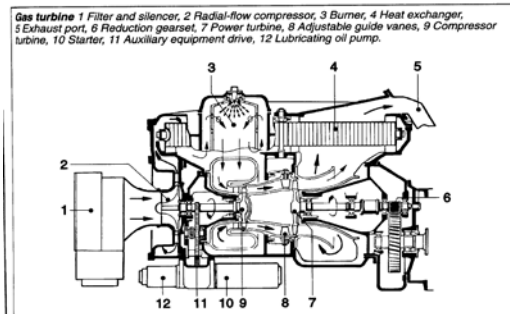
Where  $p_2/p_1 = (T_2/T_1)^{\frac{\gamma}{\gamma-1}} = (T_3/T_4)^{\frac{\gamma}{\gamma-1}}$  and  $T_4 = T_3 \cdot (T_1/T_2)^{\frac{\gamma-1}{\gamma}}$  thus

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

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

**Advantages of the gas turbine:** 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.



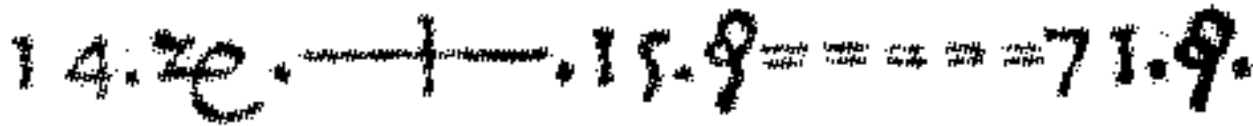
*“The change of motion is proportional to the motive force impressed”*  
– Newton

Lex. II.

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

# Modelica Background: The Form – Equations

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



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

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

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

variable = expression

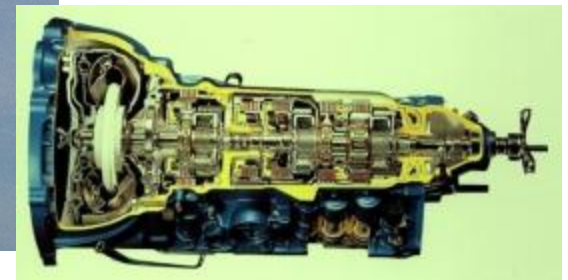
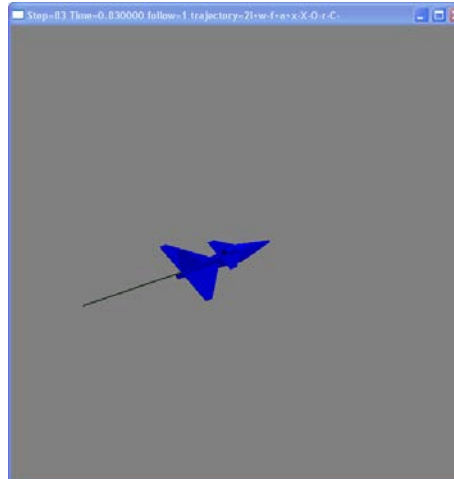
$v = \text{INTEG}(F) / m$

**Programming languages usually do not allow equations!**

# What is Modelica?

A language for modeling of complex cyber-physical systems

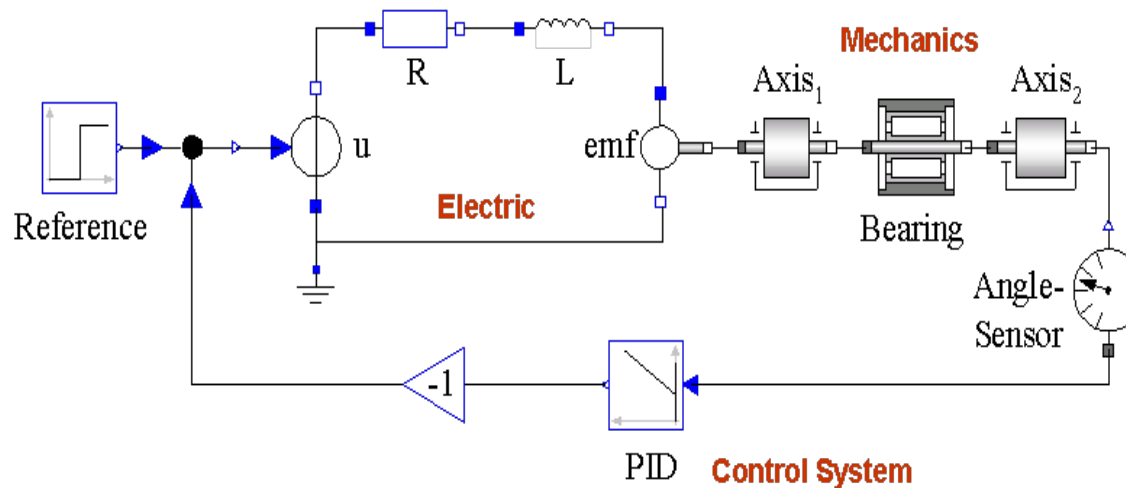
- Robotics
- Automotive
- Aircrafts
- Satellites
- Power plants
- Systems biology





# 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 one free and several commercial tools, for example:**

- **OpenModelica from OSMC**  
(in ABB Optimax, Bosch-Rexr Control Edge Designer, Mike DHI)
- Dymola from Dassault systems
- Wolfram System Modeler from Wolfram MathCore
- SimulationX from ITI, part of ESI Group
- MapleSim from MapleSoft  
(also in Altair solidThinking Activate)
- AMESIM from LMS
- Impact from Modelon  
(also in ANSYS Simpler, Rickardo tool, etc.)
- MWORKS from Tongyang Sw & Control
- IDA Simulation Env, from Equa

Available at: [www.modelica.org](http://www.modelica.org)

*Developed and standardized  
by Modelica Association*

# Modelica – The Next Generation Modeling Language

## Declarative statically typed language

Equations and mathematical functions allow acausal modeling,  
high level specification and static type checking for increased correctness

## Multi-domain modeling

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

## Everything is a class

Safe engineering practices by statically typed object-oriented language,  
general class concept, Java & MATLAB-like syntax

## Visual component programming

Hierarchical system architecture capabilities

## Efficient, non-proprietary

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

# Modelica Acausal Modeling

What is *acausal* modeling/design?

Why does it increase *reuse*?

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

Example: a resistor *equation*:

$$R * i = v;$$

can be used in three ways:

$$i := v/R;$$

$$v := R*i;$$

$$R := v/i;$$

# What is Special about Modelica?

---

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

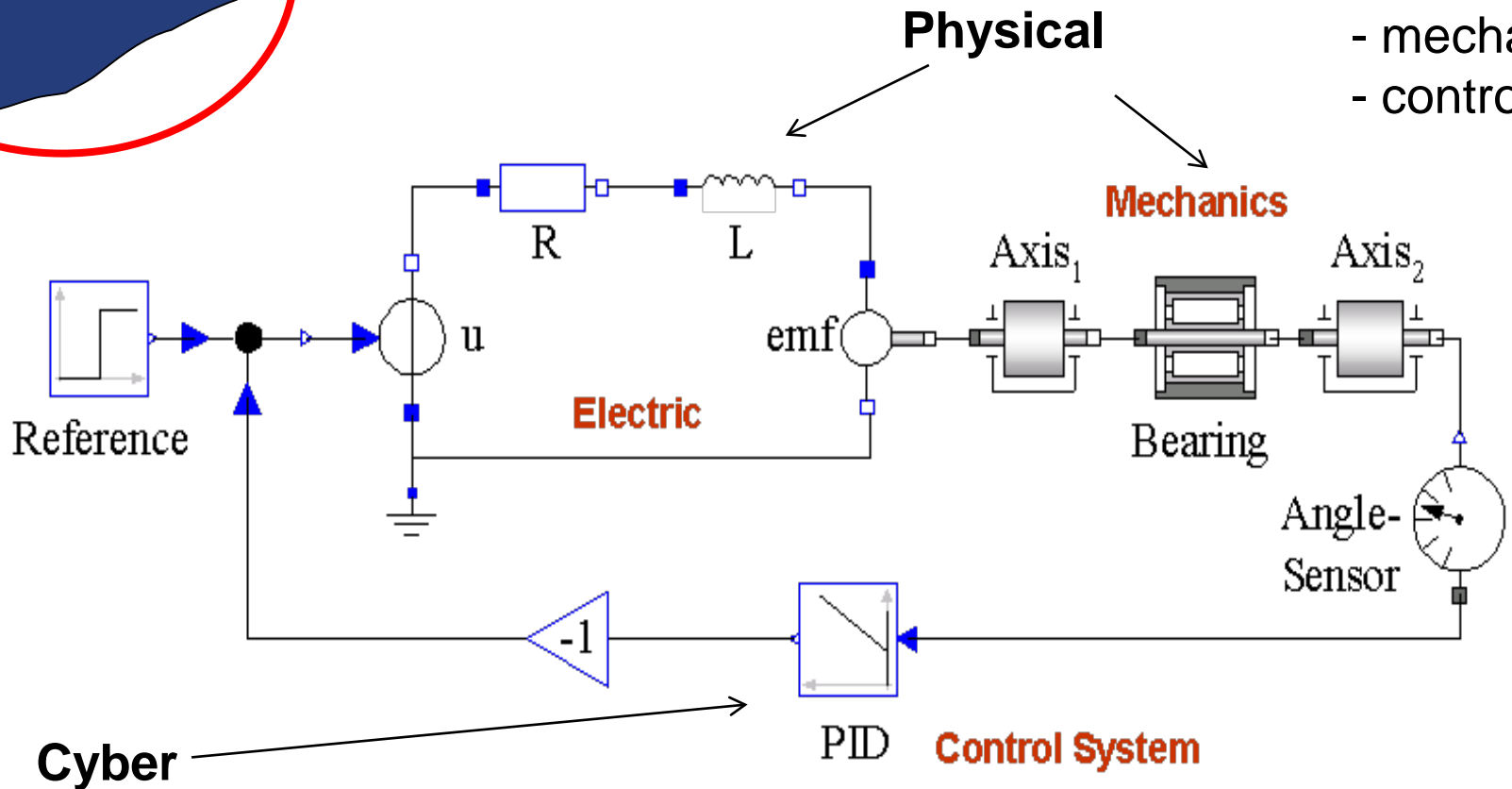


# What is Special about Modelica?

Multi-Domain  
Modeling

## Cyber-Physical Modeling

3 domains  
- electric  
- mechanics  
- control



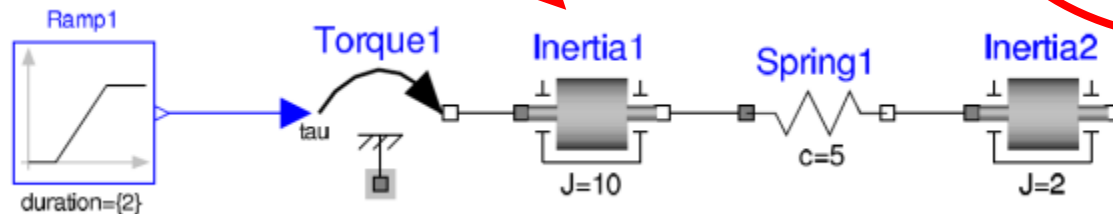
# What is Special about Modelica?

Multi-Domain  
Modeling

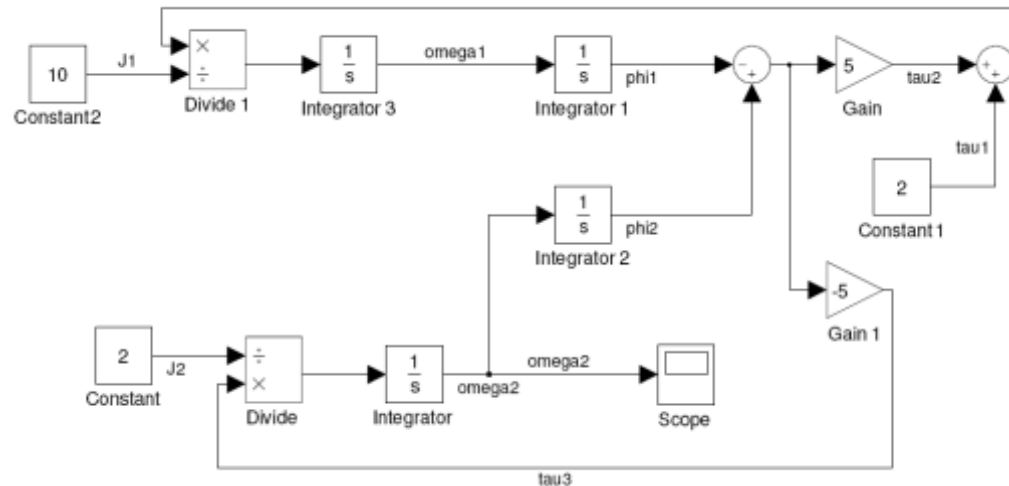
Keeps the physical  
structure

Visual Acausal  
Hierarchical  
Component  
Modeling

Acausal model  
(Modelica)



Causal  
block-based  
model  
(Simulink)

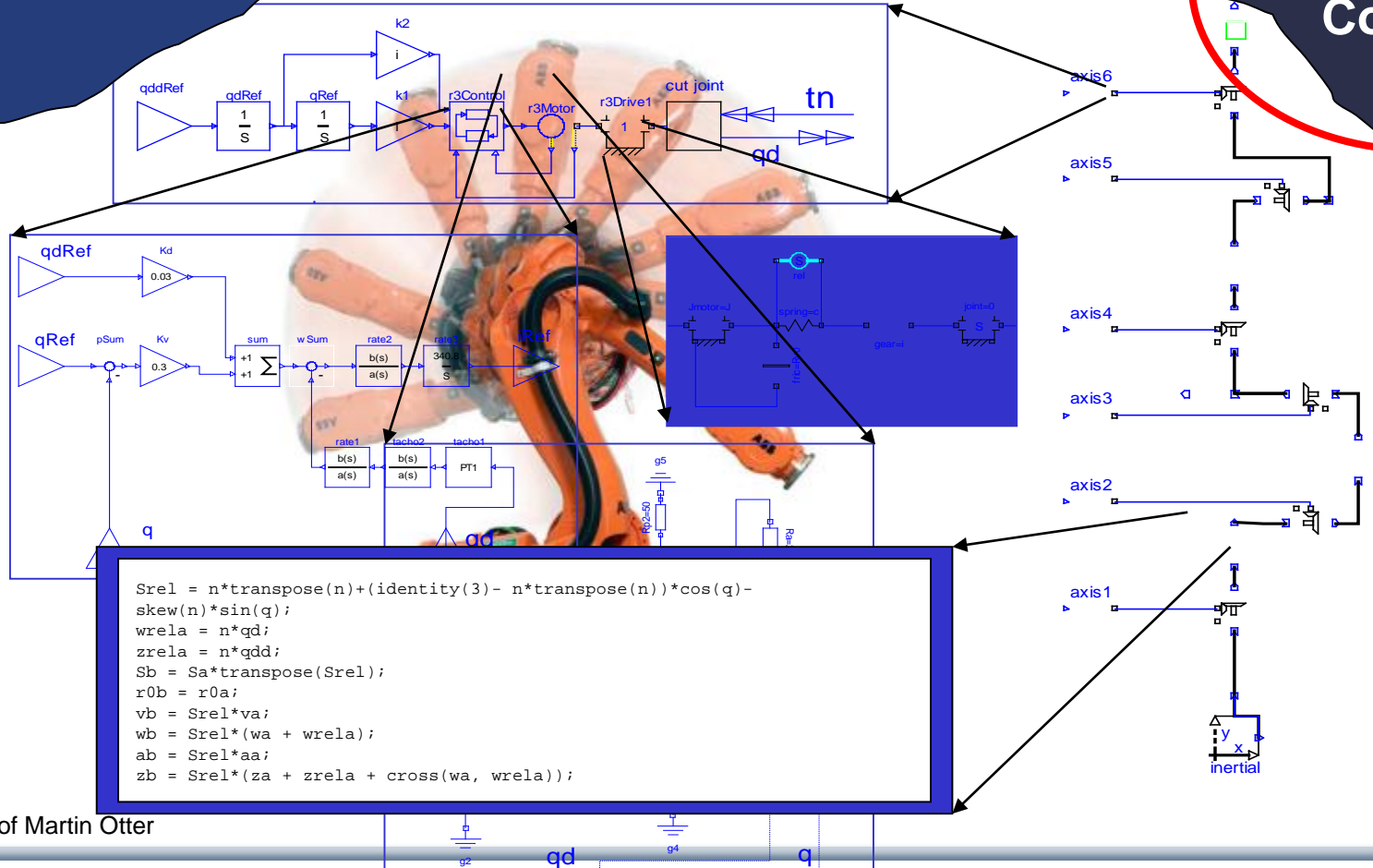


# What is Special about Modelica?

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

## Behavior described declaratively using

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

Variable  
declarations

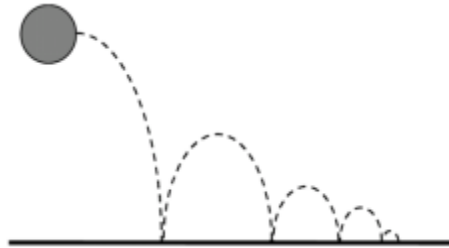
```
class VanDerPol "Van der Pol oscillator model"
  Real x(start = 1) "Descriptive string for x";
  Real y(start = 1) "y coordinate";
  parameter Real lambda = 0.3;
equation
  der(x) = y;
  der(y) = -x + lambda*(1 - x*x)*y;
end VanDerPol;
```

Typed  
Declarative  
Equation-based  
Textual Language

Differential equations

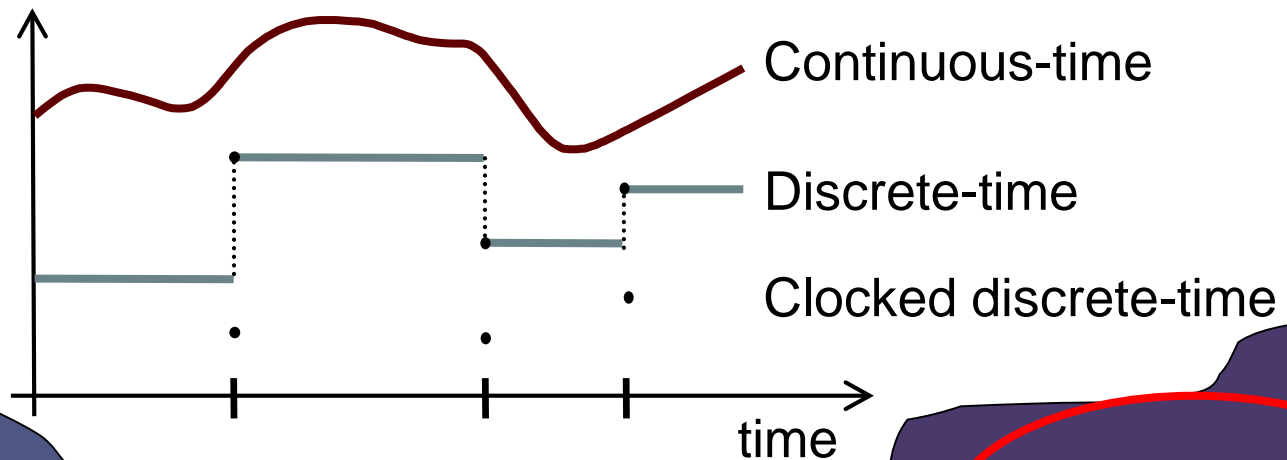
# What is Special about Modelica?

Multi-Domain  
Modeling



Visual Acausal  
Component  
Modeling

Hybrid modeling =  
continuous-time + discrete-time modeling



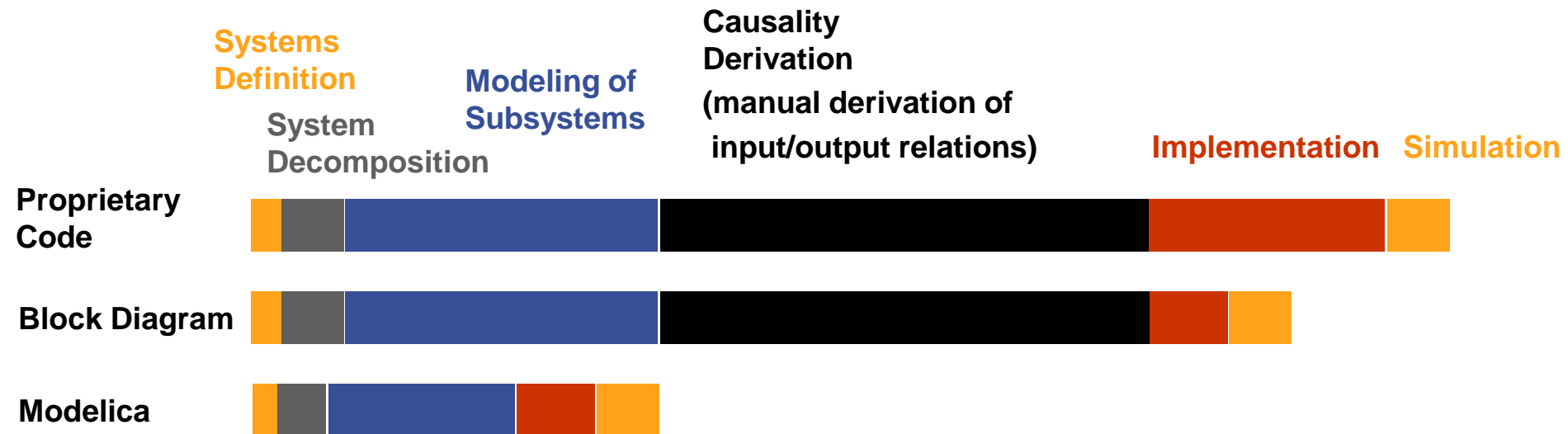
Typed  
Declarative  
Equation-based  
Textual Language

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

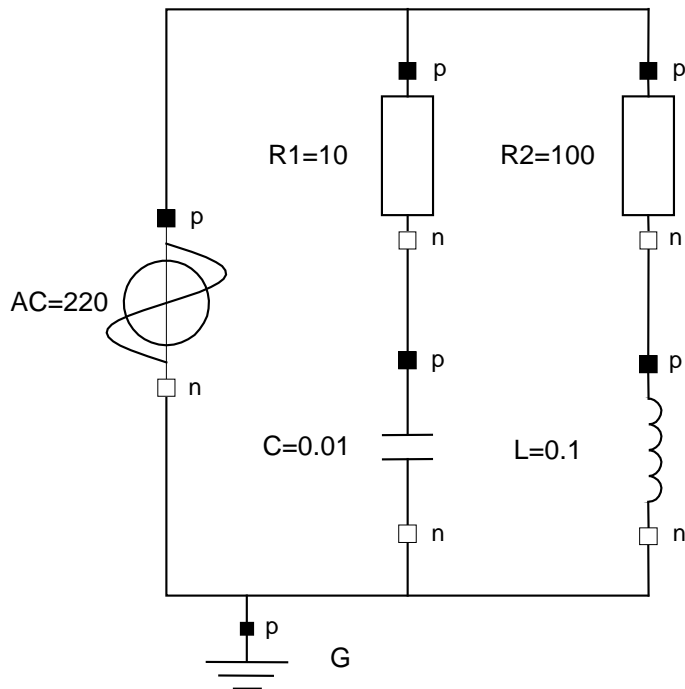


# Modelica vs Simulink Block Oriented Modeling

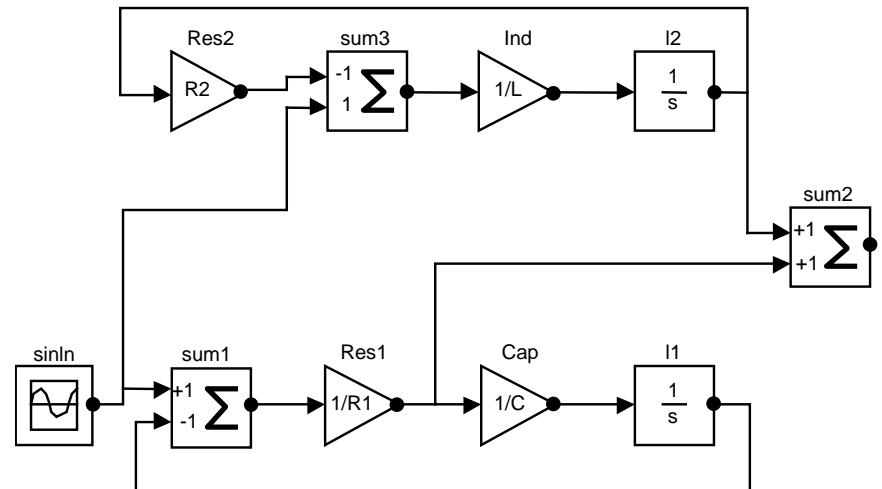
## Simple Electrical Model

**Modelica:**  
Physical model –  
easy to understand

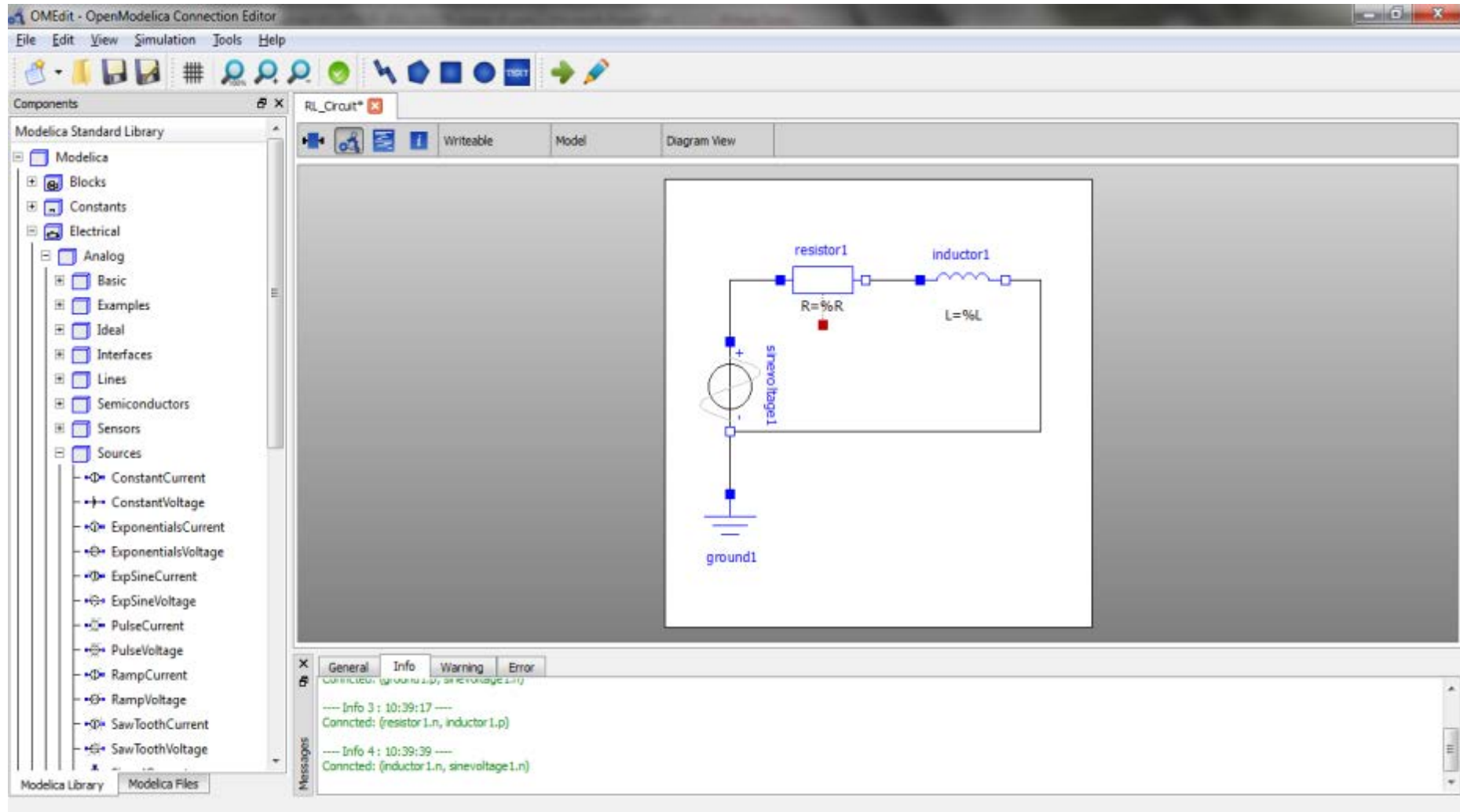
Keeps the  
physical  
structure



**Simulink:**  
Signal-flow model – hard to  
understand



# Graphical Modeling - Using Drag and Drop Composition



# Graphical Modeling with OpenModelica Environment



# Multi-Domain (Electro-Mechanical) Modelica Model

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

**model** DCMotor

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

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

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

```
Ground G;
```

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

```
Inertia load;
```

**equation**

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

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

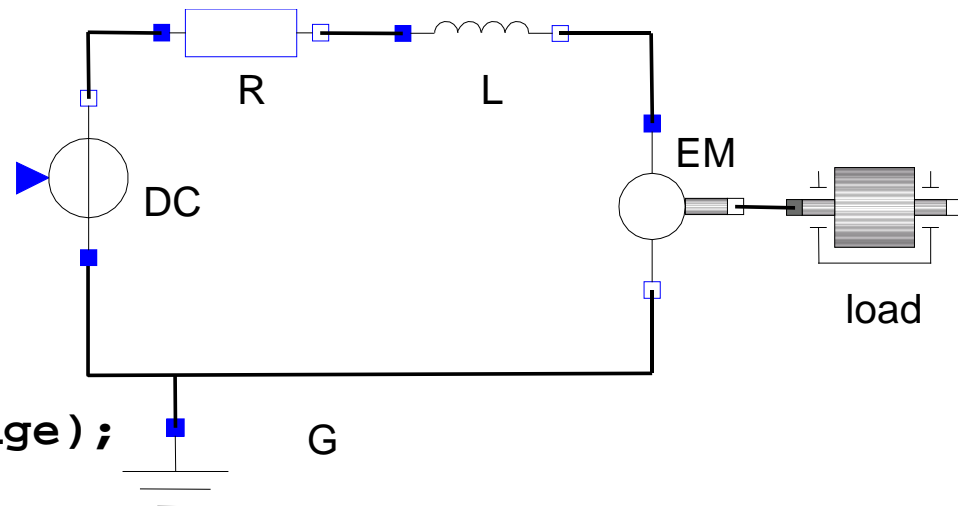
```
connect(L.p, EM.n);
```

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

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

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

**end** DCMotor



# Corresponding DCMotor Model Equations

The following equations are automatically derived from the Modelica model:

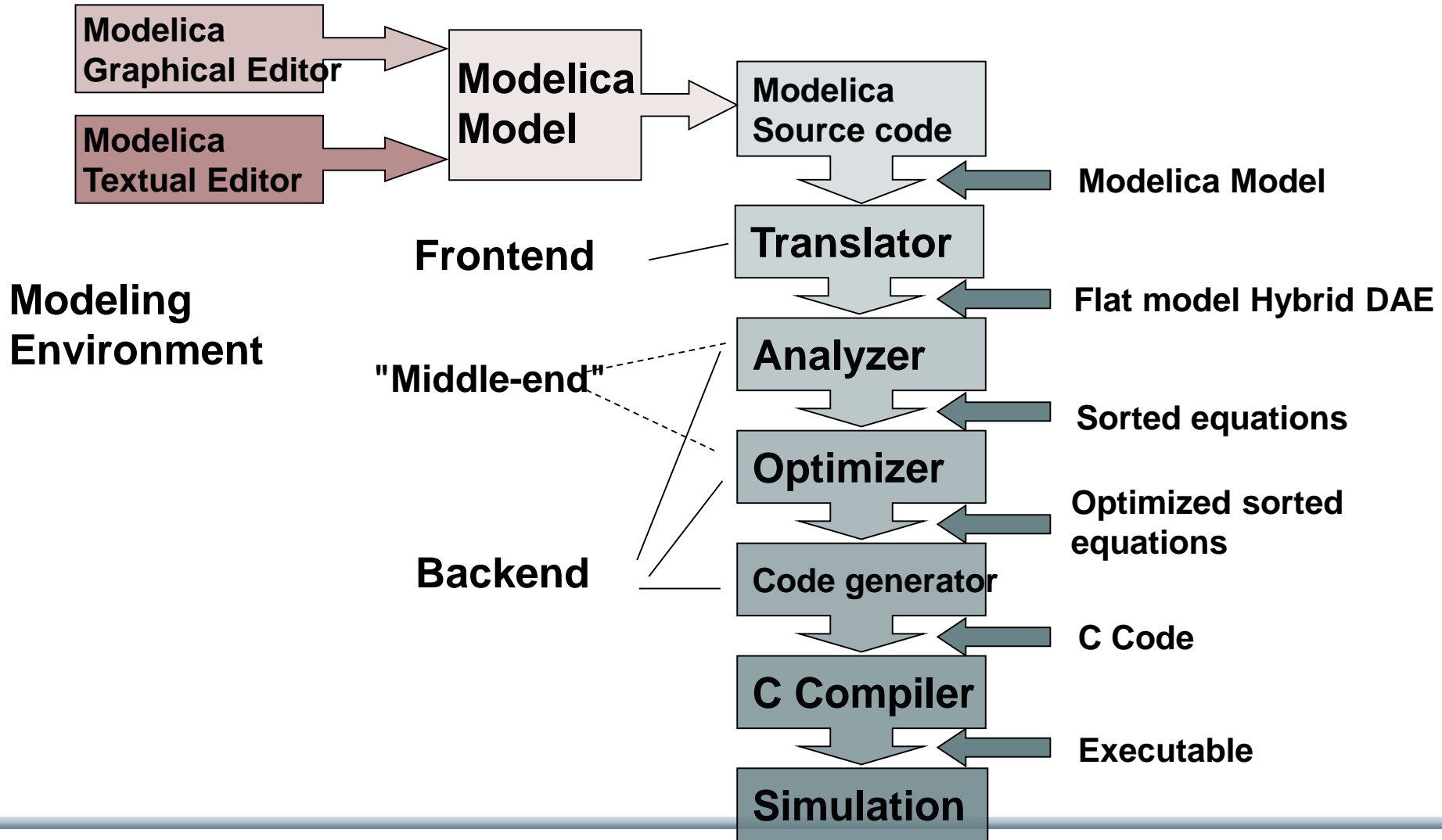
0 == DC.p.i + R.n.i	EM.u == EM.p.v - EM.n.v	R.u == R.p.v - R.n.v
DC.p.v == R.n.v	0 == EM.p.i + EM.n.i	0 == R.p.i + R.n.i
	EM.i == EM.p.i	R.i == R.p.i
0 == R.p.i + L.n.i	EM.u == EM.k * EM.ω	R.u == R.R * R.i
R.p.v == L.n.v	EM.i == EM.M / EM.k	
	EM.J * EM.ω == EM.M - EM.b * EM.ω	L.u == L.p.v - L.n.v
0 ==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

(load component not included)

Automatic transformation to ODE or DAE for simulation:

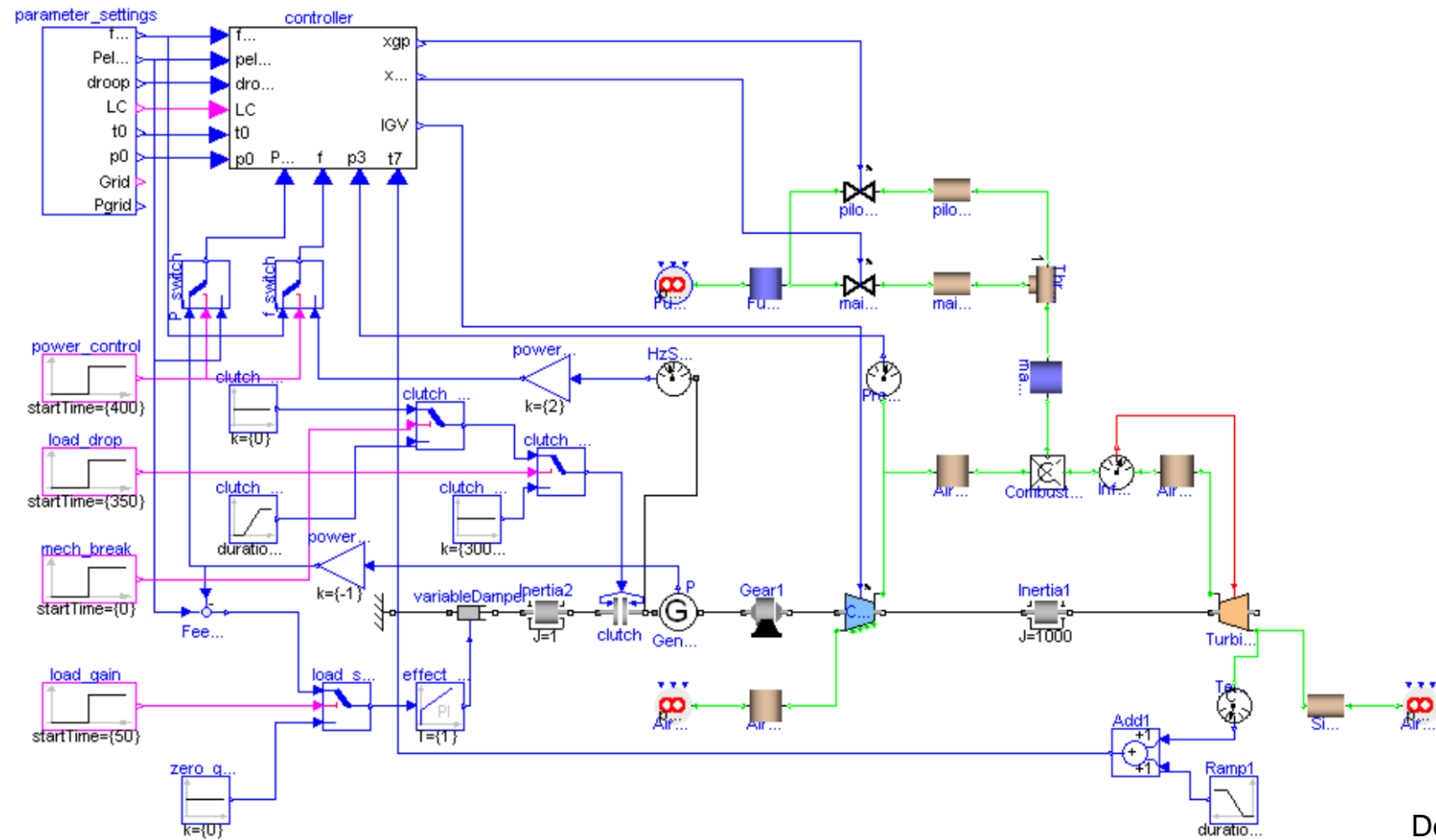
$$\frac{dx}{dt} == f[x, u, t] \quad g\left[\frac{dx}{dt}, x, u, t\right] == 0$$

# Model Translation Process to Hybrid DAE to Code



# Modelica in Power Generation

## GTX Gas Turbine Power Cutoff Mechanism

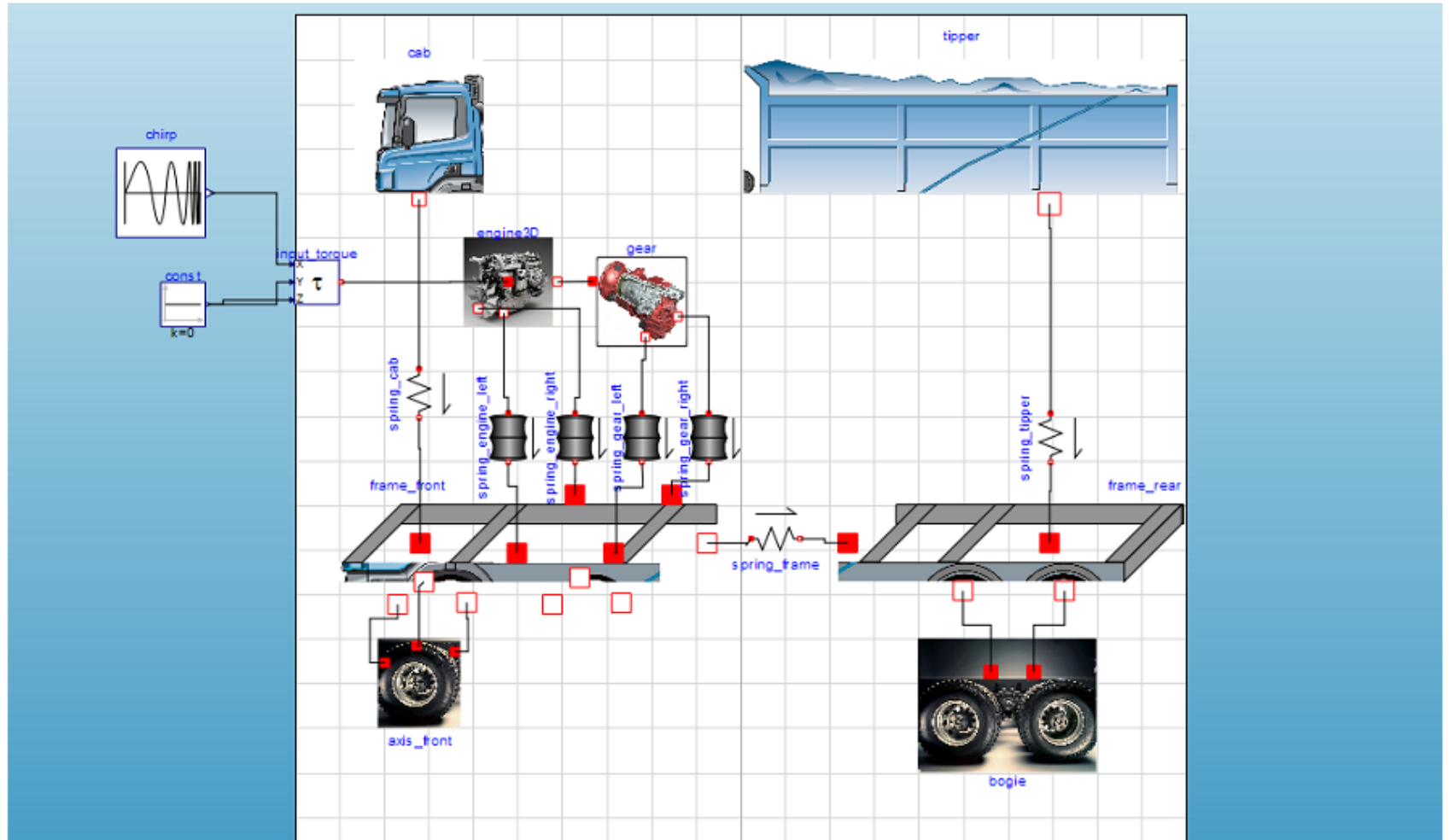


Developed  
by MathCore  
for Siemens

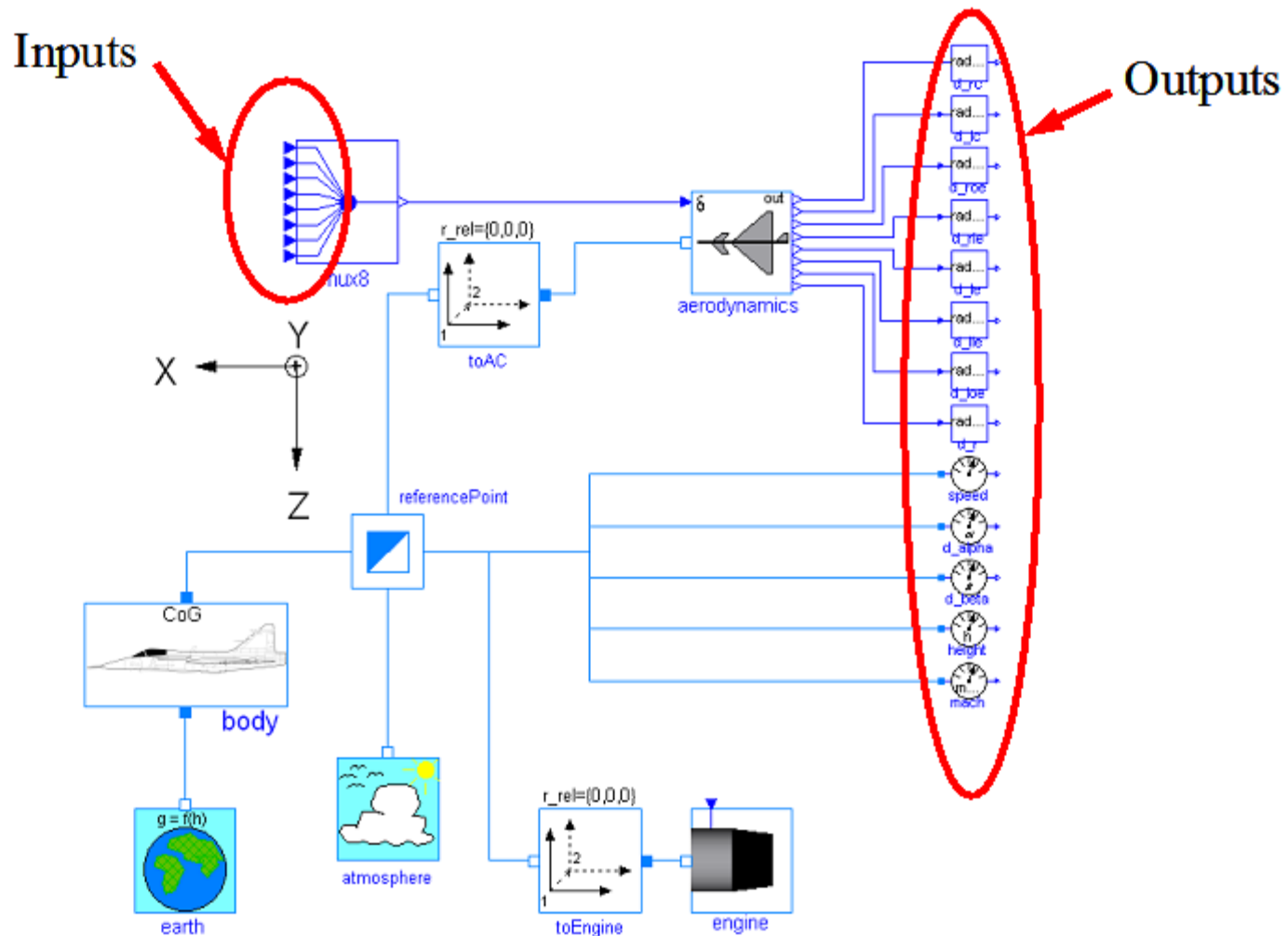
Courtesy of Siemens Industrial Turbomachinery AB



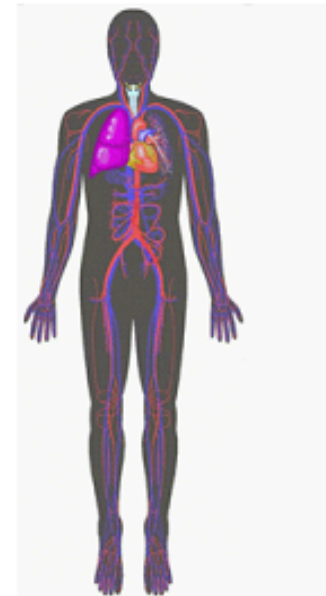
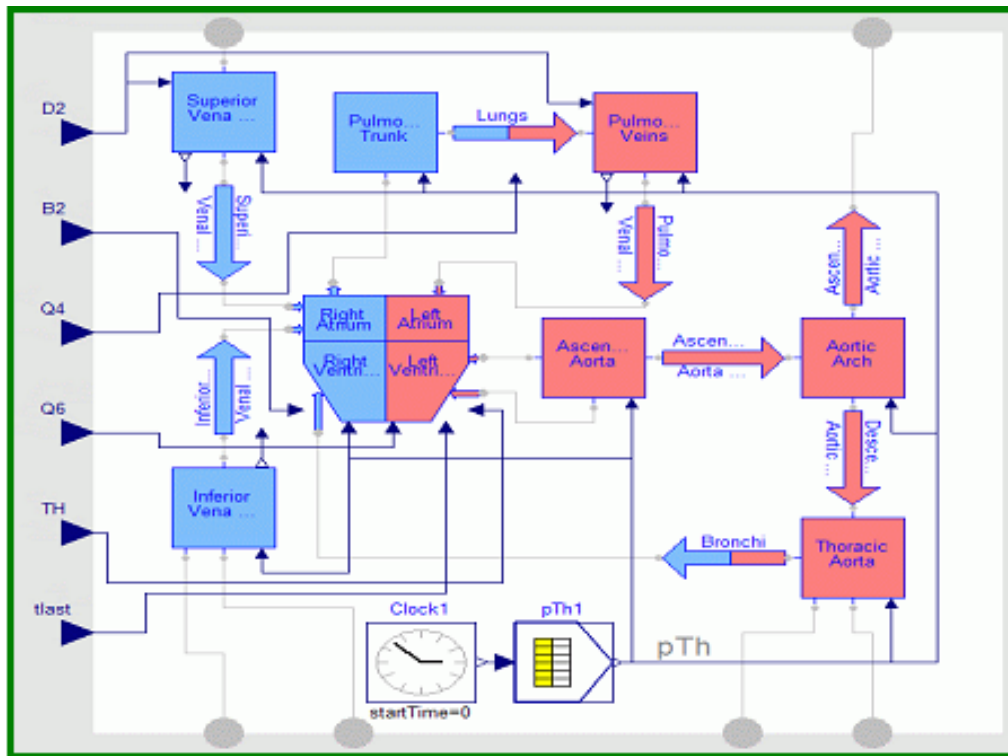
# Modelica in Automotive Industry



# 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 next page)



Courtesy of Tobias Bellmann, DLR,  
Oberpfaffenhofen, Germany

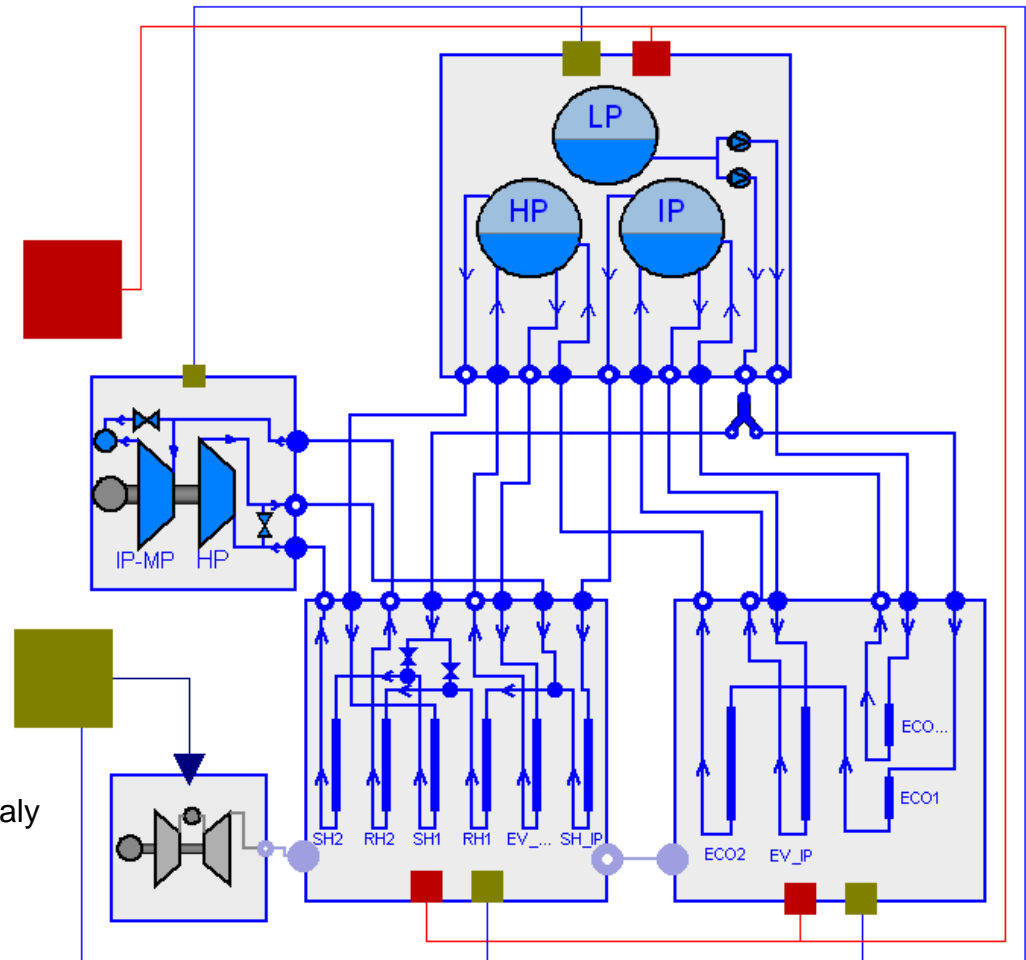
# DLR Real-time Training Simulator Movie Demo



# Combined-Cycle Power Plant

## Plant model – system level

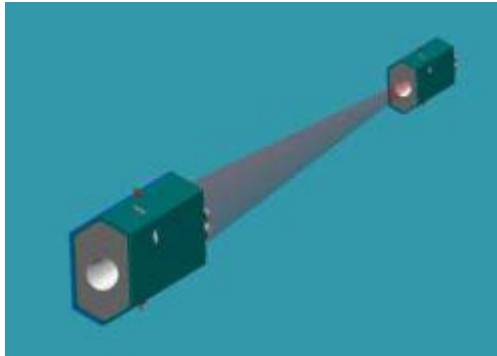
- GT unit, ST unit, Drum boilers unit and HRSG units, connected by thermo-fluid ports and by signal buses
- Low-temperature parts (condenser, feedwater system, LP circuits) are represented by trivial boundary conditions.
- GT model: simple law relating the electrical load request with the exhaust gas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



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

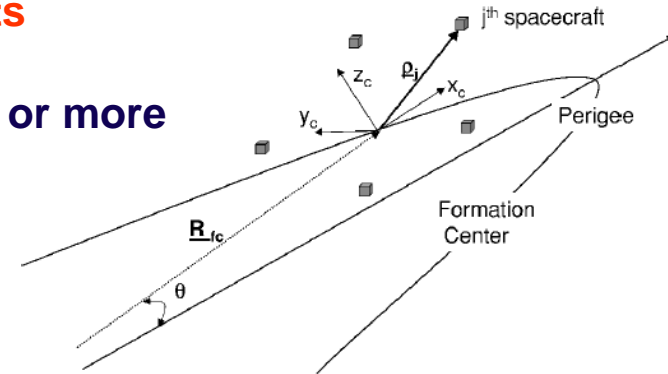
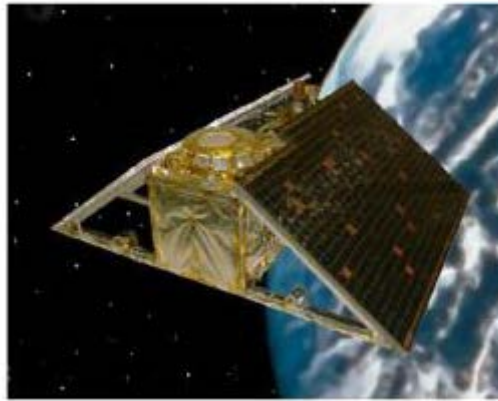


# Modelica Spacecraft Dynamics Library



**Formation flying on elliptical orbits**

**Control the relative motion of two or more spacecraft**



**Attitude control for satellites  
using magnetic coils as actuators**

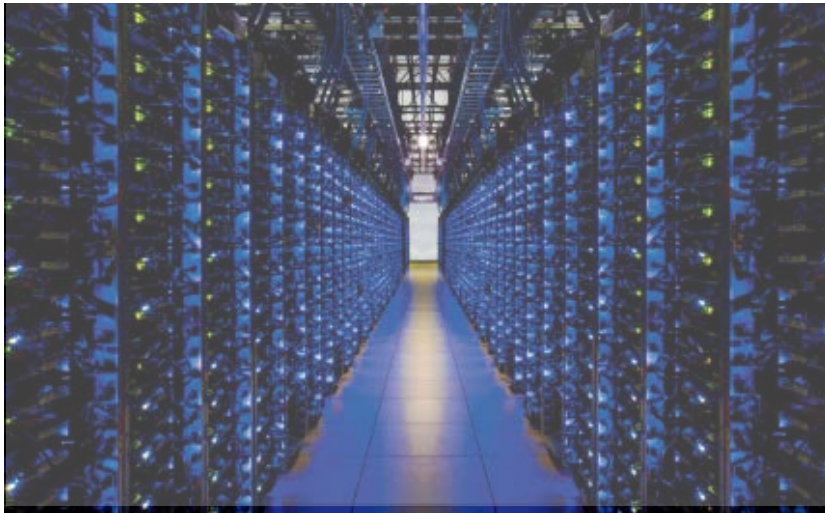
**Torque generation mechanism:  
interaction between coils and  
geomagnetic field**

Courtesy of Francesco Casella, Politecnico di Milano, Italy



# 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

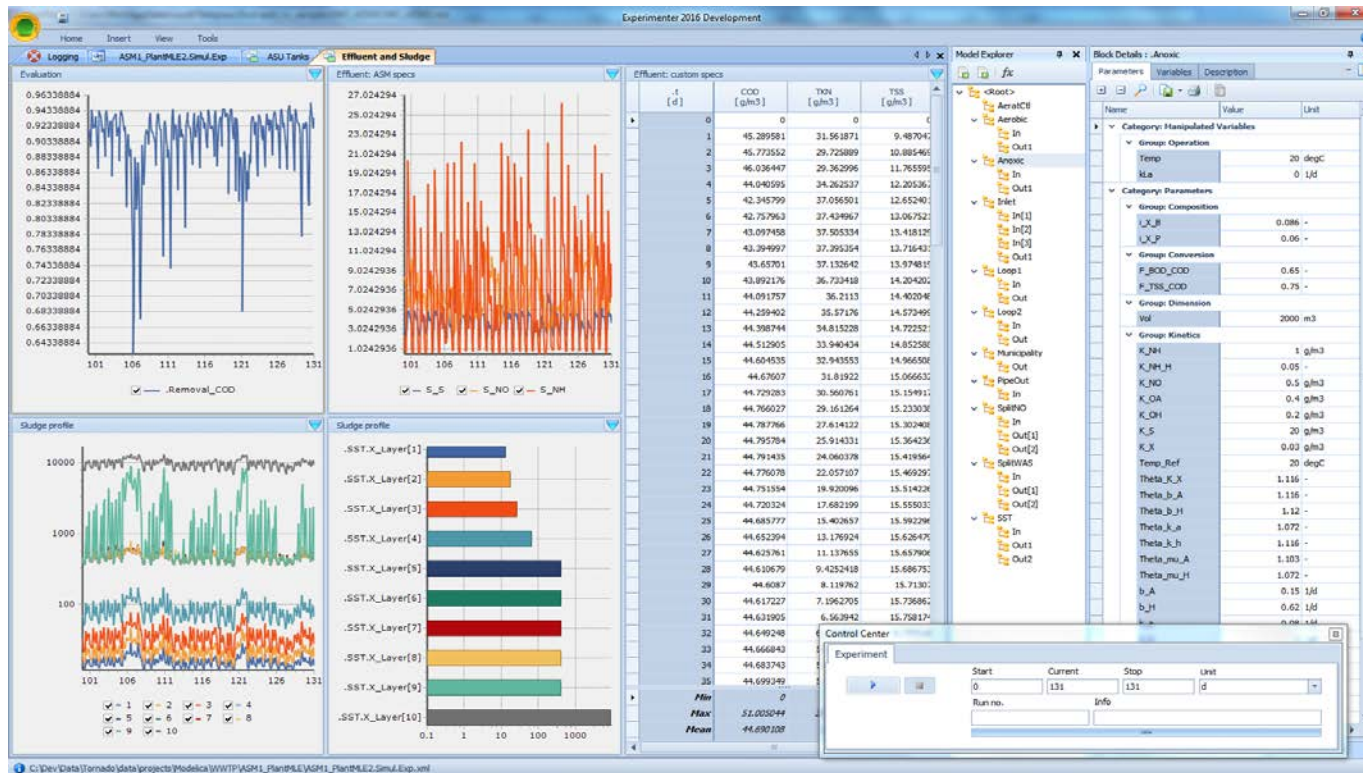
### Manage 7.5% - 10% of German Power

- Since 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, Water Treatment and Sludge

- **MIKE by DHI**, [www.mikebydhi.com](http://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.



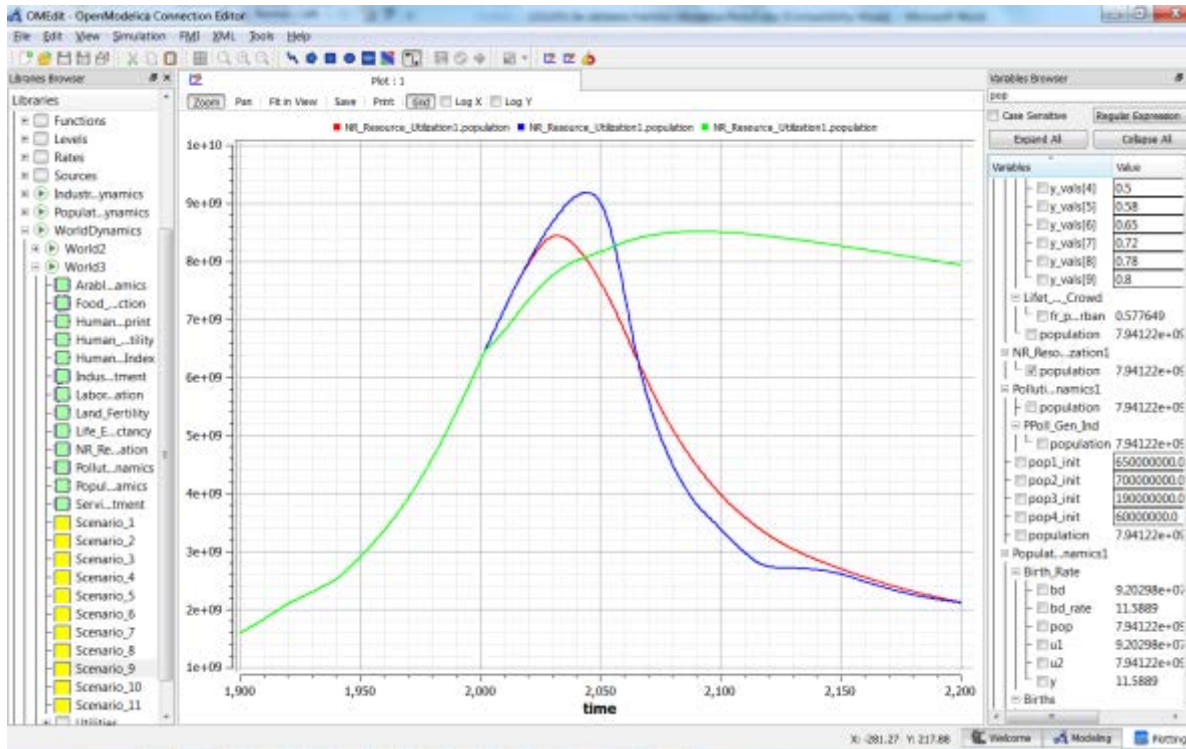
---

**Most important challenge  
for humanity -  
Develop a sustainable society!**

**Use Modelica in to model and optimize  
sustainable technical innovations,  
and a sustainable circular economy**

# 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

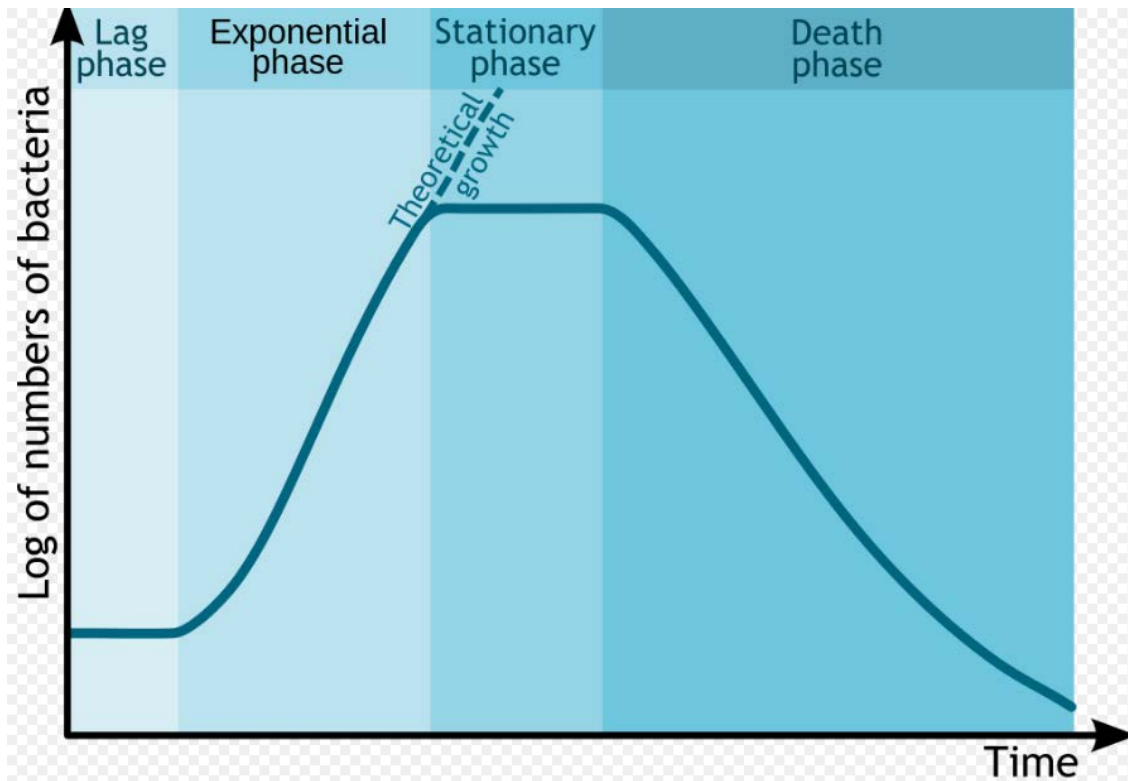
# Are Humans More Intelligent than Bacteria?

**Not yet evident!**

Humans  
on a  
**finite**  
**Earth**

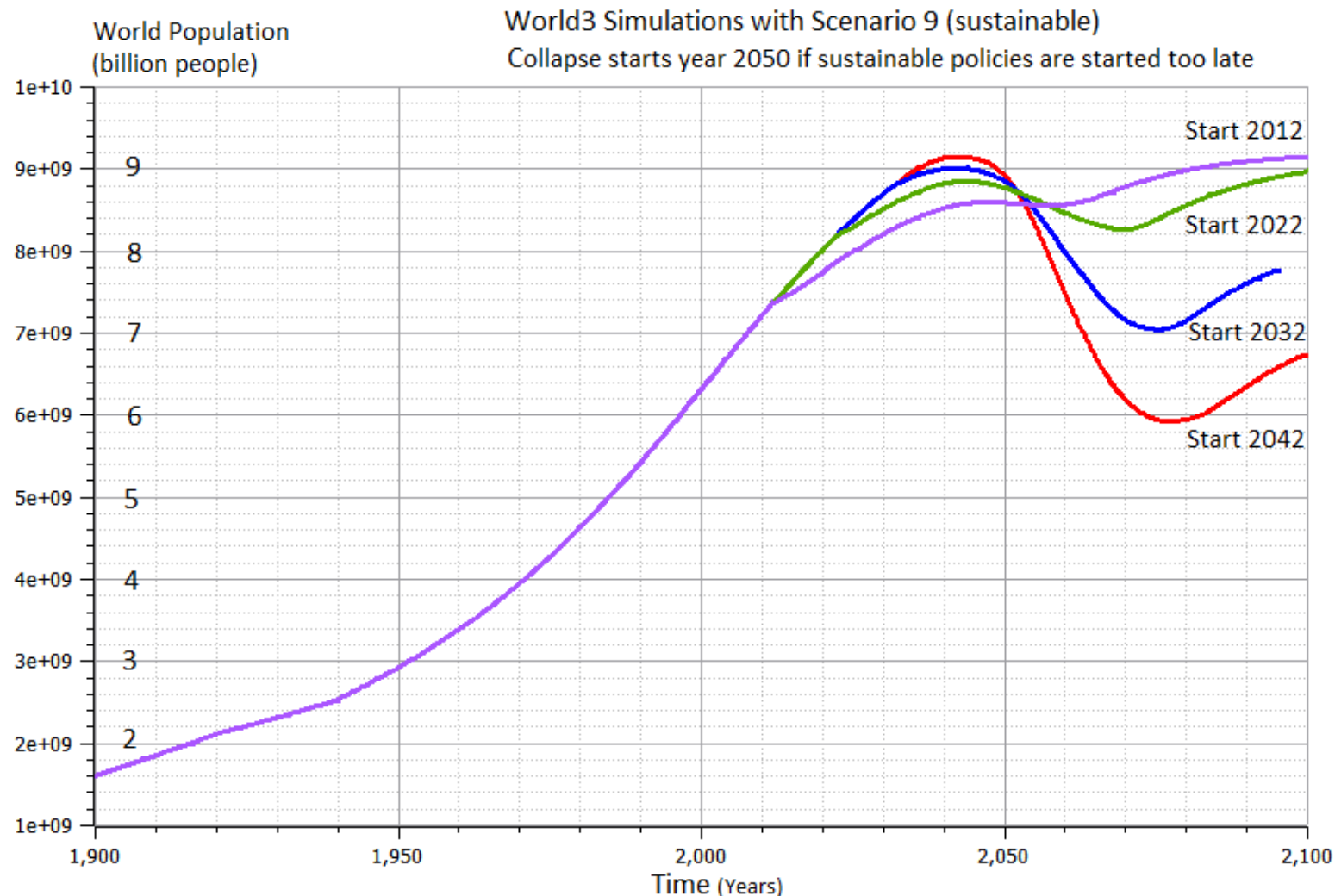
vs

Bacteria  
on a  
**finite**  
**substrate**



**Bacterial growth curve /kinetic curve (Wikipedia)**

# 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

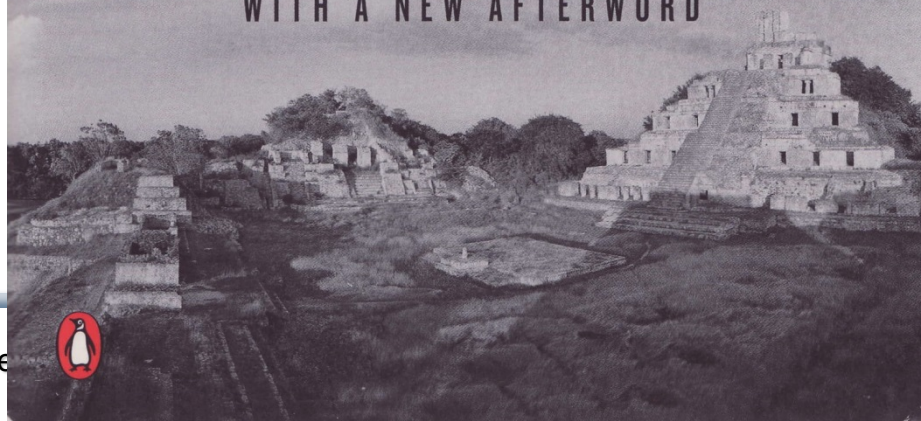
HOW SOCIETIES CHOOSE  
TO FAIL OR SUCCEED

# JARED DIAMOND

author of the Pulitzer Prize-winning

*GUNS, GERMS, and STEEL*

WITH A NEW AFTERWORD



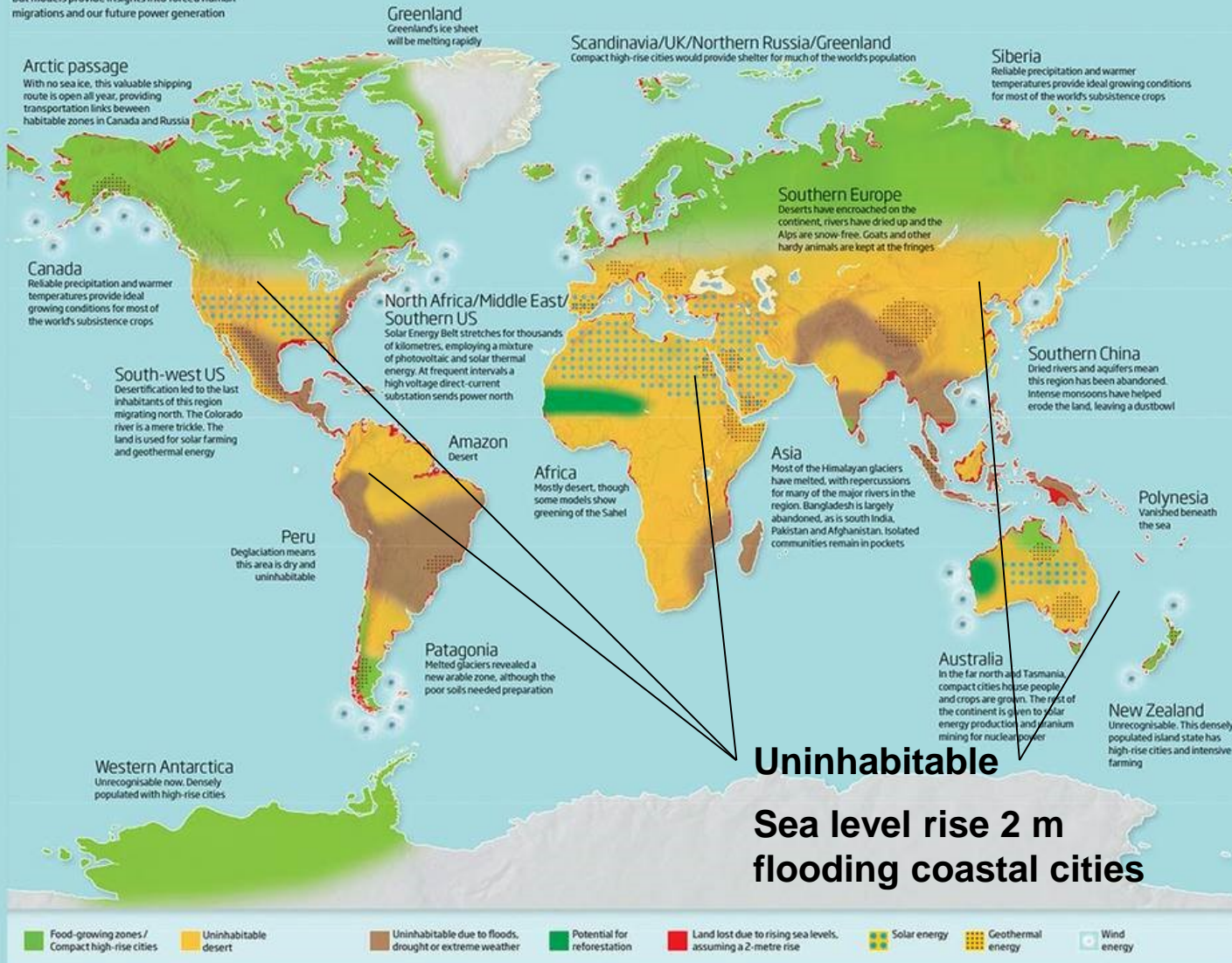


# How the world could be in 80-100 years at a global warming of 4 degrees

## Business-as-usual scenario, IPCC

### The world: 4°C warmer

No one knows exactly what this world will look like, but models provide insights into forced human migrations and our future power generation



- Cities, agriculture
- Uninhabitable desert
- Uninhabitable due to extreme weather
- Flooded

**Massive migration to northern Europe, Russia, and Canada**

### Example Emissions

CO<sub>2</sub>e / person

- Earth can handle 2 ton/yr
- Flight Spain – 1 ton
- Flight Canarys – 2 ton
- Flight Thailand – 4 ton

### References

New Scientist, 28 February 2009  
 IPCC, business as usual scenario  
[www.climate-lab-book.ac.uk](http://www.climate-lab-book.ac.uk)  
[www.atmosfair.de](http://www.atmosfair.de)

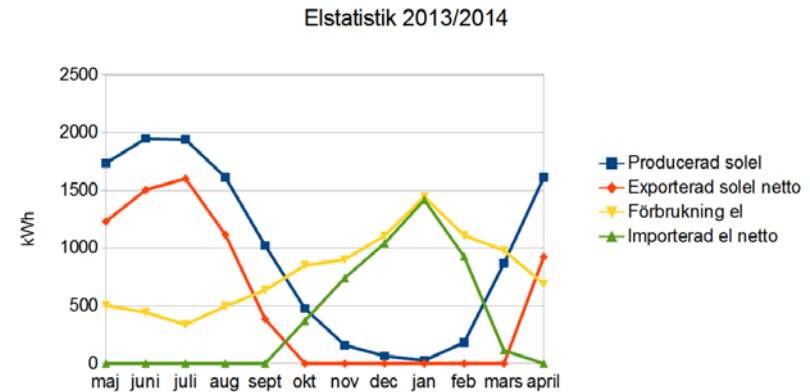
# What Can You Do?

## 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



**Small car Renault ZOE; 5 seat;  
Range with 51 kwh battery (2020)**

- WLTP drive cycle 390 km
- In practice, summer, ca 360 km
- Winter: ca 240 km

**Can use common Type 2 AC chargers (up to 22kW)**



**DLR ROboMObil**

- experimental electric car
- Modelica models



**2018, Tesla Model 3 LR, range 560 km**

**Tesla Model S, range about 550 km**

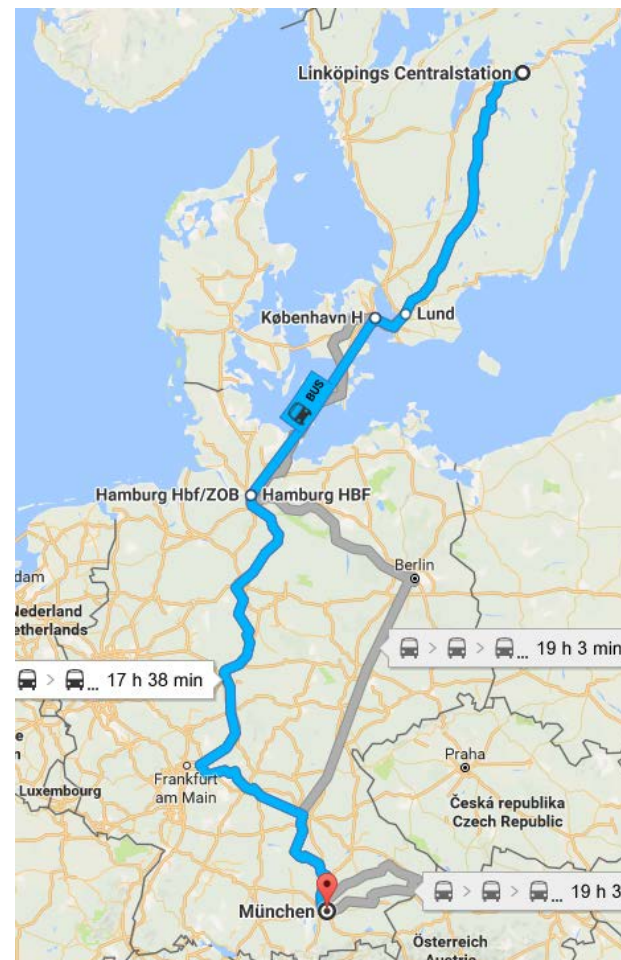
# What Can You Do?

## 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 CO<sub>2</sub>e 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

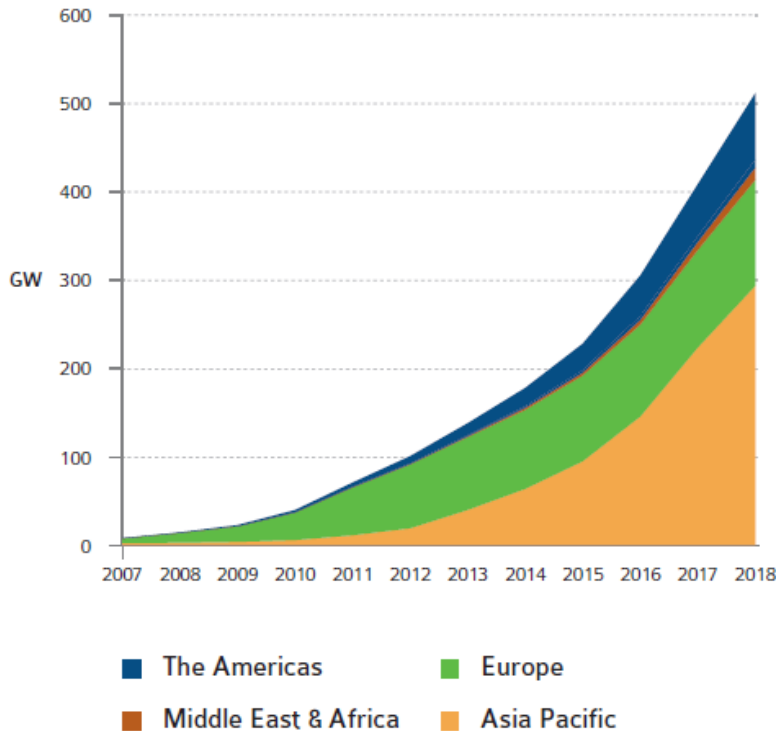
A world map showing the continents of North America, South America, Africa, Europe, Asia, and Australia. Overlaid on the map are several small yellow rectangles, each representing a surface area needed to provide 100% of the world's solar energy. The rectangles are located in Mexico, Brazil, North Africa, the Middle East, India, China, and Australia. A large yellow text box is positioned in the upper right quadrant of the map.

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



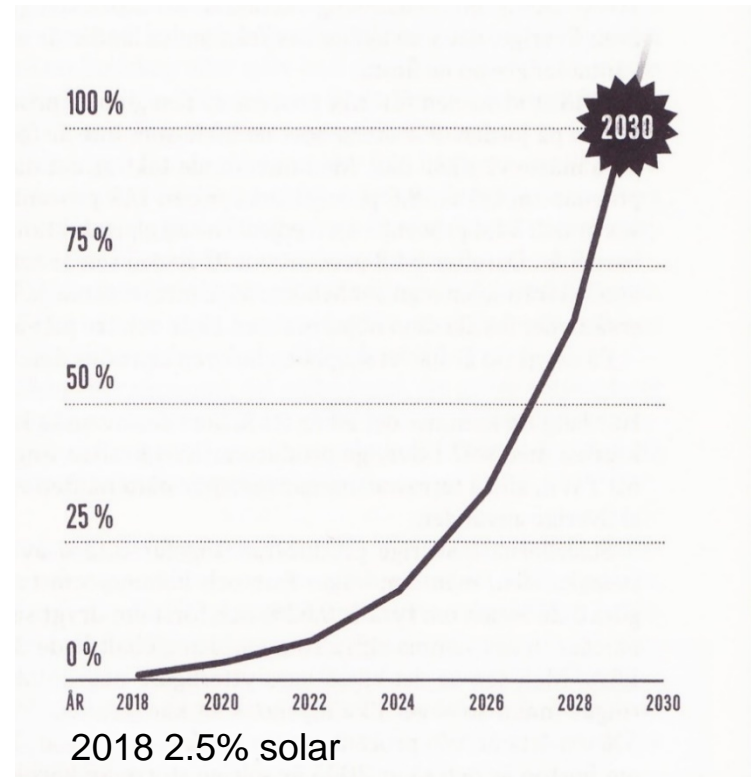
# Solar Energy PhotoVoltaics Growth Trends

FIGURE 2.5: EVOLUTION OF REGIONAL PV INSTALLATIONS (GW)



Almost  
Exponential  
worldwide  
Growth of  
Photovoltaics  
2006 – 2018

IEA PVPS  
TRENDS IN  
PHOTOVOLTAIC  
APPLICATIONS  
2019



100% of global electricity  
production year 2030 if  
strong exponential growth  
continues

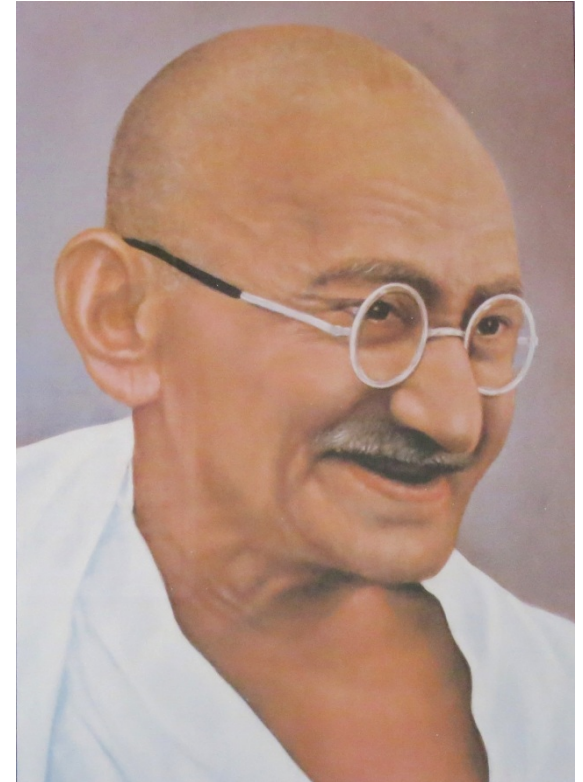
# Sustainable Society Necessary for Human Survival

## Almost Sustainable

- India, recently 1.4 ton CO<sub>2</sub>/person/year
- Healthy vegetarian food
- Small-scale agriculture
- Small-scale shops
- Simpler life-style (Mahatma Gandhi)

## Non-sustainable

- USA 17 ton CO<sub>2</sub>, Sweden 7 ton CO<sub>2</sub>/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 centers
- 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 released April 2017
  - 3.5 released February 2021
- Modelica Association was established in 2000 in Linköping
  - Open, non-profit organization

# Modelica Conferences

- The 1<sup>st</sup> International Modelica conference October, 2000
- The 2<sup>nd</sup> International Modelica conference March 18-19, 2002
- The 3<sup>rd</sup> International Modelica conference November 5-6, 2003 in Linköping, Sweden
- The 4<sup>th</sup> International Modelica conference March 6-7, 2005 in Hamburg, Germany
- The 5<sup>th</sup> International Modelica conference September 4-5, 2006 in Vienna, Austria
- The 6<sup>th</sup> International Modelica conference March 3-4, 2008 in Bielefeld, Germany
- The 7<sup>th</sup> International Modelica conference Sept 21-22, 2009 in Como, Italy
- The 8<sup>th</sup> International Modelica conference March 20-22, 2011 in Dresden, Germany
- The 9<sup>th</sup> International Modelica conference Sept 3-5, 2012 in Munich, Germany
- The 10<sup>th</sup> International Modelica conference March 10-12, 2014 in Lund, Sweden
- The 11<sup>th</sup> International Modelica conference Sept 21-23, 2015 in Versailles, Paris
- The 12<sup>th</sup> International Modelica conference May 15-17, 2017 in Prague, Czech Rep
- The 13<sup>th</sup> International Modelica conference March 4-6, 2019, Regensburg, Germany
- The 14<sup>th</sup> International Modelica conference Sept 20-24, 2021, Linköping, Sweden
- Also: Asian Modelica conferences 2016, 2017, 2018, 2020, 2022
- Also: US Modelica conference 2018, 2020, 2022

---

# **Exercises Part I**

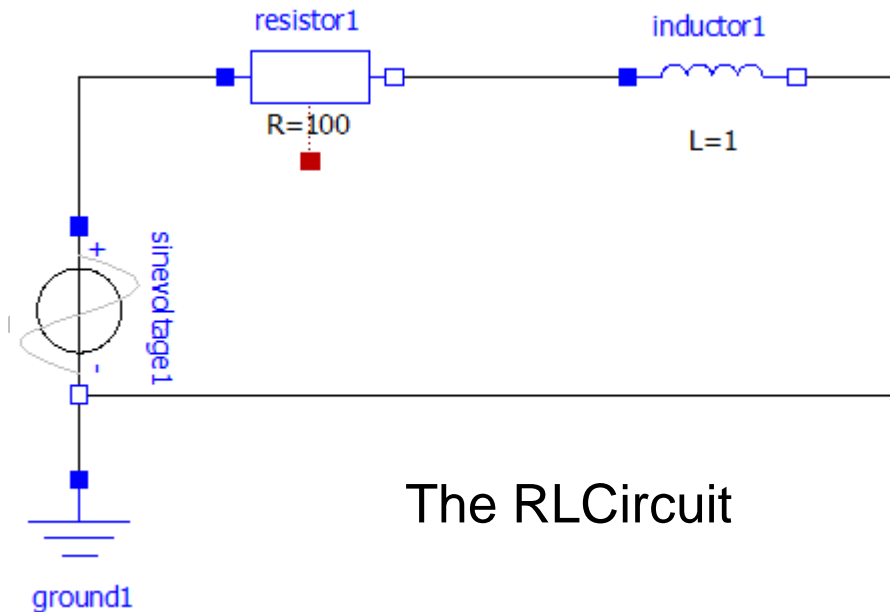
## **Hands-on graphical modeling**

### **(15 minutes)**

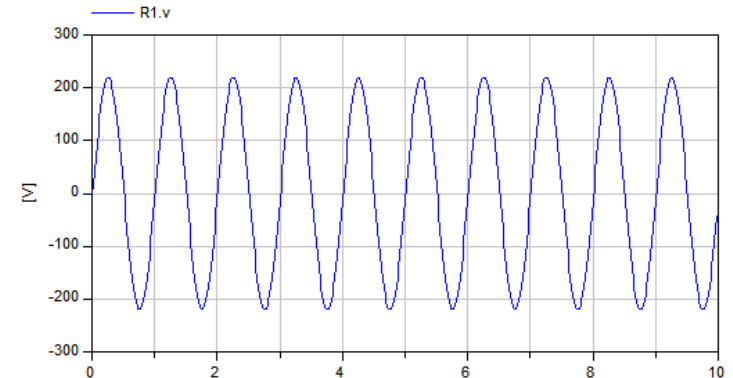


# Exercises Part I – Basic Graphical Modeling

- (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 Modelica Class**, 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**:

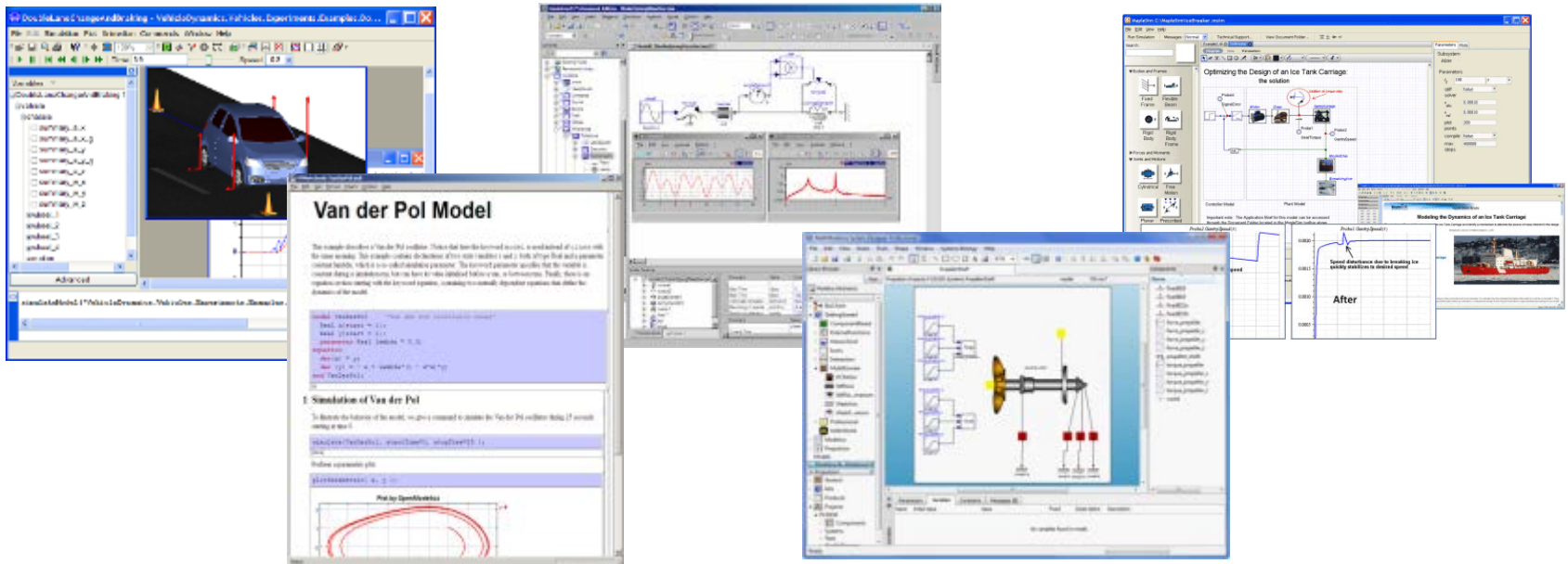
- Contains The standard Modelica library components
- The **Modelica files** contains the list of models you have created.

# Exercises Part I – OMEdit Instructions (Part II)

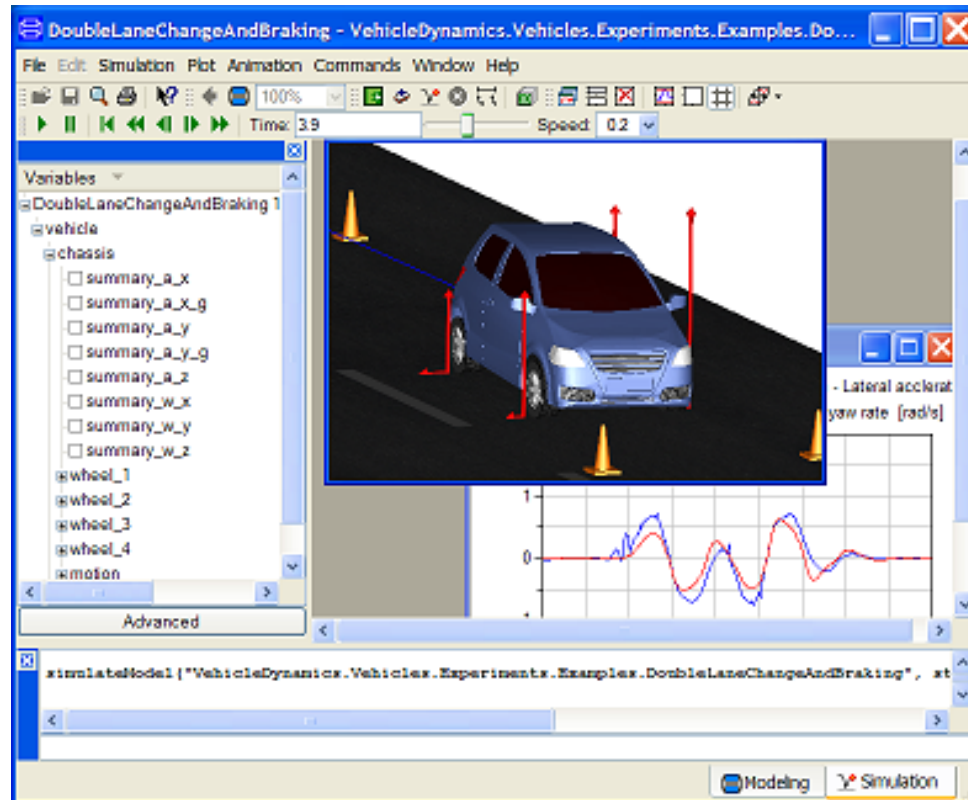
- For the RLCircuit model, **browse** the Modelica standard library and **add** the following component models:
  - Add `Ground`, `Inductor` and `Resistor` component models from `Modelica.Electrical.Analog.Basic` package.
  - Add `SineVoltage` component model from `Modelica.Electrical.Analog.Sources` package.
- Make the corresponding **connections** between the component models as shown in the previous slide.
- To **draw a connection line**: first single-click on a connector box; then start drawing while keeping the mouse button down; after drawing a little you can release the mouse button and continue drawing.
- **Simulate** the model
  - Go to the 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

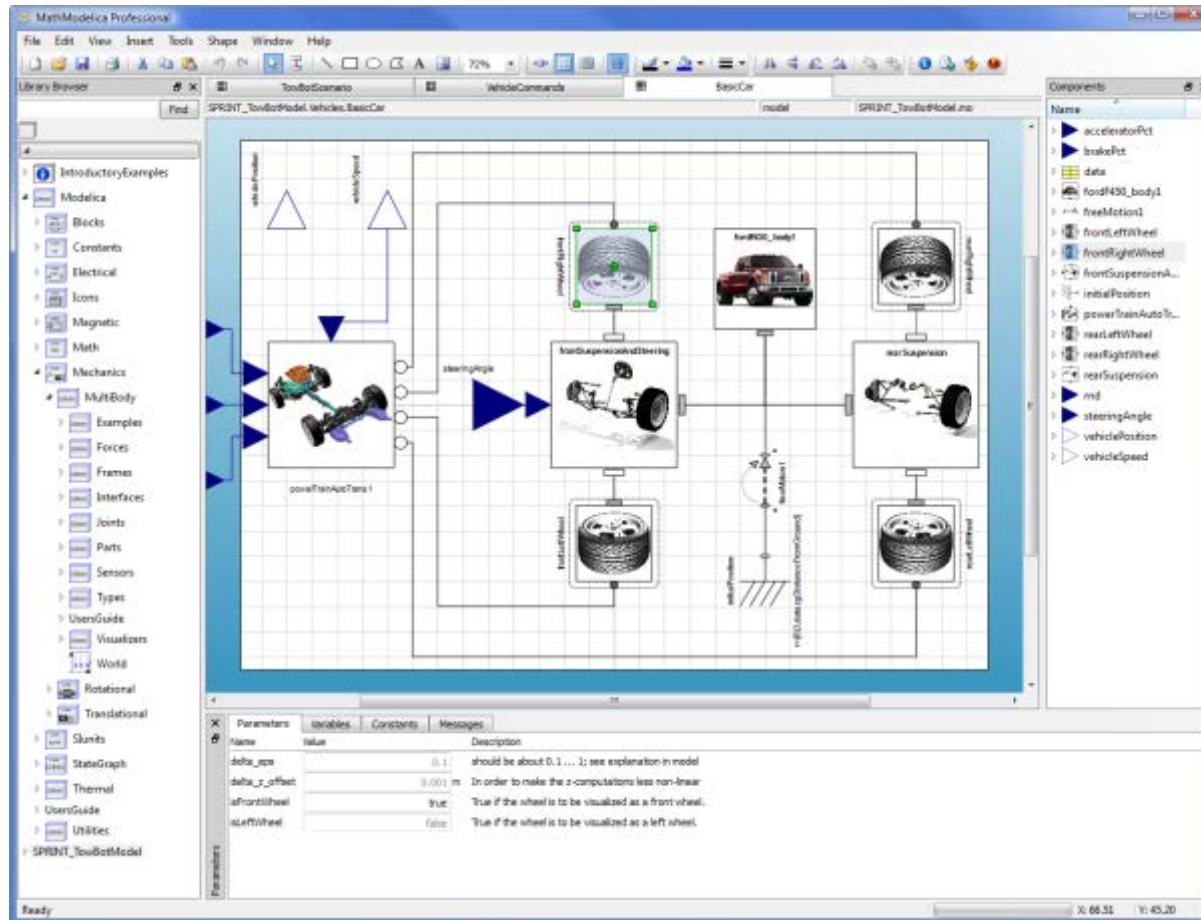


# Dymola

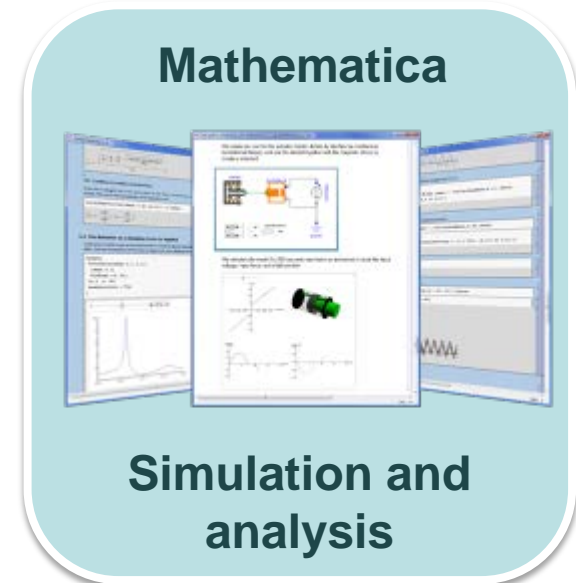


- Dassault Systemes Sweden
- Sweden
- First Modelica tool on the market
- Initial main focus on automotive industry
- [www.dymola.com](http://www.dymola.com)

# Wolfram System Modeler – Wolfram MathCore



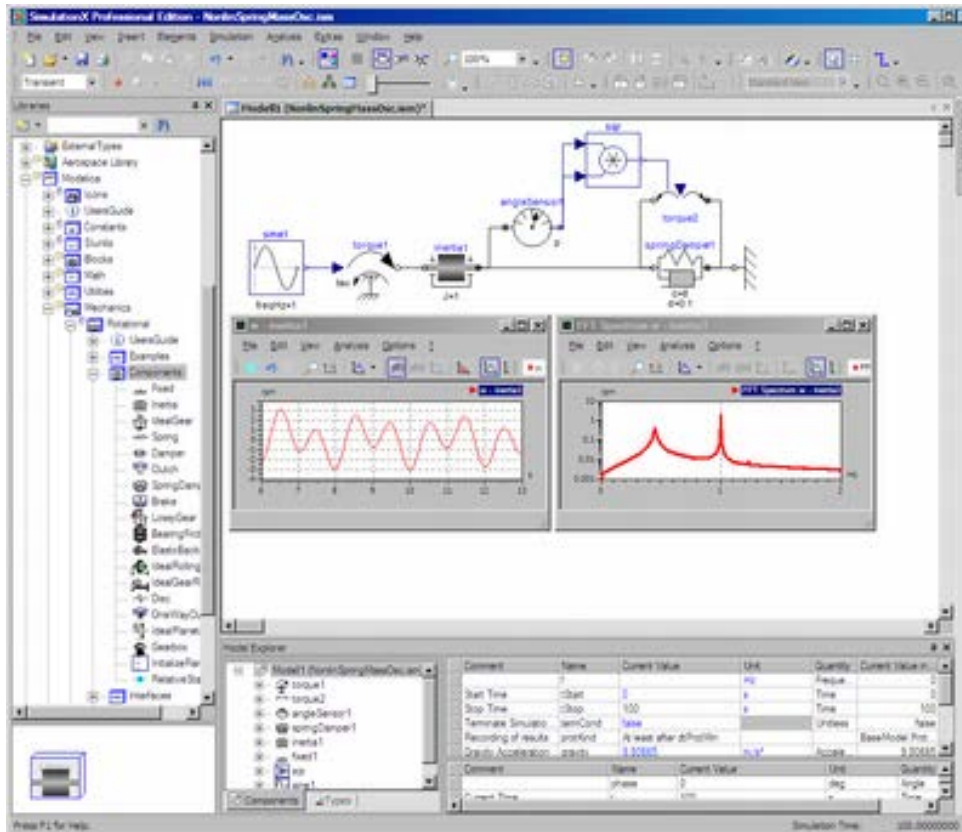
- Wolfram Research
- USA, Sweden
- General purpose
- Mathematica integration
- [www.wolfram.com](http://www.wolfram.com)
- [www.mathcore.com](http://www.mathcore.com)



Car model graphical view

Courtesy  
Wolfram  
Research

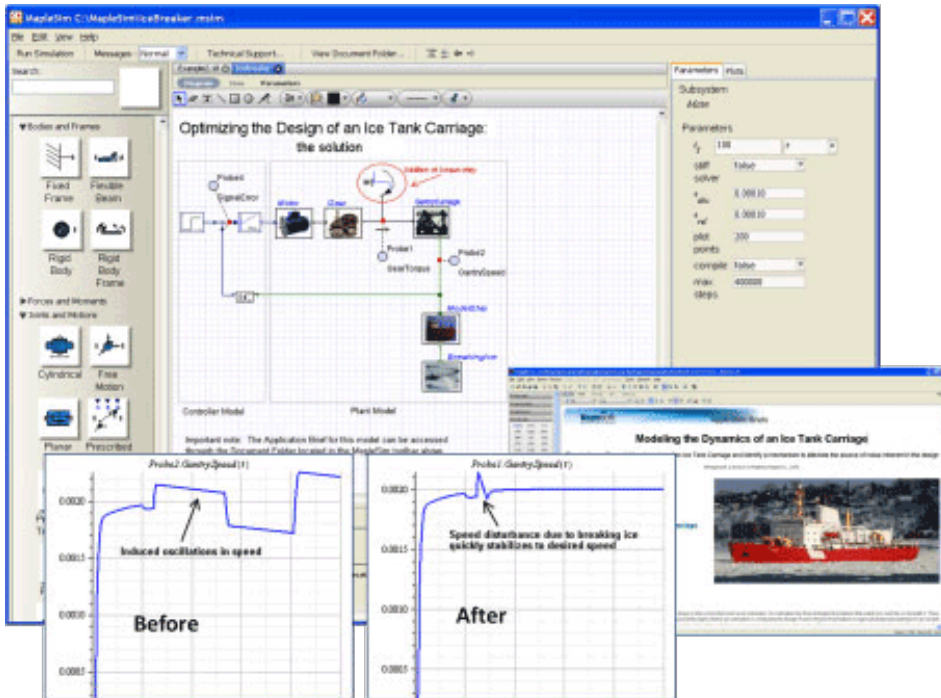
# Simulation X



- ITI GmbH (Part of ESI Group)
- Germany
- Mechatronic systems
- [www.simulationx.com](http://www.simulationx.com)



# MapleSim



- Maplesoft
- Canada
- Integrated with Maple
- [www.maplesoft.com](http://www.maplesoft.com)



# Modelon



## Modelon Library Suite

*Powered by Modelica*

Our suite of libraries, built on the Modelica open standard, delivers state-of-the-art system models for a wide range of industrial applications.



## Modelon Creator Suite

Our creator suite is a powerful platform for model creation, automation, simulation and optimization.



## Modelon Deployment Suite

*Powered by FMI*

Our comprehensive suite of deployment products, built on the FMI open standard, enables collaboration and rapid deployment of system models across multiple platforms, varying tools, and organizations.

- Modelon
- Sweden and International
- Library Suite
- Creator Suite with Impact product and Optimica Compiler Toolbox and WAMS model editor
- [www.modelon.com](http://www.modelon.com)

# The OpenModelica Environment

## [www.OpenModelica.org](http://www.OpenModelica.org)

The screenshot shows the OpenModelica website homepage. At the top is a blue banner with the 'OpenModelica' logo on the left and 'Login' and 'Create an account' links on the right. Below the banner is a dark navigation bar with links: HOME, DOWNLOAD, TOOLS & APPS, USERS, DEVELOPERS, FORUM, EVENTS, RESEARCH, and a search box. The main content area is divided into three columns. The left column, titled 'Top information', contains three sections: 'Industrial Products' with a small diagram icon, 'OMEdit' with a gear icon, and 'Library Coverage' with a small circuit icon. The middle column, titled 'Introduction', contains a paragraph about the project, a goal statement, and a screenshot of the OMEEdit software interface. The right column, titled 'Latest news', contains a list of recent releases and events. At the bottom left, there is a video player titled 'Overview of M...' showing a person speaking.

**OpenModelica** Login Create an account

HOME DOWNLOAD TOOLS & APPS USERS DEVELOPERS FORUM EVENTS RESEARCH search ...

### Top information

**Industrial Products**  
Commercial Applications using OpenModelica

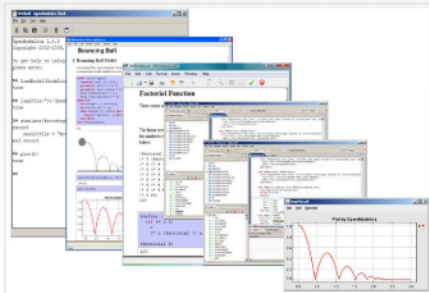
**OMEdit**  
Enhanced OpenModelica Connection Editor.

**Library Coverage**  
Latest library coverage.

### Introduction

OPENMODELICA is an open-source Modelica-based modeling and simulation environment intended for industrial and academic usage. Its long-term development is supported by a non-profit organization – the [Open Source Modelica Consortium \(OSMC\)](#). An overview journal [paper](#) is available and [slides](#) about Modelica and OpenModelica.

The goal with the OpenModelica effort 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. We invite researchers and students, or any interested developer to participate in the project and cooperate around OpenModelica, tools, and applications.



[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®.  
Interactive step-by-step beginners Modelica [on-line spoken tutorials](#)

### Latest news

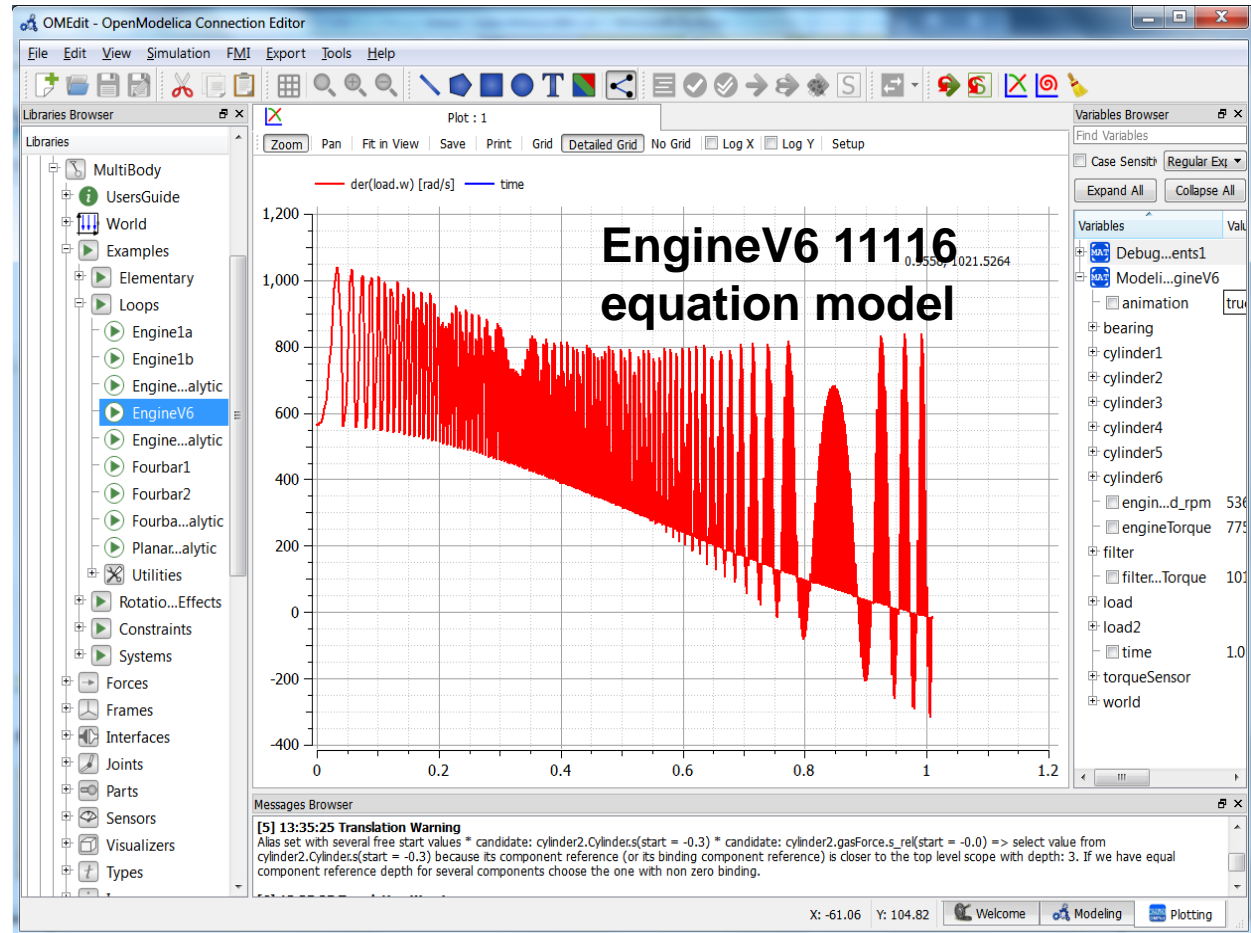
- September 4, 2021: OpenModelica 1.18.0 released!
- July 12, 2021: OpenModelica 1.18.0-dev.beta1 released!
- Join the Modelica Conference 2021!
- March 23, 2021: OpenModelica 1.17.0 released!
- February 26, 2021: OpenModelica 1.16.5 released!
- February 22, 2021: OpenModelica 1.16.4 released!
- HUBCAP Open Calls
- December 21, 2020: OpenModelica 1.16.2 released!
- November 17, 2020: OpenModelica 1.16.1 released!
- November 9. An OpenModelica overview article has been published in the MIC Journal.

Overview of M...

# OpenModelica – Free Open Source Tool

## developed by the Open Source Modelica Consortium (OSMC)

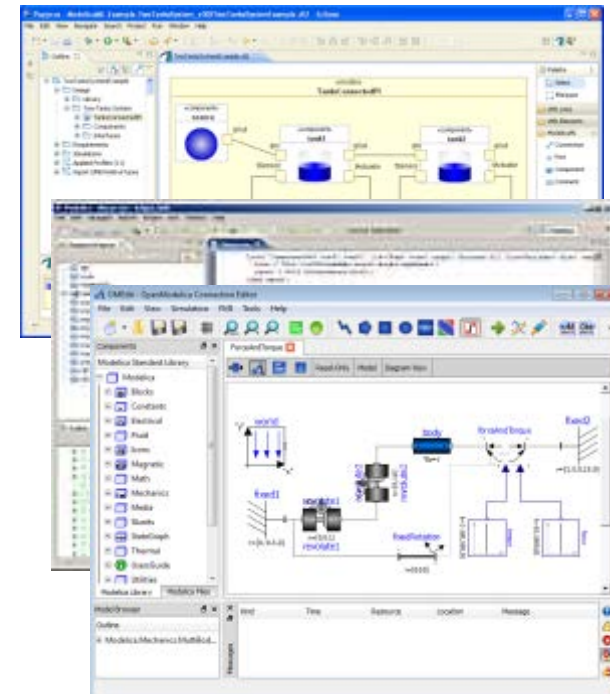
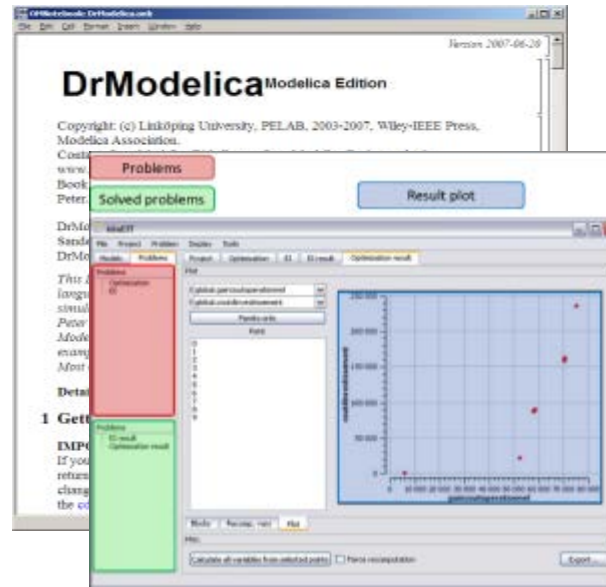
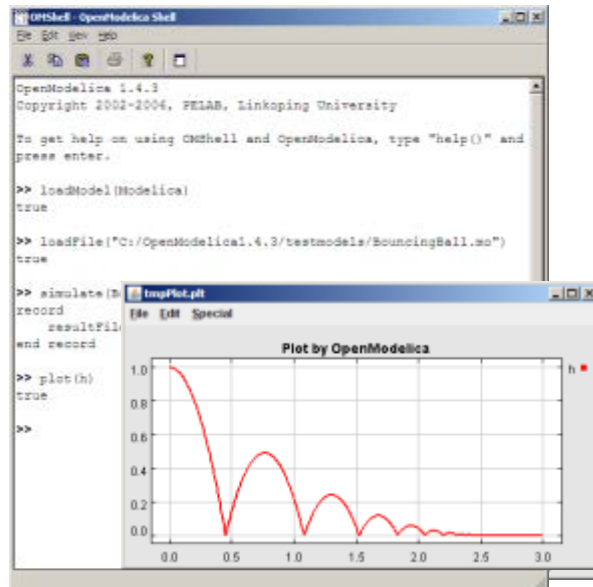
- Graphical editor
- Model compiler and simulator
- Debugger
- Performance analyzer
- Dynamic optimizer
- Symbolic modeling
- Parallelization
- Electronic Notebook and OMWebbook for teaching
- Spokentutorial for teaching



# The OpenModelica Open Source Environment

[www.openmodelica.org](http://www.openmodelica.org)

- Advanced Interactive Modelica compiler (OMC)
    - Supports most of the Modelica Language
    - **Modelica, Python, Julia, Matlab scripting**
  - OMSimulator – FMI Simulation/Co-simulation
  - 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



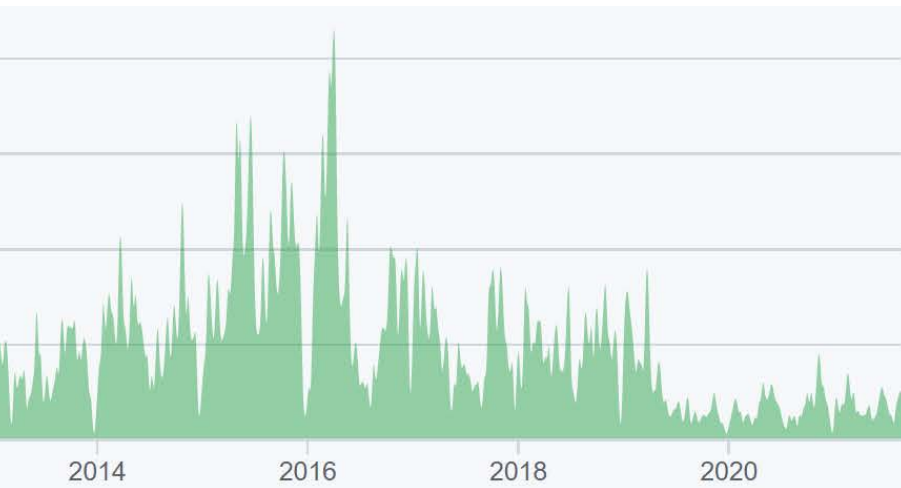
# OSMC – International Consortium for Open Source Model-based Development Tools, 53 members Feb 2022

Founded Dec 4, 2007

## Open-source community services

- Website and Support Forum
- Version-controlled source base
- Bug database
- Development courses
- [www.openmodelica.org](http://www.openmodelica.org)

## Commits Statistics



## Industrial members

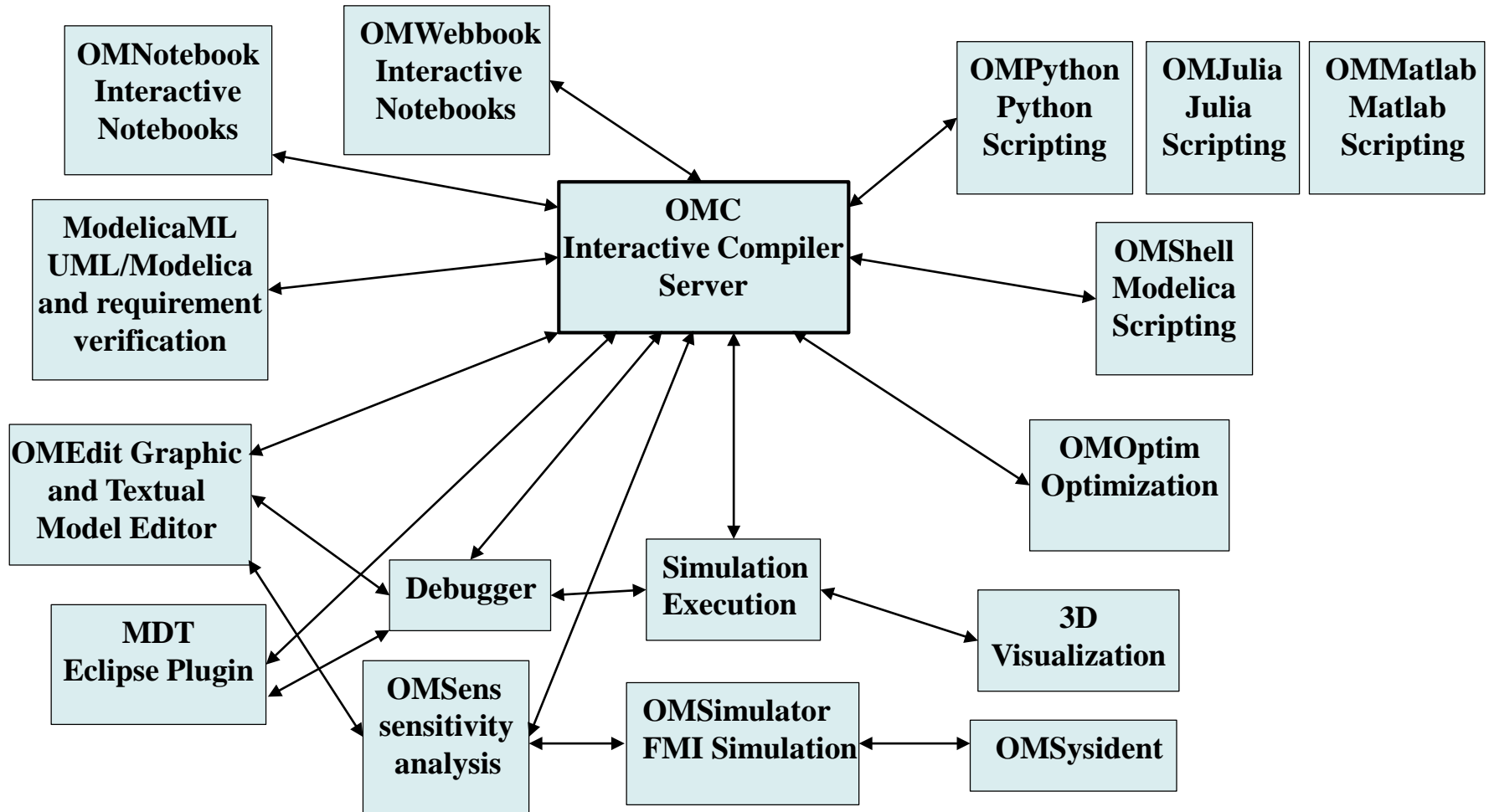
- ABB AB, Sweden
- Bosch Rexroth AG, Germany
- CDAC Centre, Kerala, India
- Creative Connections, Prague
- DHI, Aarhus, Denmark
- Dynamica s.r.l., Cremona, Italy
- EDF, Paris, France
- Equa Simulation AB, Sweden
- Fraunhofer IWES, Bremerhaven
- Fraunhofer FCC, Gothenburg
- INRIA, Rennes, France
- ISID Dentsu, Tokyo, Japan
- Juelich, FZI, Germany
- Maplesoft, Canada
- Metroscope, France
- REUSE company, Spain
- RISE, Sweden
- RTE France, Paris, France
- Saab AB, Linköping, Sweden
- SmartFluidPower, Italy,
- TLK Thermo, Germany
- Sozhou Tongyuan, China
- SRON Space Ins Netherlands
- Talent Swarm, Spain
- Volvo Cars, Sweden
- VTI, Linköping, Sweden
- XRG Simulation, Germany

## University members

- Augsburg University, Germany
- Australian Nation Univ., Australia
- FH Bielefeld, Bielefeld, Germany
- University of Bolivar, Colombia
- TU Braunschweig, Germany
- Chalmers Univ, Control, Sweden
- Chalmers Univ, Machine, Sweden
- TU Darmstadt, Germany
- TU Delft, The Netherlands
- TU Dresden, Germany
- Université Laval, Canada
- Ghent University, Belgium
- Halmstad University, Sweden
- TU Hamburg/Harburg Germany
- IIT Bombay, Mumbai, India
- Linneaus University, Sweden
- Linköping University, Sweden
- Univ of Maryland, Syst Eng USA
- Univ of Maryland, CEEE, USA
- Politecnico di Milano, Italy
- Politecnica Catalunya Spain
- Mälardalen University, Sweden
- RPI, Troy, USA
- Univ Pisa, Italy
- Univ College SouthEast Norway
- Vanderbilt Univ, USA



# The OpenModelica Tool Architecture






# Build System with Regression Testing

---

- Automatic Nightly build system (using Jenkins), and several multi-core computers
- Regression testing of libraries
- Verification testing comparing results to references

# Spoken-Tutorial step-by-step OpenModelica and Modelica Tutorial Using OMEdit. Link from [www.openmodelica.org](http://www.openmodelica.org)



Login Create an account

Spoken Tutorial Software Training Creation News Forums About Statistics

HOME DOWNLOAD TOOLS & APPS

Search Tutorials

[https://spoken-tutorial.org/tutorial-search/?search\\_foss=OpenModelica&search\\_language=English](https://spoken-tutorial.org/tutorial-search/?search_foss=OpenModelica&search_language=English)

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


04)

Submit

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

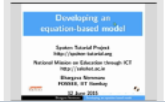
Outline: Introduction to OpenModelica Introduction to OMEdit Perspectives in OMEdit Browsers in OMEdit View icons in OMEdit Open a Class from Libraries Browser Checking for correctness..



### 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 ..




### 3. Developing an equation-based model

Foss : OpenModelica - English

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

70 Copyright © Open Source Modelica Consortium Usage: Creative Commons with attribution CC-BY



# OMNotebook Electronic Notebook with DrModelica

- Primarily for teaching
- Interactive electronic book
- Platform independent
- Also support for Jupyter notebooks

## 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)



# 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 difference between the measurement and the estimation. If the estimation is correct then the difference should be zero.

Another difficulty is that the measured quantities of

$$\begin{Bmatrix} \hat{x} \\ \hat{y} \end{Bmatrix} =$$

Here  $e$  denotes a disturbance in the input signal. It can be evaluated by the difference

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

By using this quantity as feedback we obtain the observer

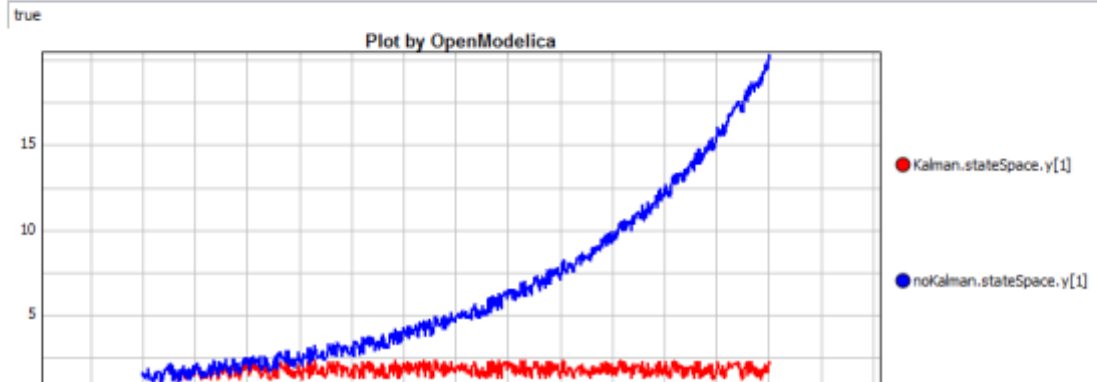
$$\dot{\hat{x}} = A\hat{x}(t) + Bu(t) + K(y(t) - \hat{y}(t))$$

Now form the error as

The differential error is

```
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*L;
  parameter Real[:,:] BL = B*L;
  parameter Real[:,:] Z = zeros(size(ABL,2),size(AKC,1));
  parameter Real[:,:] AKC = 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=Anew,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]})
```



Ready Ln 12, Col 39

MODELICA

# 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 formatting. The main content area displays a document with the following text:

**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 -> products

Consider a chemical reaction between hydrogen gas and iodine gas to form hydrogen iodine:  
$$\text{H}_2 + \text{I}_2 \rightleftharpoons 2\text{HI}$$

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

$$\frac{d}{dt} [\text{H}_2] = k_2 \cdot [\text{HI}^2] - k_1 \cdot [\text{H}_2] \cdot [\text{I}_2]$$
$$\frac{d}{dt} [\text{I}_2] = k_2 \cdot [\text{HI}^2] - k_1 \cdot [\text{H}_2] \cdot [\text{I}_2]$$
$$\frac{d}{dt} [\text{HI}] = 2k_2 \cdot [\text{H}_2] \cdot [\text{I}_2] - 2k_1 \cdot [\text{HI}^2]$$

```
\documentclass[12pt]{article}
\begin{document}
\thispagestyle{empty}
\[\frac{\mathrm{d}}{\mathrm{d}t}[\mathrm{H}_2] = k_2 \cdot [\mathrm{HI}^2] - k_1 \cdot [\mathrm{H}_2] \cdot [\mathrm{I}_2]\]
\[\frac{\mathrm{d}}{\mathrm{d}t}[\mathrm{I}_2] = k_2 \cdot [\mathrm{HI}^2] - k_1 \cdot [\mathrm{H}_2] \cdot [\mathrm{I}_2]\]
\[\frac{\mathrm{d}}{\mathrm{d}t}[\mathrm{HI}] = 2k_2 \cdot [\mathrm{H}_2] \cdot [\mathrm{I}_2] - 2k_1 \cdot [\mathrm{HI}^2]\]
\end{document}
```

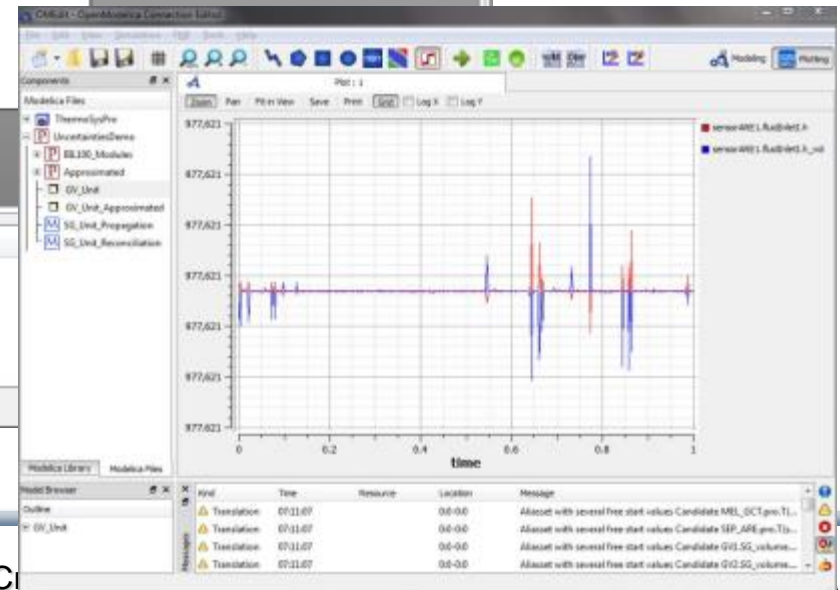
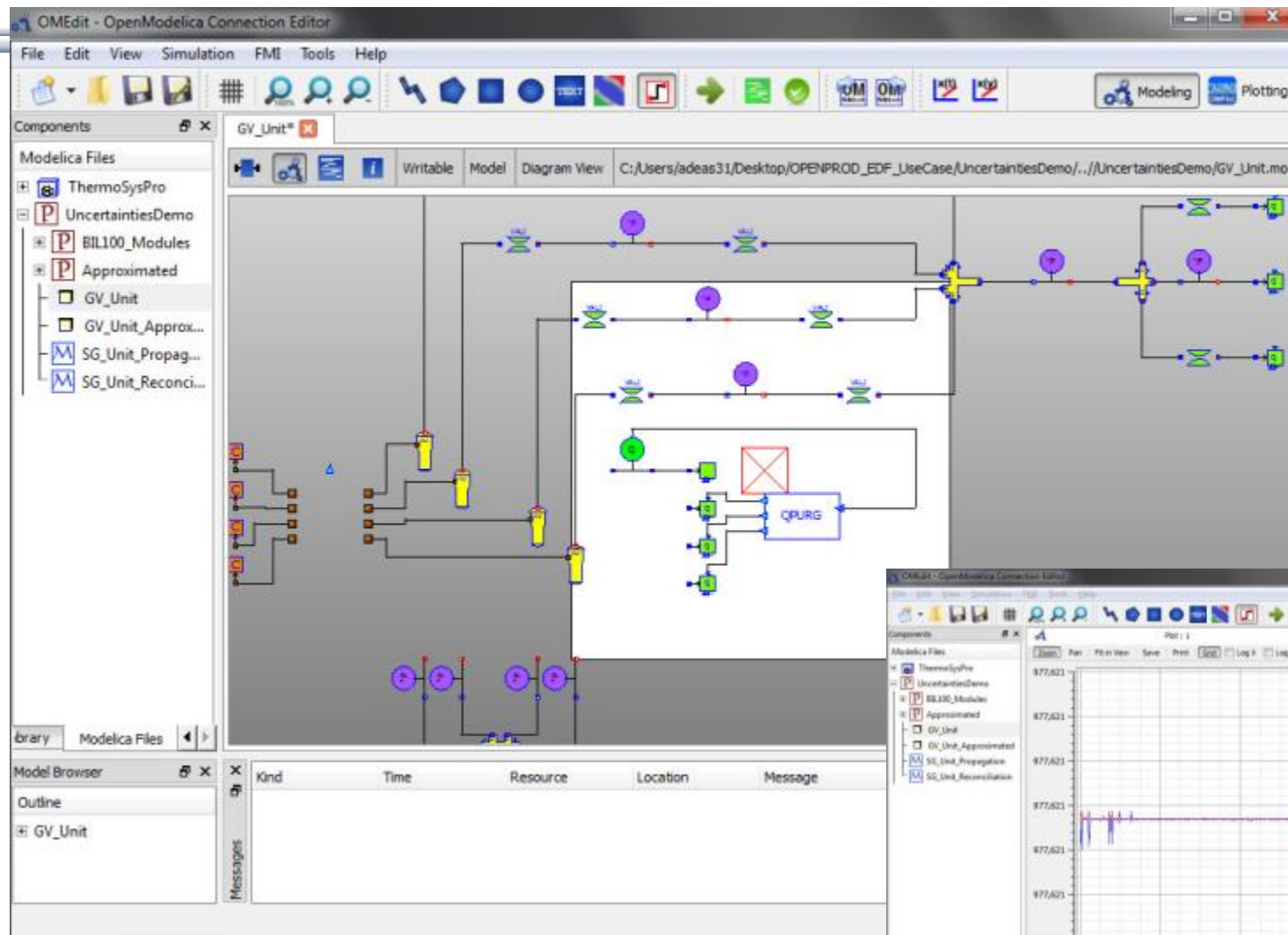
The status bar at the bottom right shows "Ready".

Contents in  
OMWebbook  
Generated from  
OMNotebook

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



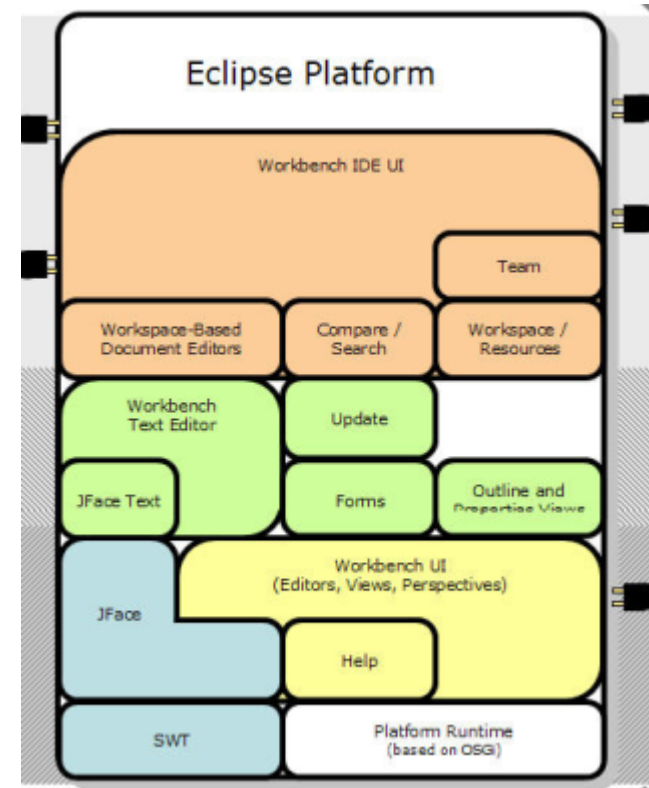
# OpenModelica Environment Demo





# 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 OpenModelica MDT Eclipse IDE interface. The top toolbar includes standard Eclipse actions like File, Edit, Navigate, Search, Project, Run, Field Assist, Window, and Help. The left sidebar shows the 'Modelica Projects' tree with a list of files including rml2sig, runtime, scripts, test\_codegen, tools, VC7, and Absyn.mo. The main editor window shows the 'Absyn.mo' file with a code snippet. A yellow tooltip is visible over the 'getCrefFromExp' function, providing a description: 'Returns a flattened list of the component references in an expression'. The bottom-left pane shows the 'Outline' view with a list of code elements under the 'Absyn' package, 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 bottom-right pane shows the 'Problems' view with a list of errors, including 'The identifier at start and end are different'. A blue callout box with white text points to the 'Identifier Info on Hovering' feature. Another blue callout box with white text points to the 'Code Outline for easy navigation within Modelica files' feature.

Modelica - OpenModelica/Compiler/Absyn.mo - Eclipse SDK

File Edit Navigate Search Project Run Field Assist Window Help

Modelica Projects

- rml2sig
- runtime
- scripts
- test\_codegen
- tools
- VC7
- Absyn.mo 3116 2008-02-04 14:44 krsta
- Absyn 3116 2008-02-04 14:44 krsta
- Algorithm.mo 2992 2007-12-22 22:17 adrho
- Builtin.mo 3585 2008-05-22 07:03 adrho
- Ceval.mo 3605 2008-05-27 02:48 adrho
- ClassInf.mo 3496 2008-04-23 11:59 krsta
- ClassLoader.mo 3193 2008-02-15 05:17 adrho
- Codegen.mo 3585 2008-05-22 07:03 adrho
- Connect.mo 3584 2008-05-22 06:45 adrho
- Constants.mo 3011 2007-12-22 22:36 adrho
- Convert.mo 3496 2008-04-23 11:59 krsta

Absyn.mo

```
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
    11 = getCrefFromExp(e1);
    12 =
      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
            11 =
              ComponentRef cr;
            12 =
              listAppend(11, 12);
          then
            res = listAppend(11, 12);
          then
```

Identifier Info on Hovering

Code Outline for easy navigation within Modelica files

Outline

- Absyn
- ADD
- ALG\_ASSIGN(Exp assignComponent, Exp value)
- ALG\_BREAK
- ALG\_CATCH(list<AlgorithmItem> catchBody)
- ALG\_EQUALITY(Algorithm equ)
- ALG\_FAILURE(Algorithm equ)
- ALG\_FOR(ForIterators iterators, list<AlgorithmItem> forBo
- ALG\_GOTO(String labelName)
- ALG\_IF(Exp ifExp, list<AlgorithmItem>
- ALG\_LABEL(String labelName)
- ALG\_NORET\_CALL(ComponentRef
- ALG\_RETURN
- ALG\_THROW
- ALG\_TRY(list<AlgorithmItem> tryBody
- ALG\_WHEN\_A(Exp whenStmt, list<Alg

Problems

113 errors, 0 warnings, 0 infos

Description

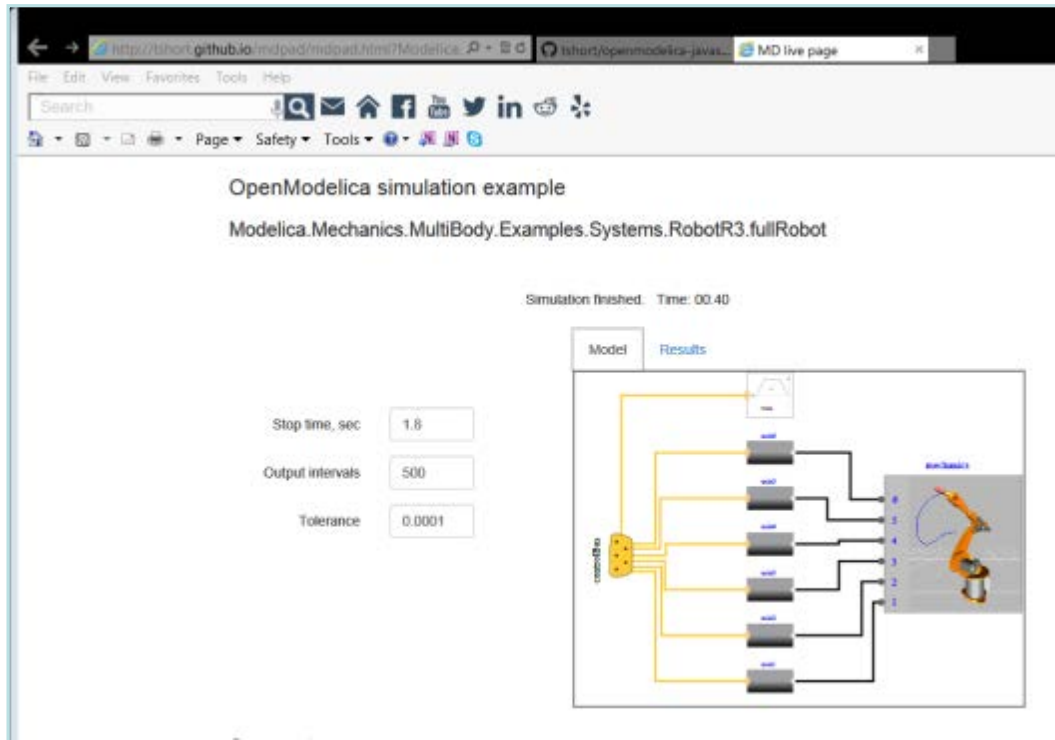
- Errors (100 of 113 items)
- The identifier at start and end are different
- The identifier at start and end are different
- The identifier at start and end are different, par

ken'; on line	rml2mod.mo	OpenModelica/tools/rml2mod
ken'; on line	rml2mod.mo	OpenModelica/tools/rml2mod
ken'; on line	modrefacto...	OpenModelica/tools/rml2mod
ken'; on line	moddump.mo	OpenModelica/tools/rml2mod
ken'; on line	rml2mod.mo	OpenModelica/tools/rml2mod
ken'; on line	rml2mod.mo	OpenModelica/tools/rml2mod
ken'; on line	rml2mod.mo	OpenModelica/tools/rml2mod

64M of 254M

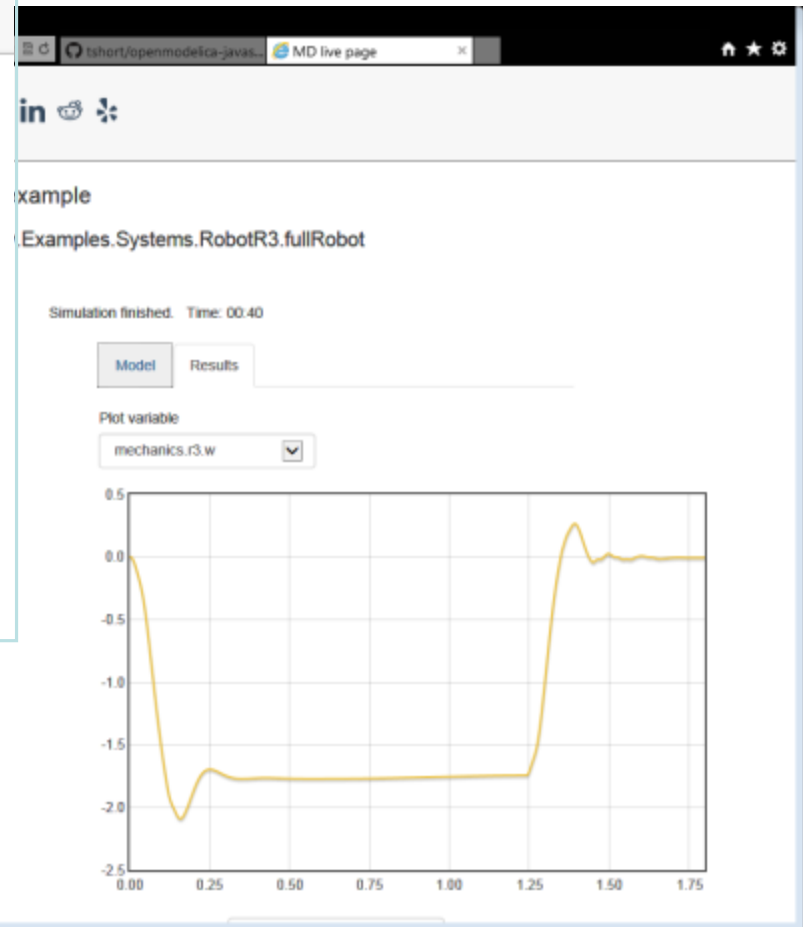
Ctrl Contrib (Bottom)

# OpenModelica Simulation in Web Browser Client



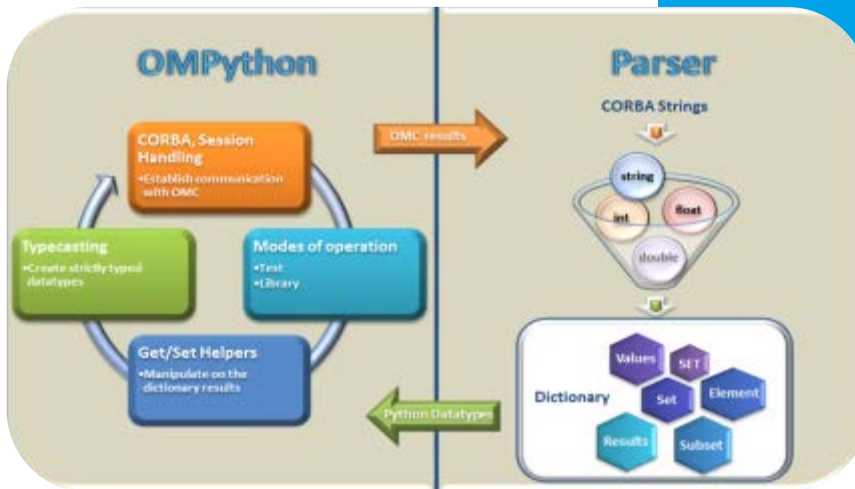
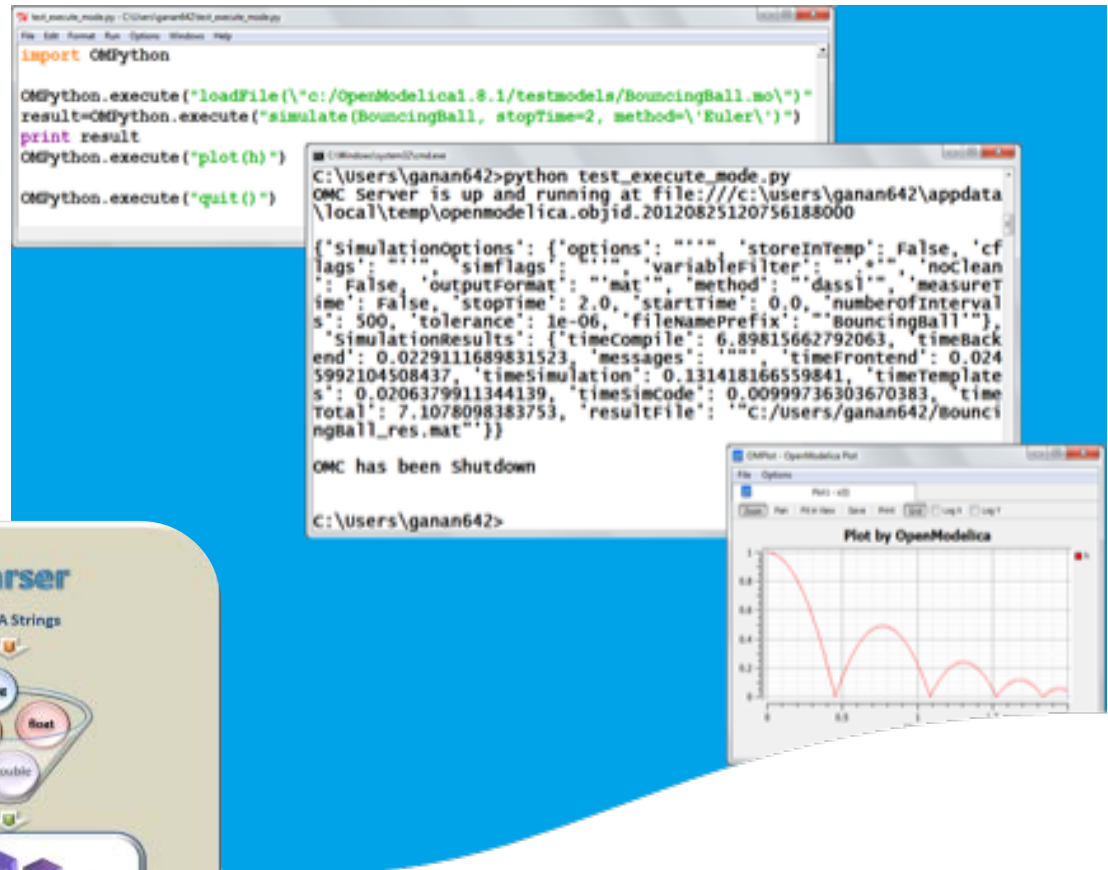
**OpenModelica compiles to efficient JavaScript code which is executed in the web browser**

## MultiBody RobotR3.FullRobot



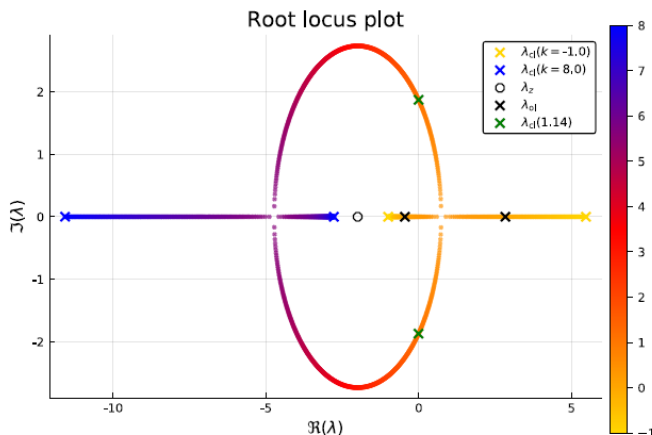
# OMPython – Python Scripting with OpenModelica

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



# OMJulia – Julia Scripting with OpenModelica

- Interpretation of Modelica commands and expressions from Julia, transfer of data
- Control design using Julia control package together with OpenModelica
- Interactive Session handling
- Library / Tool
- Separately downloadable. be run with OpenModelica 1.13.2 or later
- Works with Jupyter notebooks



## Control example with OMJulia in Jupyter notebooks

### Use of Modelica + Julia in Process Systems Engineering Education

Complex models of "Seborg reactor"

Bernt Lie\*, Arunkumar Palanisamy\*\*, Peter Fritzson\*\*

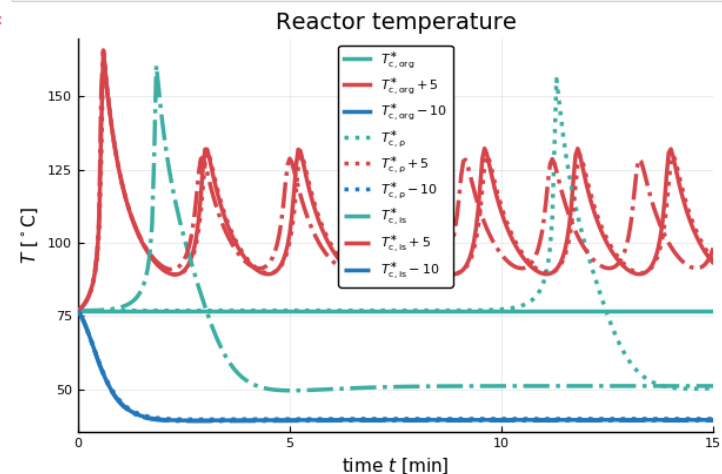
\*University of South-Eastern Norway, Norway

\*\*University of Linköping, Sweden

Introducing packages

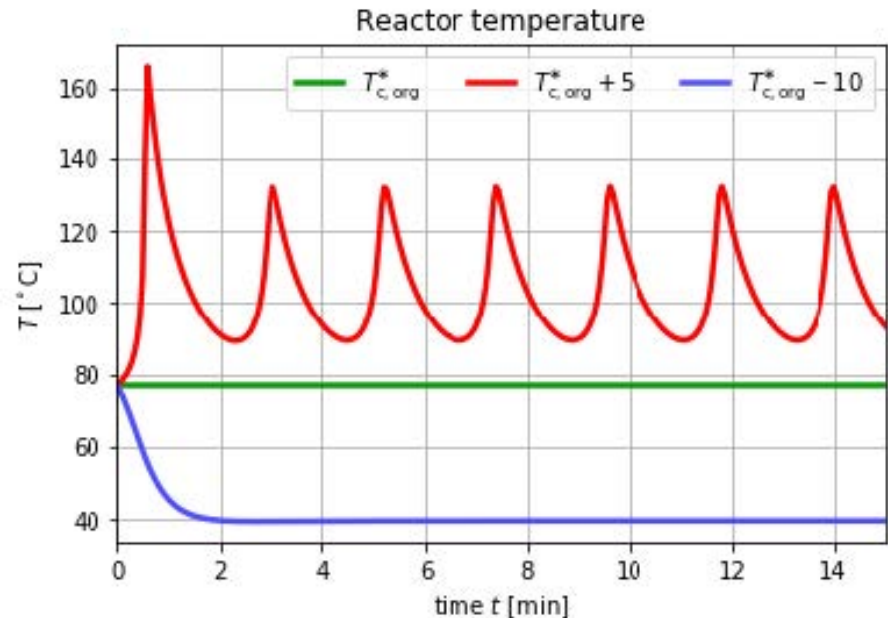
```
In [1]: # Pkg.add("Plots") -- we assume that this step already has been carried out
using Plots; pyplot()
using LaTeXStrings
using DataFrames
using OMJulia
#using DifferentialEquations
```

Out[9]:



# OMMatlab – Matlab Scripting with OpenModelica

- Interpretation of Modelica commands and expressions from Matlab, transfer of data
- Interactive Session handling
- Library / Tool
- Separately downloadable. be run with OpenModelica
- Similar API functions as in OMJulia and OMPython
- Can be used for control design from Matlab





# Experimental OpenModelica Compiler in Julia

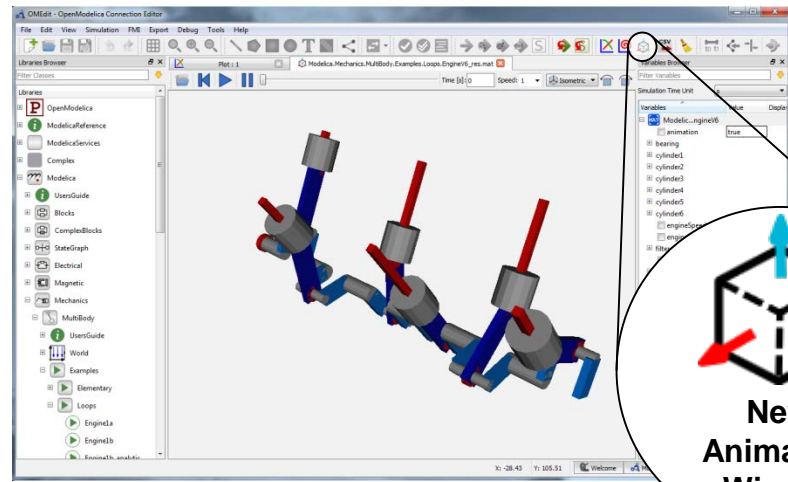
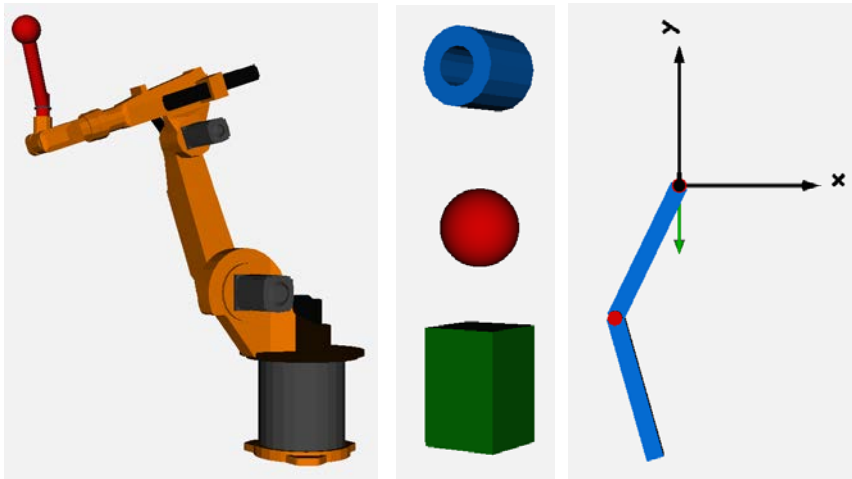
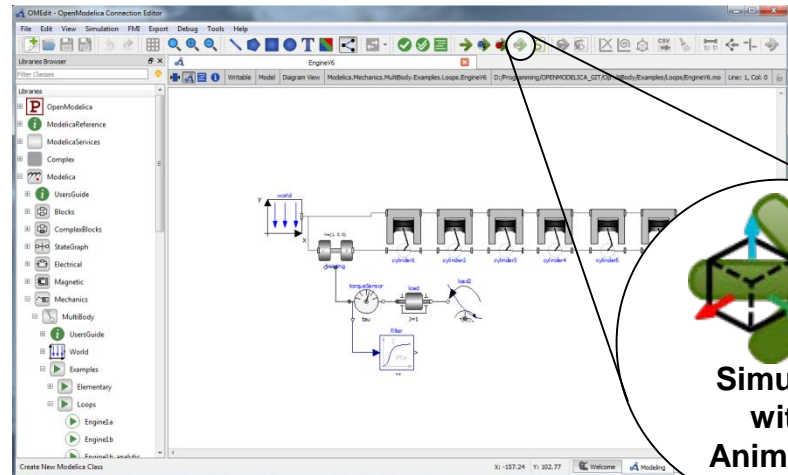
John Tinnerholm and Adrian Pop)

- OpenModelica.jl: modular and extensible Modelica compiler framework in Julia
- Developed a preliminary MetaModelica to Julia translator
- **Translated the high-performance frontend.**
- **Able to execute and translate MetaModelica functions**
- **Able to simulate discrete-hybrid systems + regular continuous systems**
- **Experimental backends developed**
  - Targeting DifferentialEquations.jl and ModelingToolkit.jl (MTK)
  - Completed causalization sorting, matching.
  - Integrated LightGraphs.jl package, DAG representation of the hybrid DAE
  - Integrated Plots.jl for interactive plotting and animation
  - Integrated the Reduce Computer Algebra system for automatic symbolic manipulation and symbolic derivation.
  - Integration with Sundials. IDAS used for numerical integration
- Further performance **tuning needed**
- Currently experimenting with **variable-structure systems**



# OMEdit 3D Visualization of Multi-Body Systems

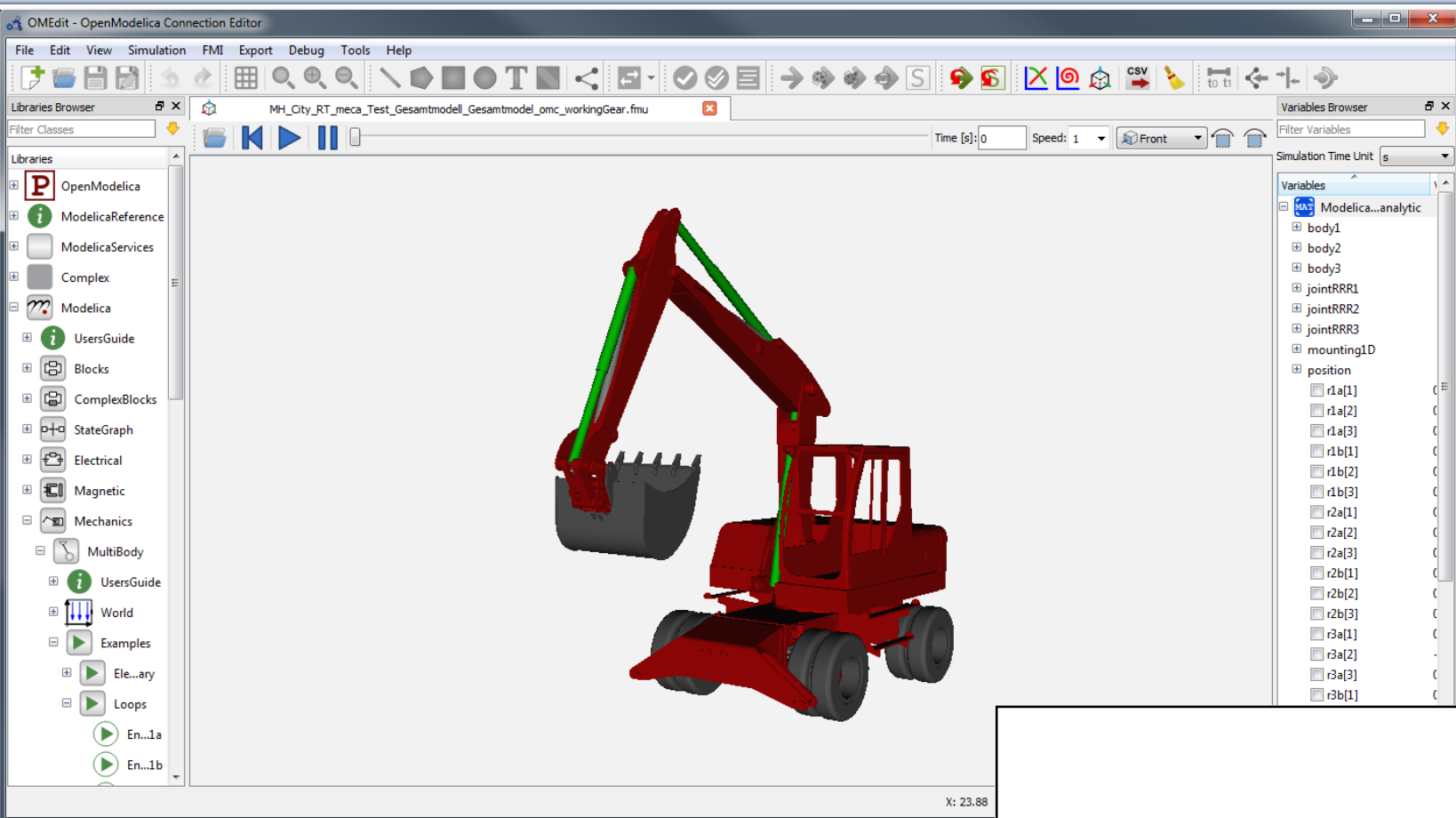
- Built-in feature of OMEdit to animate MSL-Multi-Body shapes
- Visualization of simulation results
- Animation of geometric primitives and CAD-Files



# OpenModelica 3D Animation Demo (V6Engine and Excavator)



# OpenModelica 3D Animation – Excavator



## Connection with Unity

Courtesy of Volker Waurich - TU Dresden



# OpenModelica 3D Animation – WheelLoader



# OpenModelica 3D Animation – BouncingBall



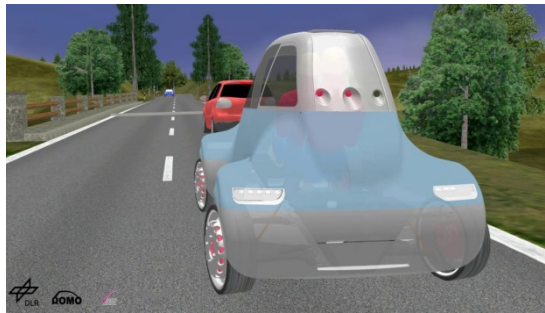
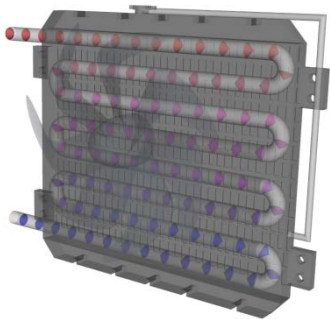
Collision detection in Unity

Courtesy of Volker Waurich - TU Dresden

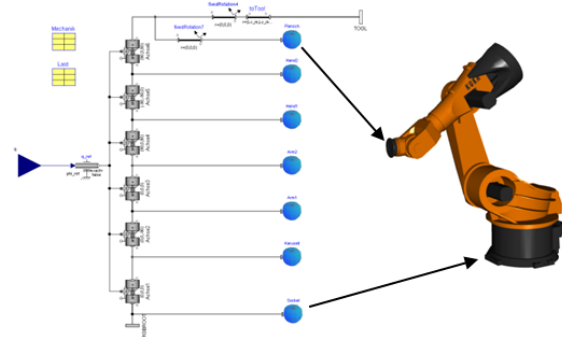
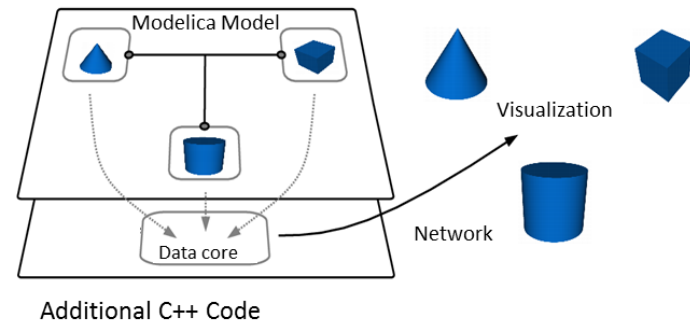


# Visualization using Third-Party Libraries: DLR Visualization Library

- Advanced, model-integrated and vendor-unspecific visualization tool for Modelica models
- Offline, online and real-time animation
- Video-export function
- Commercial library, feature reduced free Community Edition exists



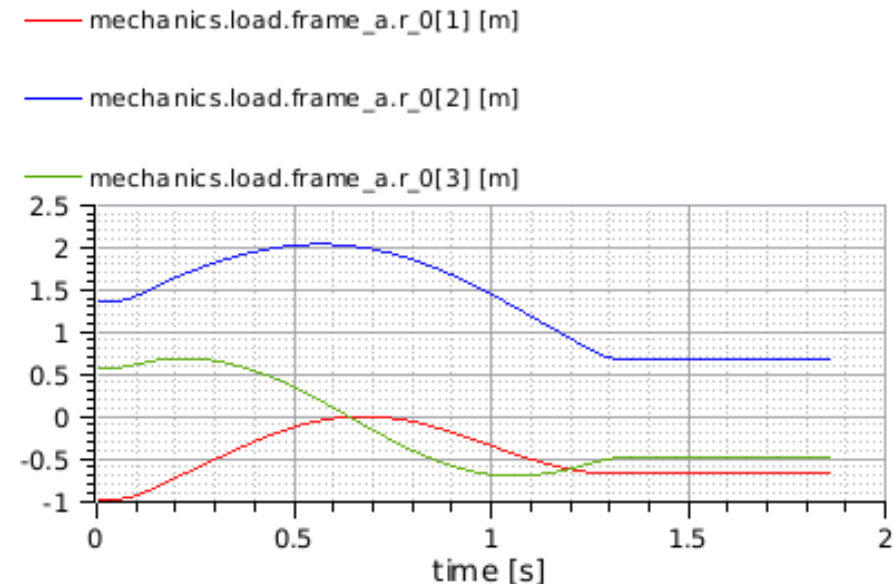
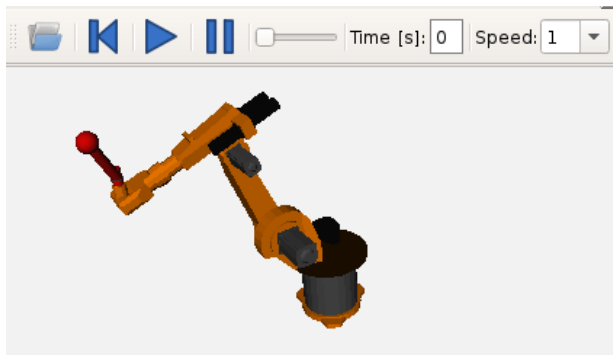
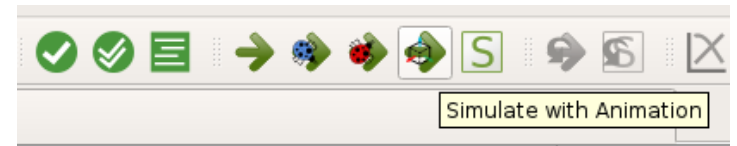
Integration of visualizer blocks into the model and Communication to an external viewer (SimVis)



Courtesy of Dr. Tobias Bellmann (DLR)

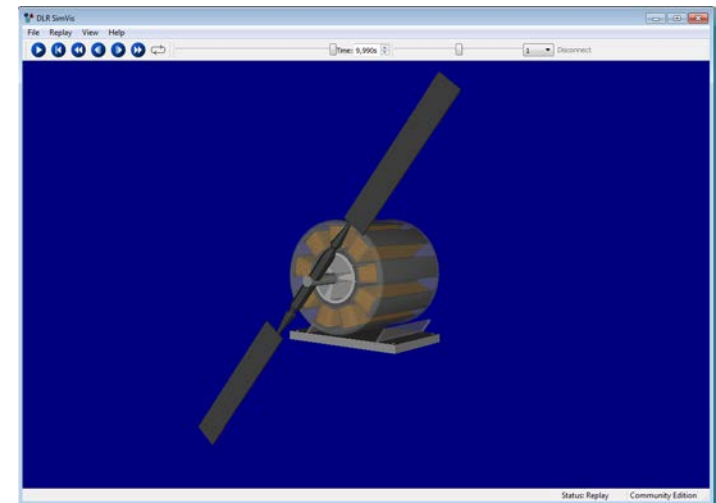
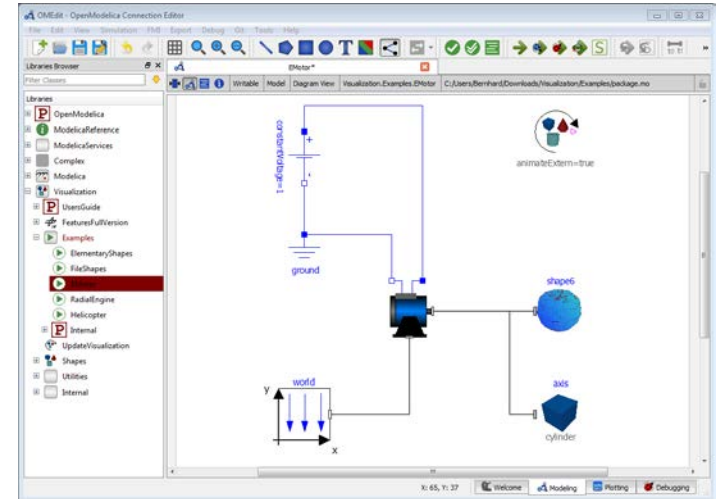
# Exercise 1.2: Use 3D Visualization for Robot model

- Open the Modelica.Mechanics.MultiBody.Examples.Systems.RobotR3.fullRobot example in OMEdit
- Press Simulate with Animation
- Replay the animation
- Compare with the plot



# Exercise 1.3: Visualization using the DLR Visualization Community Edition (1)

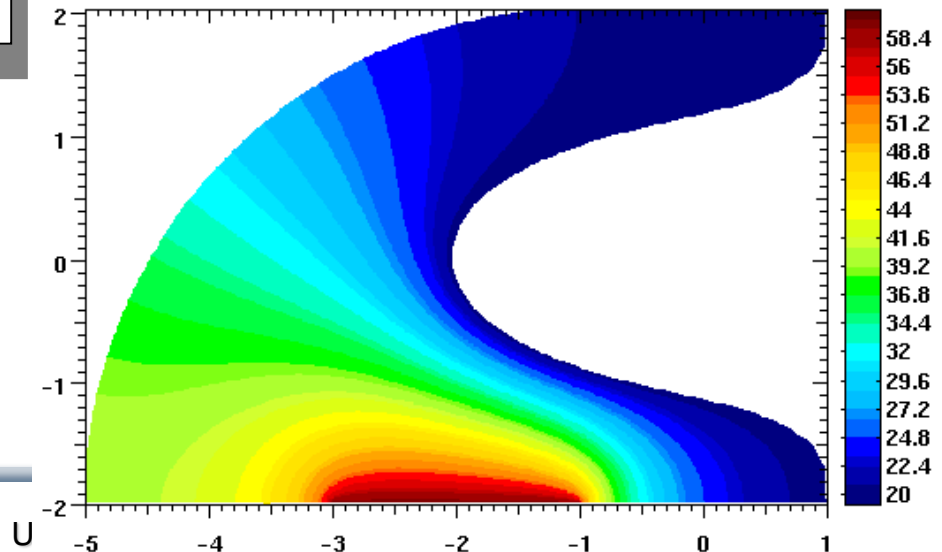
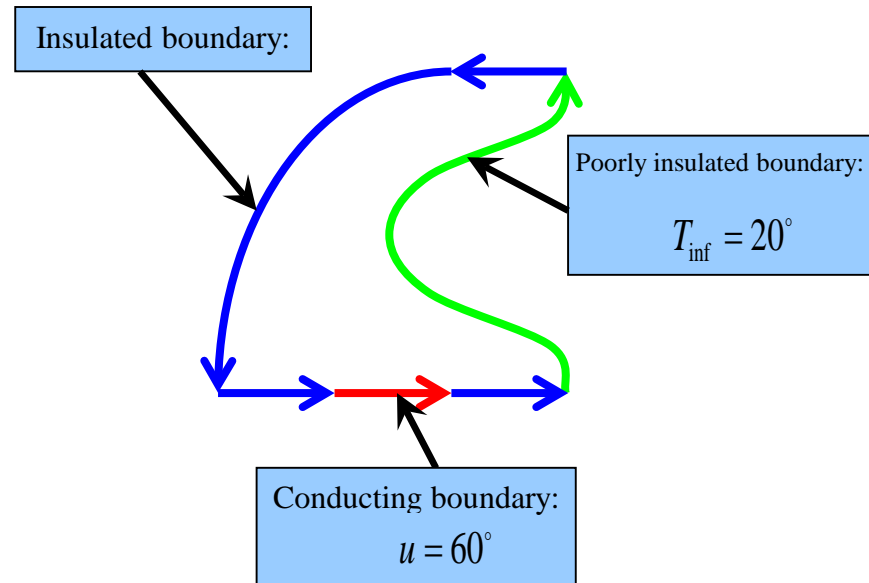
- Unpack VisualizationCommunityEdition.zip
- Open the library in OMEdit
- Simulate the EMotor example
- The DLR SimVis visualization app should start automatically
- Export the animation (File→Export Replay as Video)



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

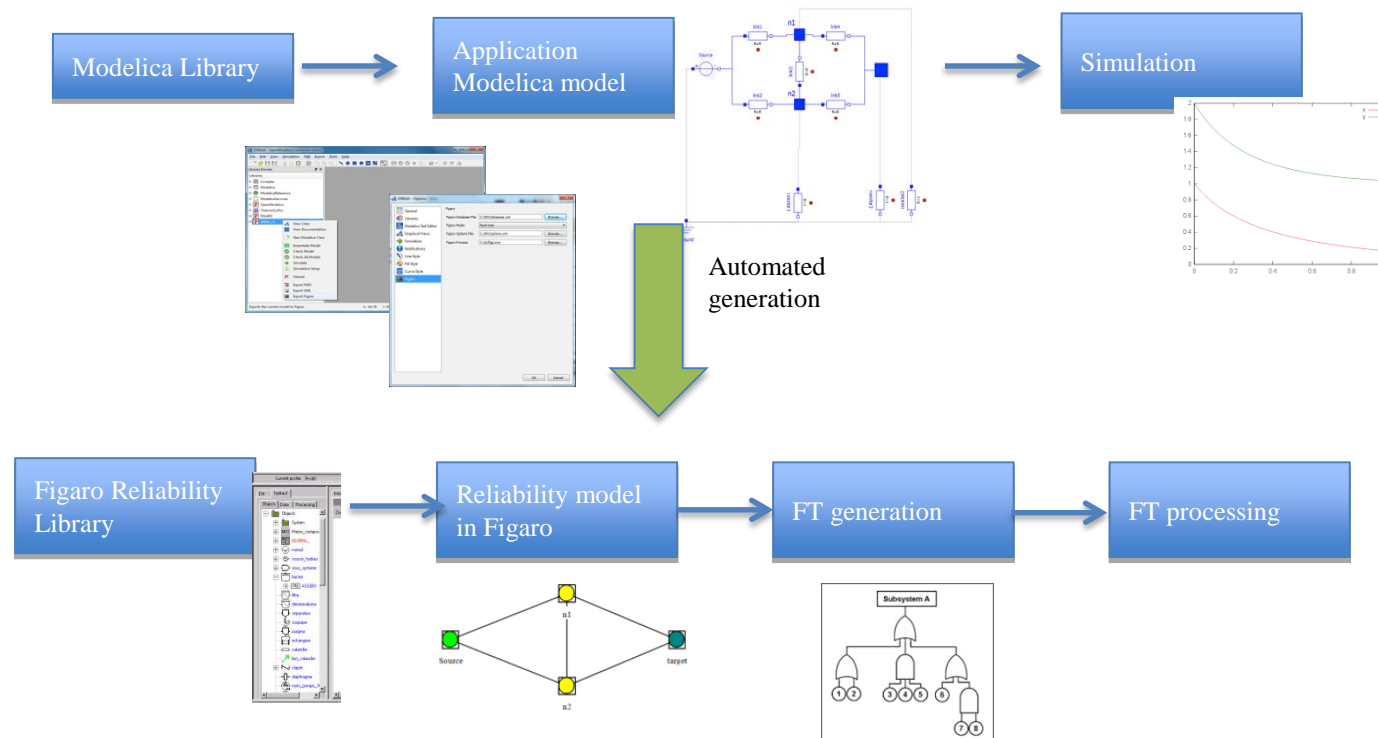
```
class PDEModel
  HeatNeumann h_iso;
  Dirichlet h_heated(g=50);
  HeatRobin h_glass(h_heat=30000);
  HeatTransfer ht;
  Rectangle2D dom;
equation
  dom.eq=ht;
  dom.left.bc=h_glass;
  dom.top.bc=h_iso;
  dom.right.bc=h_iso;
  dom.bottom.bc=h_heated;
end PDEModel;
```

Prototype in OpenModelica 2005  
PhD Thesis by Levon Saldamli  
[www.openmodelica.org](http://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.



# OMOptim – Optimization (1)

Model structure

Model Variables

Optimized parameters

Optimized Objectives

MinEIT

File Project Problem Display Tools

Models Problems

Name

- Pc
- Va
- Vb
- Ia
- Ib
- Ic
- Ea
- Eb
- Ec
- coutdinvestissement
- gaincoperationnel
- EmCO2PAC1
- Ca
- Cb
- Cc
- Puissae
- Puissbe
- Puissce
- n
- na
- nb
- nc
- OCb
- OChp
- coutdefonctavecPAC
- T0SygmaA
- T0SygmaB
- T0SygmaECS
- COPECSSystem
- PElecECSMax
- ⊕ EchIAOutCold
- ⊕ Sortieeffluents
- ⊕ echA
- ⊕ Sourcemod
- ⊕ scenarioEchA
- ⊕ scenarioPACA
- ⊕ echB

Project Optimization EI EI result

Variables

Filter :

Name	Value	Description
global.sourceeadeville.h	1,18294e+06	[J/kg]
global.sourceeadeville.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.scenariosourceEadeville.debit	0,940001	[kg/s]
global.scenariodepartB.z	0	

Optimized variables

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

Scanned variables

Name	Description	Scan Minimum	Scan Maximum	n
------	-------------	--------------	--------------	---

Optimization objectives

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

Variables Components Launch



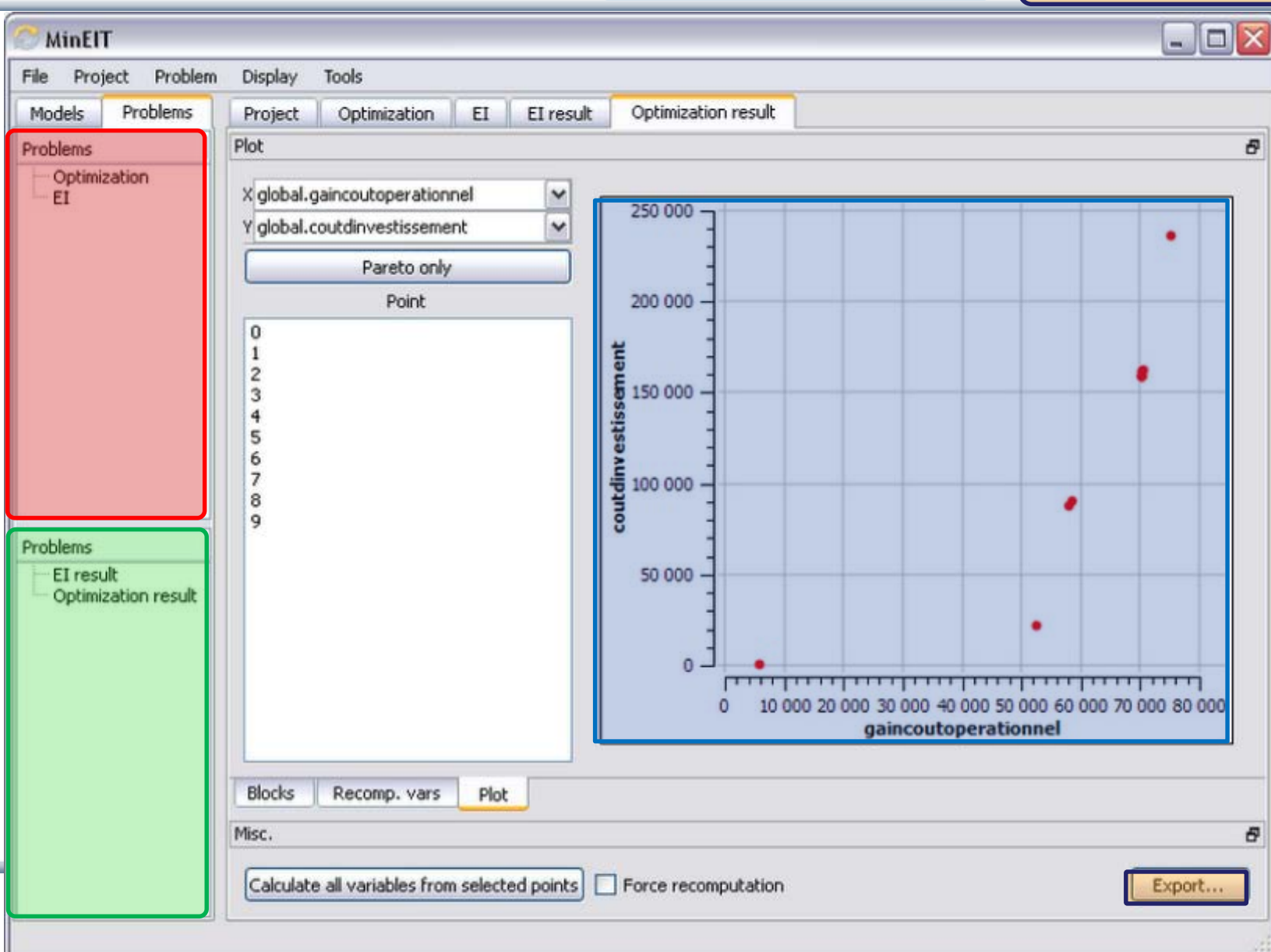
Problems

# OMOptim – Optimization (2)

Solved problems

Result plot

Export result data .csv

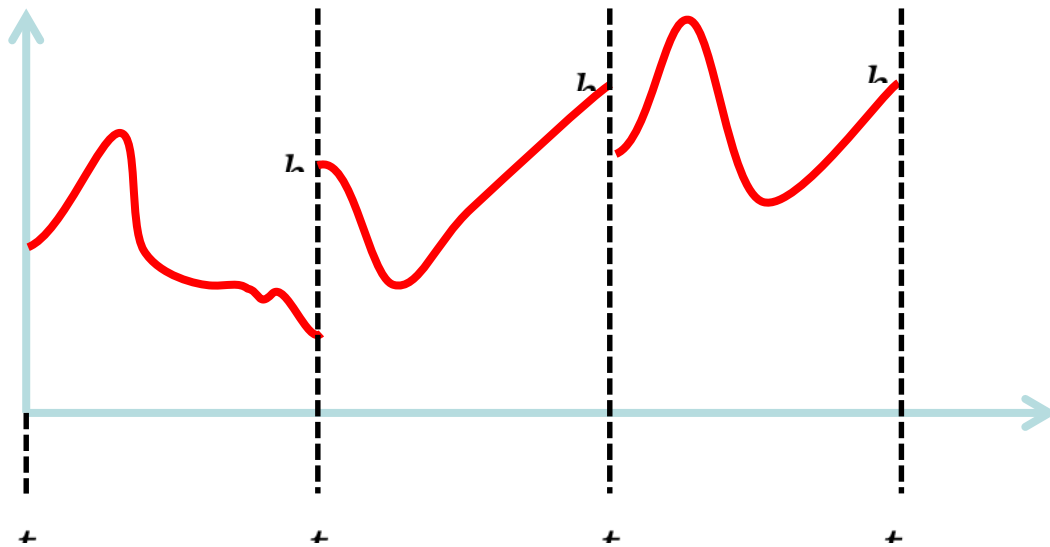


# Multiple-Shooting and Collocation

## Dynamic Trajectory Optimization

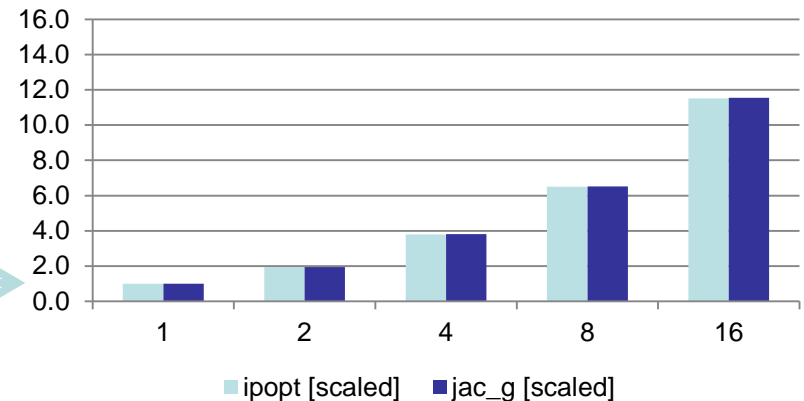
- Minimize a goal function subject to model equation constraints, useful e.g. for NMPC
- Multiple Shooting/Collocation
  - Solve sub-problem in each sub-interval

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

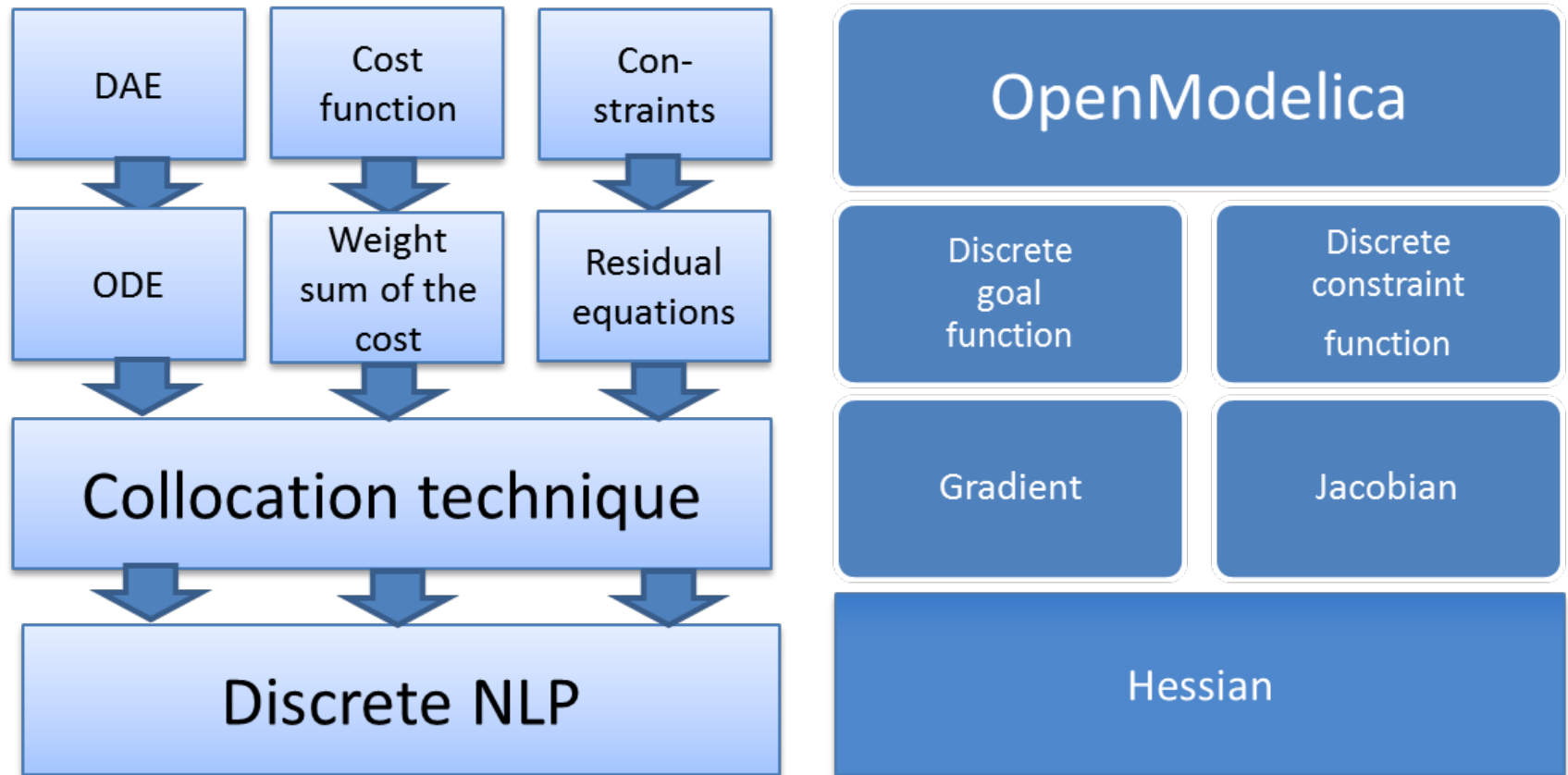


Example speedup, 16 cores:

**MULTIPLE\_COLLOCATION**



# OpenModelica Dynamic Optimization Collocation



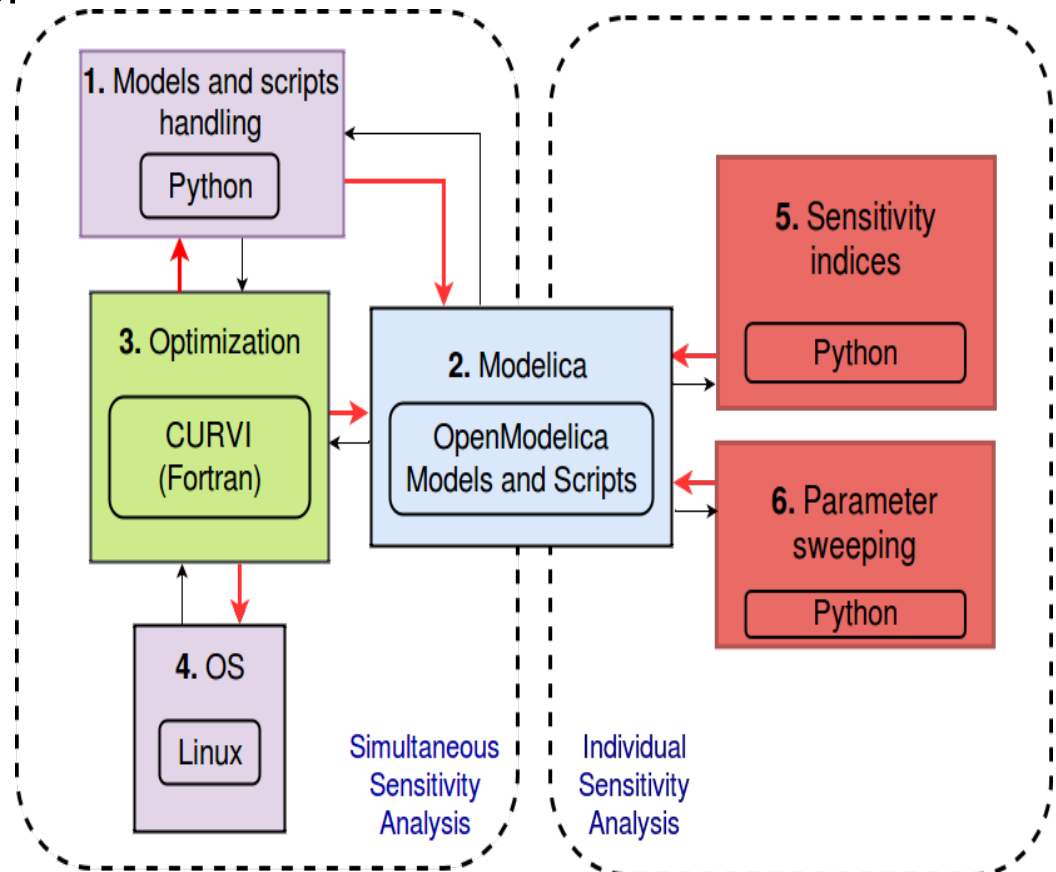
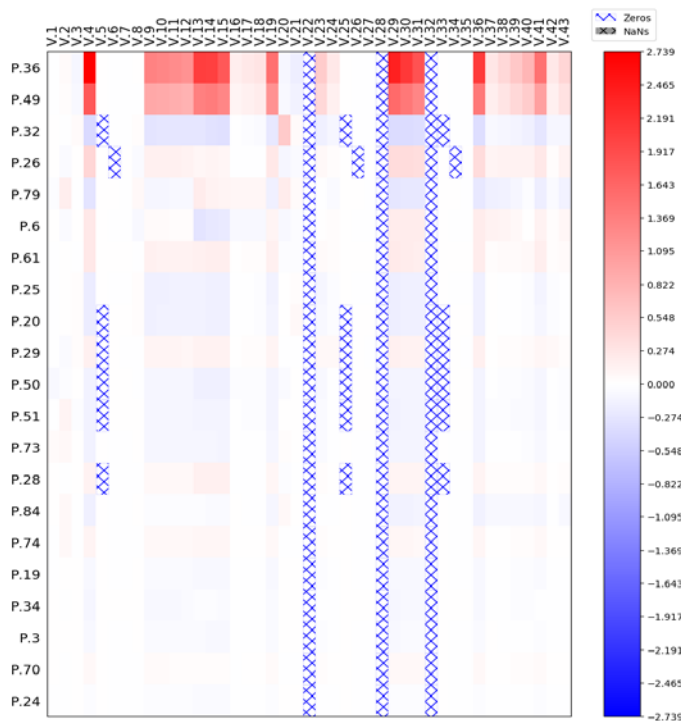
# OMSens – Multi-Parameter Sensitivity Analysis

- Individual and simultaneous multi-parameter analysis
- Optimization-based simultaneous analysis
- Robust derivative free optimizer

For an exercise, see further in these slides

## Tool architecture

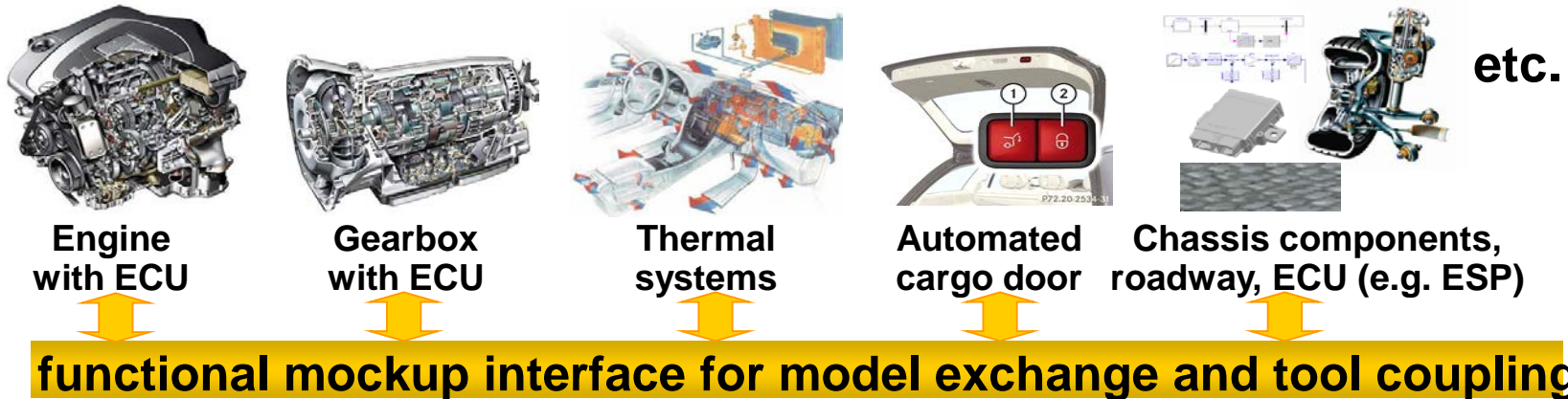
## Heatmap visualization



# OMSysIdent – System Parameter Identification

- OMSysIdent is a module for parameter estimation of behavioral models (wrapped as FMUs) on top of the OMSimulator API.
- Identification of the parameter values is typically based on measurement data
- It uses the Ceres solver (<http://ceres-solver.org/>) for the optimization task.

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



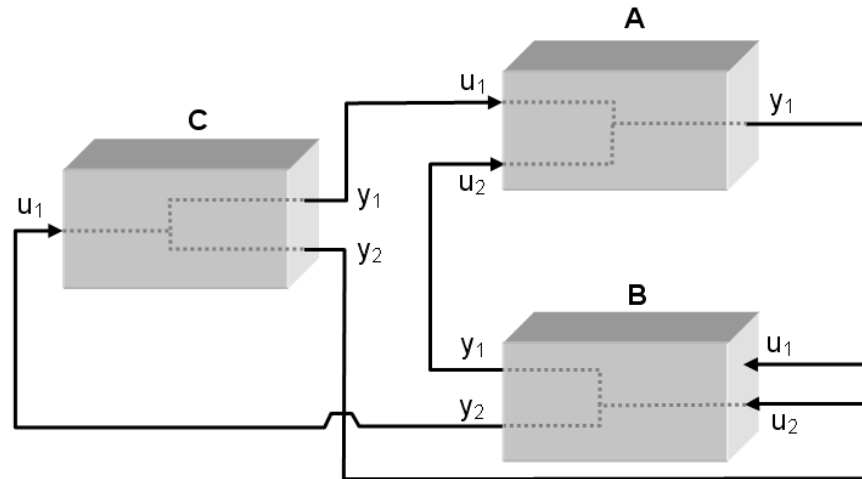
courtesy Daimler

- FMI development was started by ITEA2 MODELISAR project. FMI is now a Modelica Association Project
- **Version 1.0**
- FMI for Model Exchange (released Jan 26, 2010)
- FMI for Co-Simulation (released Oct 12, 2010)
- **Version 2.0** (released July 25 2014) **2.0.2** (released Dec 15, 2020)
- **Version 3.0** (Beta2 pre-release July 21 2021)
- FMI for Model Exchange and Co-Simulation
- ~ **150 tools** supporting it (<https://www.fmi-standard.org/tools>)

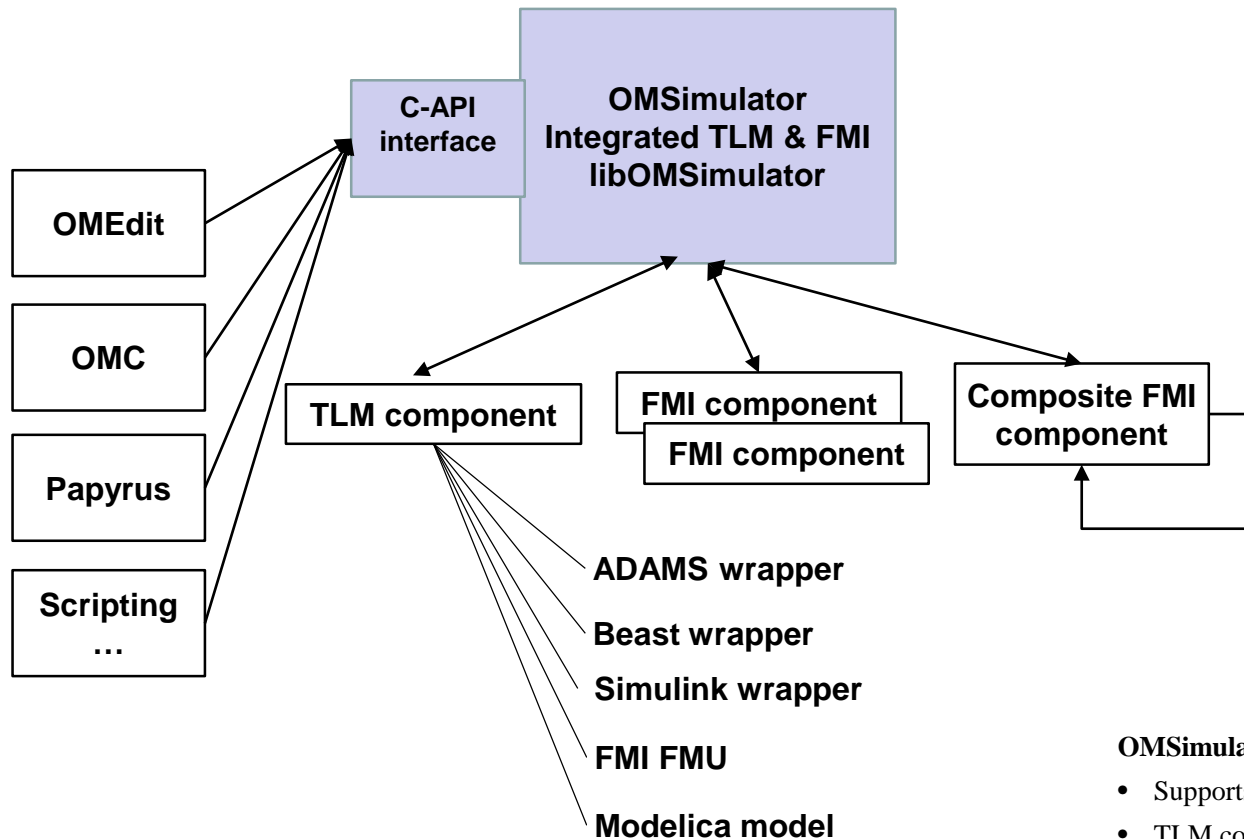


# Functional Mockup Units

- 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



# OMSimulator – Integrated FMI and TLM-based Cosimulator/Simulator – part of OpenModelica



## Main Framework Aspects

### Unified co-simulation/simulation tool

- FMI 2.0 (model exchange and co-simulation)
- TLM (transition line modelling)
- Real-time and offline simulation

### Standalone open source simulation tool with rich interfaces

- C/Java
- Scripting languages Python, Lua

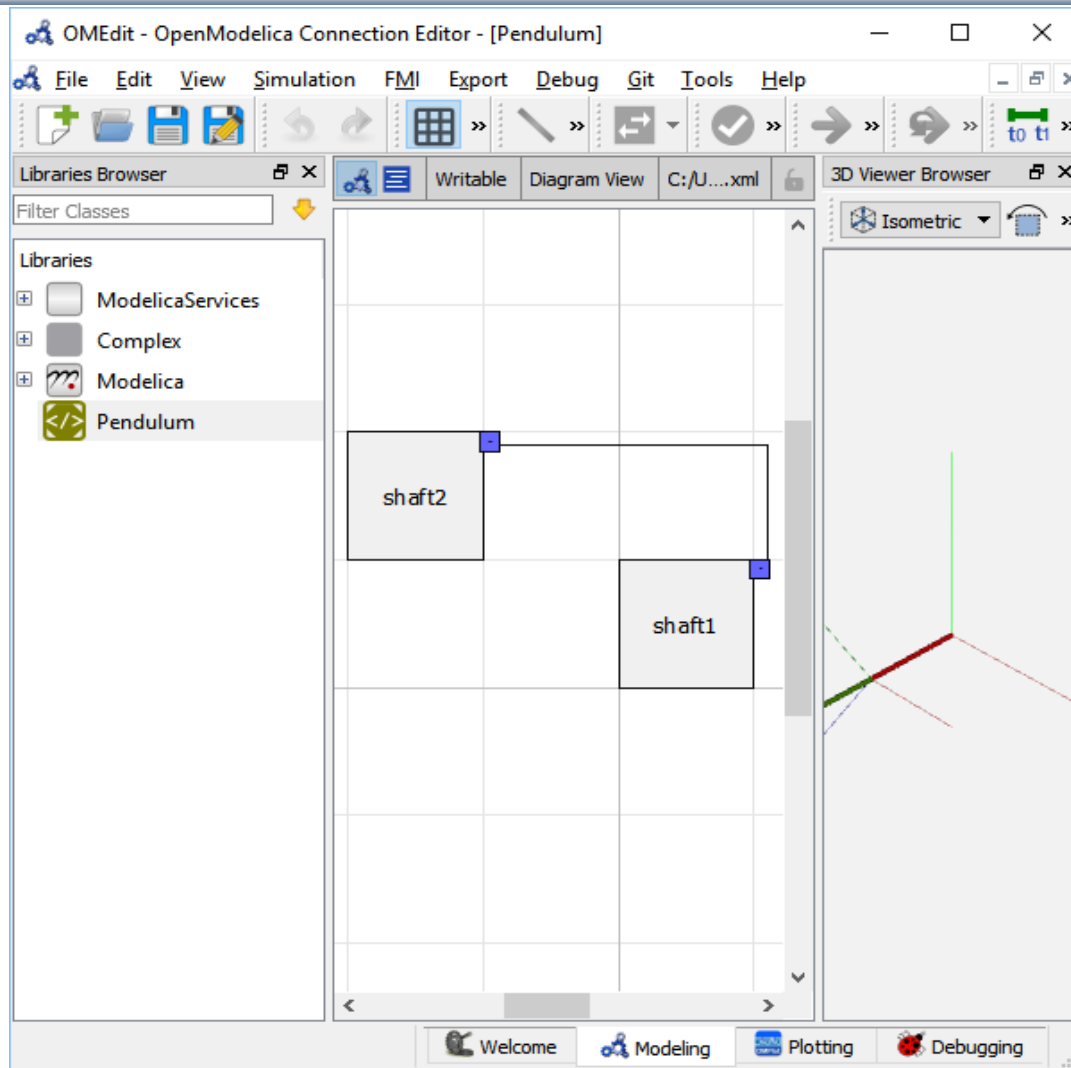
### Co-simulation framework as a solid base for engineering tools

- Integration into OpenModelica/Papyrus
- Open for integration into third-party tools and specialized applications (e.g. flight simulators, optimization)

### OMSimulator in OpenModelica 1.18.0

- Supports both FMI and TLM
- TLM connections are optional
- Co-simulation to multiple tools
- Composite model editor
- External API interface and scripting (C, Python, Lua)

# OMSimulator Composite Model Editor with 3D Viewer



- **Composite model editor** with 3D visualization of connected mechanical model components which can be FMUs, Modelica models, etc., or co-simulated components
- **3D animation** possible
- Composite model saved as SSP XML-file
- **Support for SSP** – System Structure and Parameterization standard
- **Numerically stable** co-simulation with **TLM**

# OMSimulator – GUI and SSP support

OMEdit - OpenModelica Connection Editor

File Edit View Simulation Debug SSP Sensitivity Optimization Tools Help

Libraries Browser

Filter Classes

Libraries

- > OpenModelica
- > ModelicaReference
- > ModelicaServices
- > Complex
- > Modelica
- > Model
  - > WC Root
    - > FMU DualMassOscillator\_System2
    - > FMU DualMassOscillator\_System1

Simulate Model

Diagram View D:/Model.ssp

100% Cancel Simulation

server.py --model=C:/Users/Andre/AppData/Local/Temp/OpenModelica/OMEdit//Model.ssp --endpoint-pub=tcp://

X: -211, Y: 103 Welcome Modeling Plotting Debugging

FMU1

FMU2

Creative Commons with attribution CC-BY

MODELICA

# OMSimulator Simulation, SSP, and Tool Comparison

## Adding SSP bus connections

OMEdit - Add Bus Connection

Add Bus Connection

Connect **bus2** input connectors to **bus1** output connectors

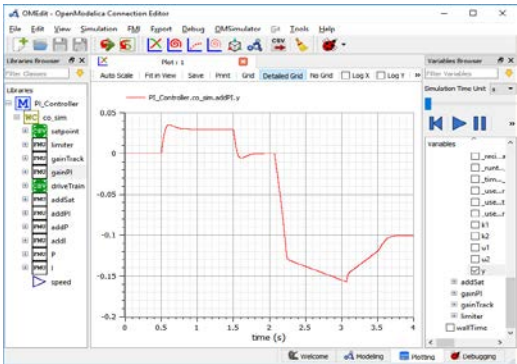
	bus2 inputs	bus1 outputs	ssid:Connection
1	<input checked="" type="checkbox"/> u1	y	<ssid:Con...t="sc2"
2	<input type="checkbox"/> u2		

Connect **bus2** output connectors to **bus1** input connectors

	bus2 outputs	bus1 inputs	ssid:Connection
1	<input checked="" type="checkbox"/> y1	u1	<ssid:Con...t="sc2"
2	<input checked="" type="checkbox"/> y2	u2	<ssid:Con...t="sc2"
3	<input type="checkbox"/> y3		

OKCancel

## FMI Simulation results in OMEdit

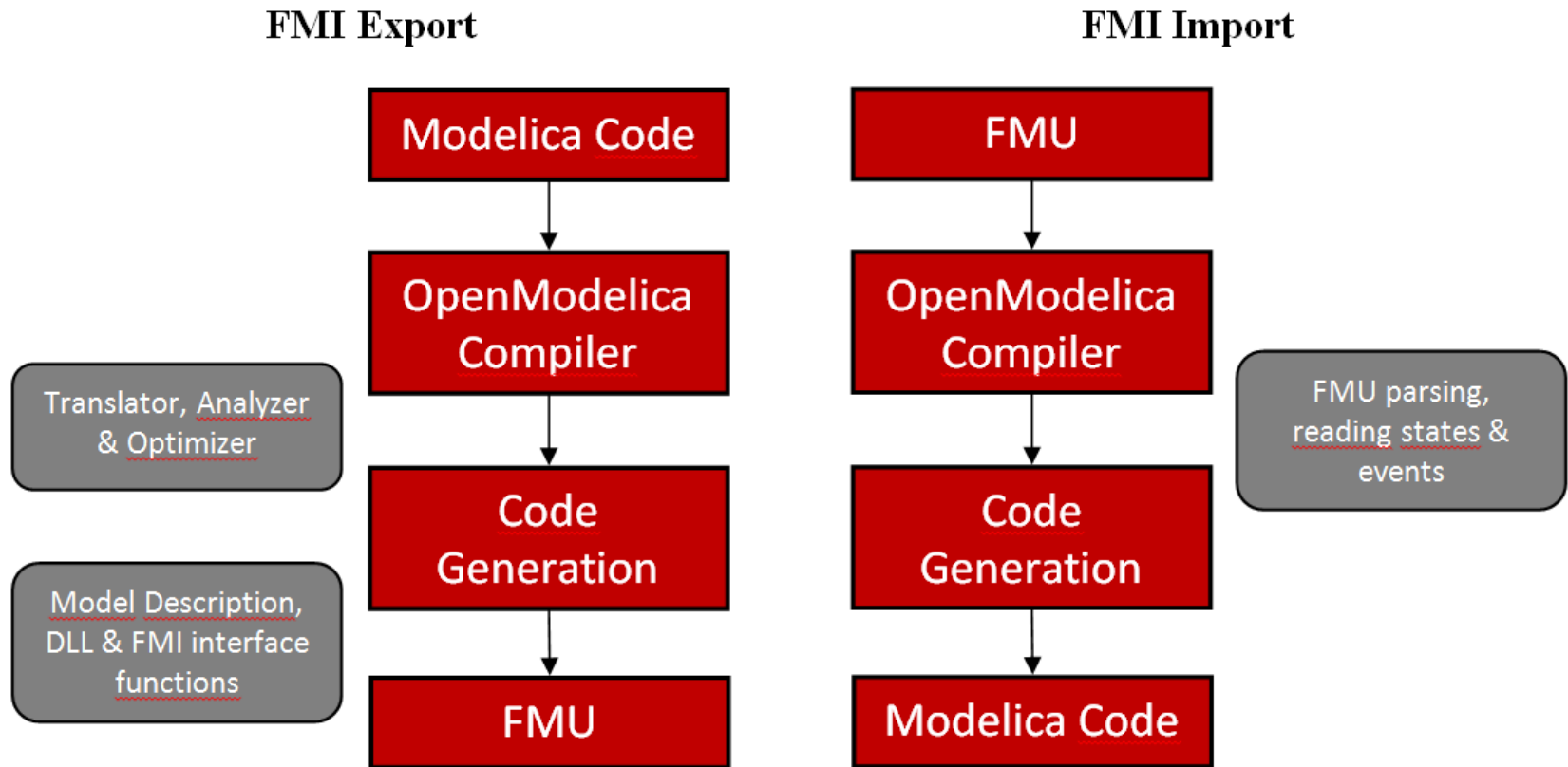


## FMI Simulation Tool Comparison

	OMSimulator	DACCOSIM	Simulink	PyFMI
Commercial	No	No	Yes	No
Open-source	OSMC-PL, GPL	AGPL2	No	LGPL
Lookup Table	Yes	Yes	Yes	No
Alg. Loops	Yes	Yes	No	Yes
Scripting	Python, Lua	proprietary	proprietary	Python
GUI	Yes	Yes	Yes	No
SSP	Yes	No	No	No
platform	Linux/Win/macOS	Linux/Win	Linux/Win/macOS	Linux/Win/macOS

	Dymola	PySimulator	FMI Go!	FMI Composer
Commercial	Yes	No	No	Yes
Open-source	No	BSD	MIT	No
Lookup Table	Yes	Yes	Yes	Yes
Alg. Loops	Yes	Yes	Yes	Yes
Scripting	proprietary	Python	Go	No
GUI	Yes	Yes	No	Yes
SSP	No	No	Yes	Yes
platform	Linux/Win	Linux/Win	Linux/Win/macOS	Linux/Win/macOS

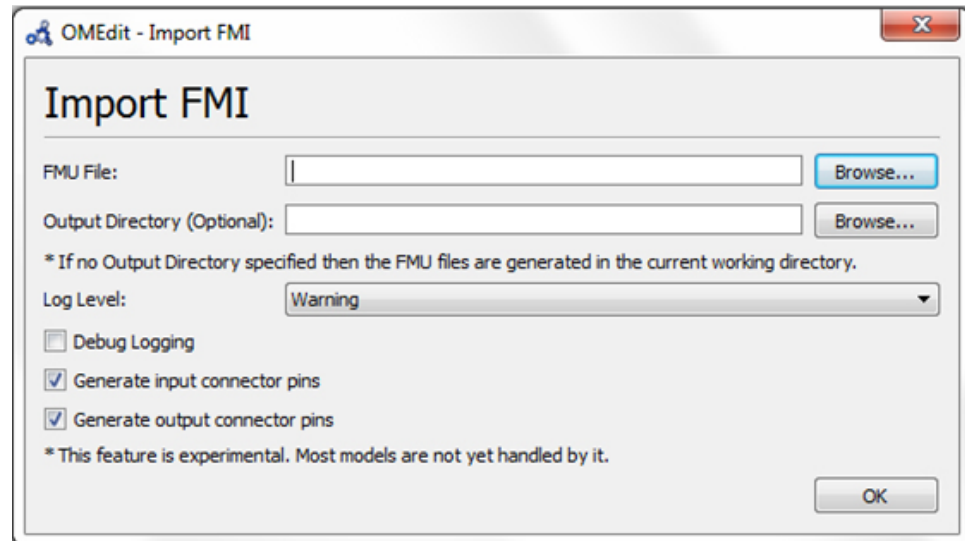
# OpenModelica Functional Mockup Interface (FMI)





# FMI in OpenModelica

- Model Exchange implemented (FMI 2.0)
- FMI 2.0 Co-simulation implemented
- The FMI interface is accessible via the **OpenModelica scripting environment**, the **OpenModelica Connection Editor** and the **OMSimulator** tool in OpenModelica



# 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)	<b>68 kB</b> flash memory <b>25 kB</b> variables (RAM)	<b>4090 B</b> flash memory <b>151 B</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.

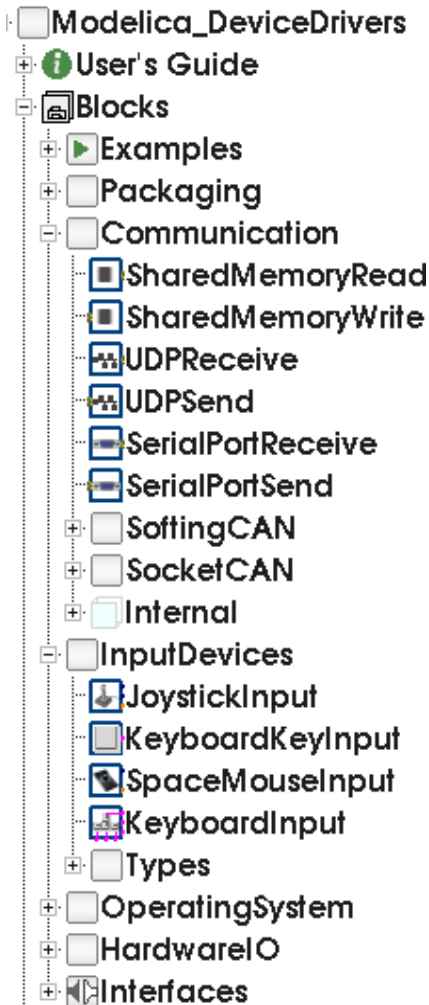
The ATmega16 we target has **1 kB SRAM available** (stack, heap, and global variable

# The Simple Code Generator

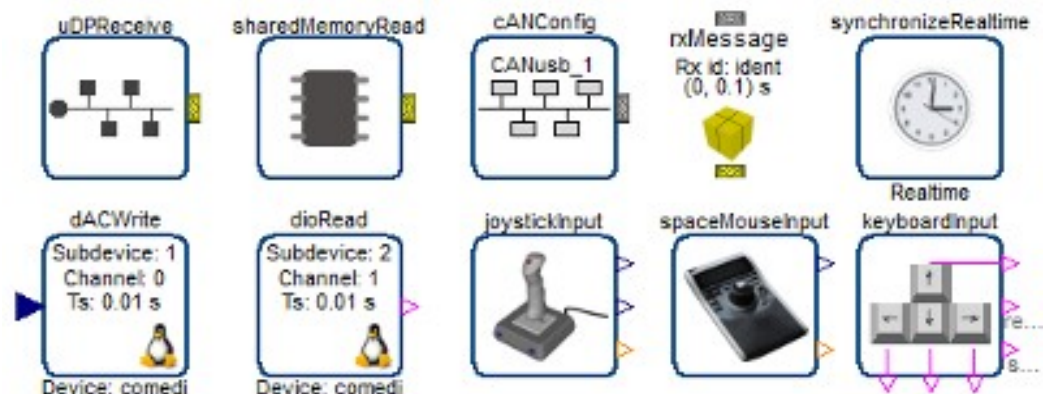
Supports only a limited Modelica subset

- 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



[https://github.com/modelica/Modelica\\_DeviceDrivers/](https://github.com/modelica/Modelica_DeviceDrivers/)

# Modelica connected to external hardware

- IMU (Inertial Measurement Unit)
- Interfaced with a CAN-bus (Controller Area Network bus) - uses Modelica\_DeviceDrivers Library
- Visualized in OMEdit

Courtesy of Volker Waurich - TU Dresden





# 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)

- Used 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, see next page!**

# Example – Code Generation to SHBS

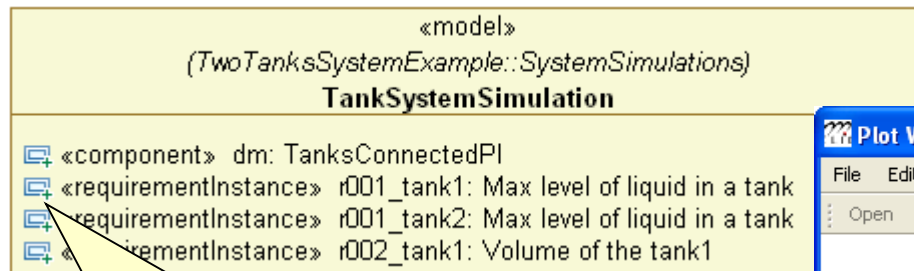


# OpenModelica – ModelicaML UML Profile

## 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

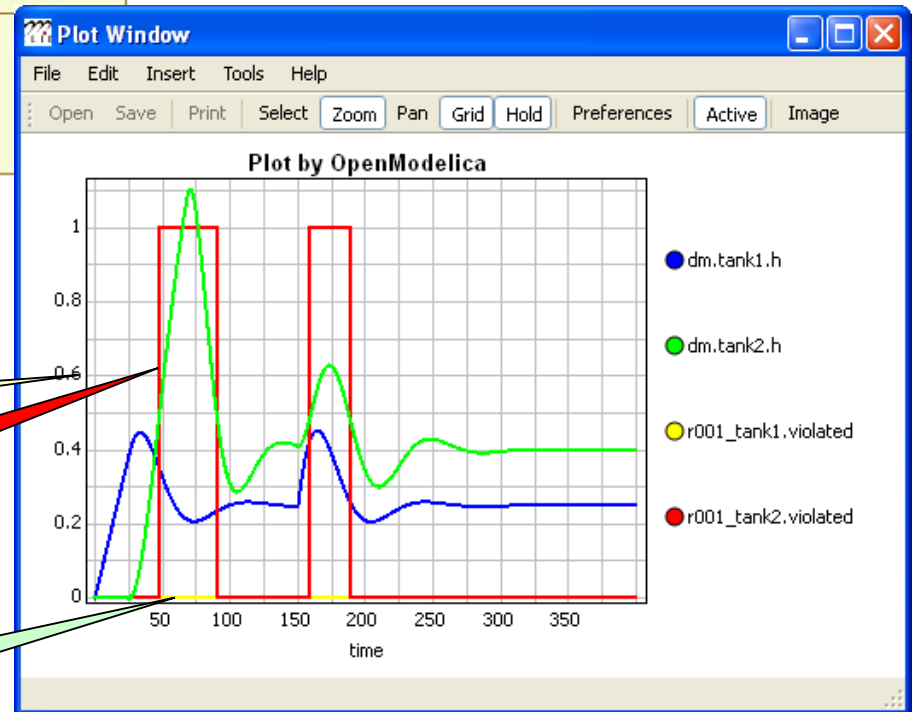


**Req. 001 is instantiated 2 times  
(there are 2 tanks in the system)**

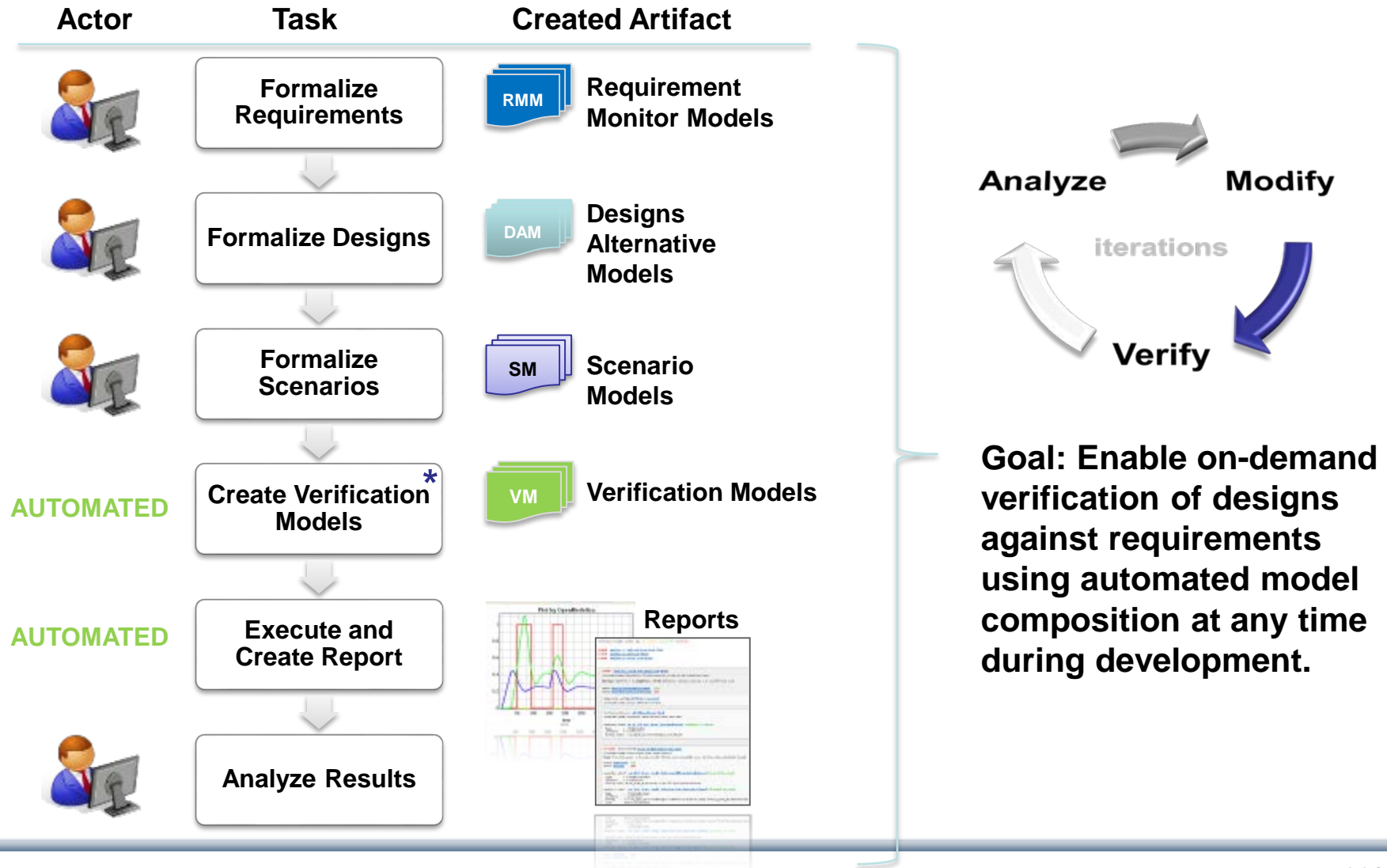
**tank-height is 0.6m**

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

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



# vVDR Method – virtual Verification of Designs vs Requirements





# Need for Debugging Tools

## Map Low vs High Abstraction Level

- A **major part** of the total **cost** of software projects is due to testing and debugging
- US-Study 2002:  
Software errors cost the US economy **annually~ 60 Billion \$**
- **Problem: Large Gap in Abstraction Level**  
from **Equations** to **Executable Code**
- Example error message (hard to understand)  
Error solving nonlinear system 132  
time = 0.002  
residual[0] = 0.288956  
x[0] = 1.105149  
residual[1] = 17.000400  
x[1] = 1.248448  
...

# OpenModelica MDT Algorithmic Code Debugger

The screenshot displays the Eclipse IDE interface for the OpenModelica MDT Algorithmic Code Debugger. The main window is titled "Debug - HelloWorld/SimulationModel.mo - Eclipse SDK". The interface is divided into several panes:

- List of Stack Frames:** Located in the top-left pane, it shows the current execution stack. The top frame is "Simulation Model [Modelica Development Tooling (MDT) GDB]" with a sub-frame "MDT". Below this, the "Main Thread (stepping)" is shown, with the current function being "getValueMultipliedByTwo at simulationmodel.mo:13". The file path "C:\Users\adeas31\workspaceMDT\HelloWorld\SimulationModel.exe" is also visible.
- Variables View:** Located in the top-right pane, it displays the current state of variables. The table below shows the variables:

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

- Code Editor:** The central pane shows the source code of "SimulationModel.mo". The code defines a model "SimulationModel" and a function "getValueMultipliedByTwo". The function is currently selected, showing its implementation: 

```
function getValueMultipliedByTwo
  input Real inValue;
  output Real outValue;
  algorithm
    outValue := inValue * 2;
  end getValueMultipliedByTwo;
```
- Outline View:** Located in the bottom-right pane, it shows the project structure. The "getValueMultipliedByTwo" function is highlighted, showing its inputs and outputs: "inValue (Real - IN)" and "outValue (Real - OUT)". The "SimulationModel" package is also visible, containing variables "x" and "y".
- Output View:** Located in the bottom-left pane, it shows the console output. The text "Simulation Model [Modelica Development Tooling (MDT) GDB] C:\Users\adeas31\workspaceMDT\HelloWorld\SimulationModel.exe" is visible.

# 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 GDB [Modelica Development Tooling (MDT) GDB]

MDT

Main Thread (stepping)

- オープンモデリッカー・ロック 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'  
  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';
```

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. The top bar shows the file path: OpenModelica/OMEdit/Modelica.Mechanics.MultiBody.Examples.Elementary.DoublePendulum\_info.xml. The interface is divided into three main panes:

- Variables View:** Located on the left, it shows a tree structure of variables. The 'Variables' section lists 'boxBody1', 'body', 'frame\_a', and 'R'. The 'Defined In Equations' table shows the index, type, and equation for each variable. The 'Used In Equations' table shows the index, type, and equation for each variable. The 'Variable Operations' section shows the operations performed on the variables.
- Equations View:** Located at the bottom left, it shows a list of equations. The 'Equations Browser' table lists the index, type, and equation for each equation. The 'Defines' table shows the variable defined by each equation. The 'Depends' table shows the variables that each equation depends on. The 'Equation Operations' section shows the operations performed on the equations.
- Source View:** Located on the right, it shows the source code of the model. The 'Source Browser' table lists the index, type, and equation for each line of code. The 'Source Code' section shows the actual code, including comments and mathematical expressions.

Showing equation transformations of a model:

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

(1) substitution:  
 $y + \text{der}(x * (\text{time} * z))$   
 $\Rightarrow$   
 $y + \text{der}(x * (\text{time} * 1.0))$

(2) simplify:  
 $y + \text{der}(x * (\text{time} * 1.0))$   
 $\Rightarrow$   
 $y + \text{der}(x * \text{time})$

(3) expand derivative (symbolic diff):  
 $y + \text{der}(x * \text{time})$   
 $\Rightarrow$   
 $y + (x + \text{der}(x) * \text{time})$

(4) solve:  
 $0.0 = y + (x + \text{der}(x) * \text{time})$   
 $\Rightarrow$   
 $\text{der}(x) = ((-y) - x) / \text{time}$   
 $\text{time} < 0$

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 panels:

- Variables Browser:** Lists variables with their comments and locations. For example, `phi` is located at `/usr/lib/omlibrary/Modelica 3.2.1/Mechanics/MultiBody/Jo`.
- Defined In Equations:** A table showing the index, type, and equation for each variable. For instance, `phi` is defined in equation 587 as an initial value and in equation 5016 as a regular value.
- Used In Equations:** A table showing the index, type, and equation for each variable. For example, `phi` is used in equations 386 through 391.
- Equations Browser:** Lists equations with their types and equations. For example, `der(cylinder3.B2.R_rel.T[3,3])` is defined in equation 386.
- Source Browser:** Displays the source code for the selected equation. The code includes assertions for frame connectivity, angle calculations, and frame transformations.

The **Equations Browser** table is as follows:

Inc	Type	Equation
...	regular	(assignment) cylind...ylinder3.Cylinder.s
...	regular	(assignment) cylind...linder3.gasForce.L
...	regular	(assignment) cylind...linder3.gasForce.x
...	regular	(assignment) cylind...linder3.gasForce.V
...	regular	(assignment) cylind...linder3.gasForce.L
...	regular	(assignment) cylind...linder3.gasForce.L
...	regular	(assignment) cylind...k2.frame_b.R.T[2,3]
...	regular	(linear, r_rel_a = Fra...r_0 - frame_a.r_0);
...	regular	(linear, frame_b.r_0 = ... * (s_offset + s));
...	regular	(assignment) cylind...linder3.gasForce.x
...	regular	(assignment) cylind...linder3.gasForce.p
...	regular	(assignment) cylind...r3.gasForce.d ^ 2.0
...	regular	(assignment) cylind...linder3.gasForce.k
...	regular	(assignment) cylind...ody.w_a[1] - load.w
...	regular	(assignment) der(c...r3.Rod.body.w_a[1]

The **Equation Operations** panel shows the following operations:

- solved: `der(cylinder3.B2.R_rel.T[3,3]) = (-sin(cylinder3.B2.phi)) * cylinder3.Rod.body.w_a[1]`
- substitute: `(-sin(cylinder3.B2.phi)) * cylinder3.B2.w => (-sin(cylinder3.B2.phi)) * cylinder3.Rod.body.w_a[1]`
- differentiate: `dcos(cylinder3.B2.phi)/dt = (-sin(cylinder3.B2.phi)) * der(cylinder3.B2.phi)`
- differentiate: `dcylinder3.B2.R_rel.T[3,3]/dt = der(cylinder3.B2.R_rel.T[3,3])`
- scalarize(9): `cylinder3.B2.R_rel.T = {{1.0, 0.0, 0.0}, [-0.0, c...B2.phi]}} => cylinder3.B2.R_rel.T[3,3] = cos(cylinder3.B2.phi)`
- simplify: `cylinder3.B2.R_rel.T = {{1.0 * 1.0 + (1.0 - 1.0 * 1.0) * B2.phi}, {0.0, -sin(cylinder3.B2.phi), cos(cylinder3.B2.phi)}}`
- substitute: `{{cylinder3.B2.e[1] * cylinder3.B2.e[1] + (1.0 - cy...2.phi), 0.0 * 0.0 + (1.0 - 0.0 * 0.0) * cos(cylinder3.B2.phi)}}`
- inline: `cylinder3.B2.R_rel = Modelica.Mechanics.MultiBody...[2] * cylinder3.B2.w, cylinder3.B2.e[3] * cylinder3.B2.w}}`
- original: `R_rel = Frames.planarRotation(e, phi_offset + phi, w); => flattened:`

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

OMEdit - Transformational Debugger

File Edit View Windows Help

Variables Browser

Variables	Comment	Line	Location
Ssce1		0	
Ssce2		0	
Ssce3		0	
Ssce4		0	
Ssce5		0	
Ssce6		0	
Evap	Structure...rations	4481	C:/OpenMod...package.mo
FGflow	Output ...artTime	421	C:/OpenMod...Sources.mo
FGinV	Volume ...port_b	333	C:/OpenMod...terfaces.mo
FGinVu	Constan...t value	239	C:/OpenMod...Sources.mo
FGoutV	= true ... volume	243	C:/OpenMod...terfaces.mo
FGoutVu	Constan...t value	239	C:/OpenMod...Sources.mo
FGoutletVu	Constan...t value	239	C:/OpenMod...Sources.mo
FGsink	Boundary density	13	C:/OpenMod...Sources.mo
FGsource	Get the...nector	446	C:/OpenMod...Sources.mo
FGtemp	Duration...a Step	418	C:/OpenMod...Sources.mo
FWinletV	Av (metr...efficient	436	C:/OpenMod...d/Valves.mo
FWinletVu	Constan...t value	239	C:/OpenMod...Sources.mo
FWinletVt	Tempera... = true	346	C:/OpenMod...terfaces.mo
FWpump	Needed...nector	333	C:/OpenMod...Sources.mo
FWpump_h	Height of ramps	417	C:/OpenMod...Sources.mo
FWpump_p	Offset o...t signal	420	C:/OpenMod...Sources.mo
SteamSink	Densit...medium	4484	C:/OpenMod...package.mo
system	Constan...eration	14	C:/OpenMod.../System.mo

Defined In Equations

Index	Type	Equation	Execution	Max
1461	regular	Evap.Hex.LnC.Usl[1] := Evap.Hex.LnC.ms[1] * Evap.i		
1462	regular	Evap.Hex.LnC.flowModel.ms[1] := Modelica.Med		
1463	regular	Evap.Hex.LnC.mediums[1].sat.Tsat := Modelica.Me		
1464	regular	Evap.Hex.LnC.mediums[2].p_bar := 1e-005 * Evap.i		
1465	regular	Evap.Hex.LnC.stateFM[2].phase := if Evap.Hex.Ln		
1466	regular	Evap.Hex.LnC.heatTransfer.Ts[2] := Modelica.Medi		
1467	regular	Evap.Hex.LnC.mediums[2].T_degC := -273.15 + Ev		
1468	regular	Evap.Hex.LnC.stateFM[2].d := Modelica.Media.Wi		
1469	regular	Evap.Hex.LnC.ms[2] := Evap.Hex.LnC.fluidVolume		
1470	regular	Evap.Hex.LnC.mediums[2].u := Evap.Hex.LnC.med		
1471	regular	Evap.Hex.LnC.Usl[2] := Evap.Hex.LnC.ms[2] * Evap.i		
1472	regular	Evap.Hex.LnC.flowModel.ms[2] := Modelica.Med		
1473	regular	Evap.Hex.LnC.mediums[2].sat.Tsat := Modelica.Me		
1474	regular	Evap.Hex.LnC.flowModel.Fs.pl[1] := 0.5 * (Evap.H		
1475	regular	Evap.Hex.LnC.flowModel.dps.fgl[1] := -2.0 * DIVISI		
1476	regular	Evap.Hex.LnC.m_flow[2] := Modelica.Fluid.Pipes...		
1477	regular	Evap.Hex.LnC.flowModel.ms_act[1] := if noEven...		
1478	regular	Evap.Hex.LnC.flowModel.Is[1] := Evap.Hex.LnC.m...		
1479	regular	Evap.Hex.LnC.H_flow[2] := semiLinear(Evap.Hex...		
1480	regular	Evap.Hex.LnC.flowModel.hos_act[1] := if noEvent...		
1481	regular	Evap.Hex.LnC.state_a.T := Modelica.Media.Water...		
1482	regular	Evap.Hex.LnC.state_a.d := Modelica.Media.Water...		
1483	regular	Evap.Hex.LnC.state_b.T := Modelica.Media.Water...		
1484	regular	Evap.Hex.LnC.state_b.d := Modelica.Media.Water...		
1485	regular	Evap.Hex.LnC.Area.y := Evap.Hex.LnC2.area.offse		
1486	regular	Evap.Hex.LnC2.division2.u2 := Evap.Hex.LnC2.kinn		
1487	regular	Evap.Hex.LnC2.division2.y := Evap.Hex.LnC2.division2.u1 / Evap.Hex.LnC2.division2.u2		
1488	regular	Evap.Hex.LnC2.division2.y := Evap.Hex.LnC2.division2.u1 / Evap.Hex.LnC2.division2.u2		

Equations Browser

Index	Type	Equation
1461	regular	Evap.Hex.LnC.Usl[1] := Evap.Hex.LnC.ms[1] * Evap.i
1462	regular	Evap.Hex.LnC.flowModel.ms[1] := Modelica.Med
1463	regular	Evap.Hex.LnC.mediums[1].sat.Tsat := Modelica.Me
1464	regular	Evap.Hex.LnC.mediums[2].p_bar := 1e-005 * Evap.i
1465	regular	Evap.Hex.LnC.stateFM[2].phase := if Evap.Hex.Ln
1466	regular	Evap.Hex.LnC.heatTransfer.Ts[2] := Modelica.Medi
1467	regular	Evap.Hex.LnC.mediums[2].T_degC := -273.15 + Ev
1468	regular	Evap.Hex.LnC.stateFM[2].d := Modelica.Media.Wi
1469	regular	Evap.Hex.LnC.ms[2] := Evap.Hex.LnC.fluidVolume
1470	regular	Evap.Hex.LnC.mediums[2].u := Evap.Hex.LnC.med
1471	regular	Evap.Hex.LnC.Usl[2] := Evap.Hex.LnC.ms[2] * Evap.i
1472	regular	Evap.Hex.LnC.flowModel.ms[2] := Modelica.Med
1473	regular	Evap.Hex.LnC.mediums[2].sat.Tsat := Modelica.Me
1474	regular	Evap.Hex.LnC.flowModel.Fs.pl[1] := 0.5 * (Evap.H
1475	regular	Evap.Hex.LnC.flowModel.dps.fgl[1] := -2.0 * DIVISI
1476	regular	Evap.Hex.LnC.m_flow[2] := Modelica.Fluid.Pipes...
1477	regular	Evap.Hex.LnC.flowModel.ms_act[1] := if noEven...
1478	regular	Evap.Hex.LnC.flowModel.Is[1] := Evap.Hex.LnC.m...
1479	regular	Evap.Hex.LnC.H_flow[2] := semiLinear(Evap.Hex...
1480	regular	Evap.Hex.LnC.flowModel.hos_act[1] := if noEvent...
1481	regular	Evap.Hex.LnC.state_a.T := Modelica.Media.Water...
1482	regular	Evap.Hex.LnC.state_a.d := Modelica.Media.Water...
1483	regular	Evap.Hex.LnC.state_b.T := Modelica.Media.Water...
1484	regular	Evap.Hex.LnC.state_b.d := Modelica.Media.Water...
1485	regular	Evap.Hex.LnC.Area.y := Evap.Hex.LnC2.area.offse
1486	regular	Evap.Hex.LnC2.division2.u2 := Evap.Hex.LnC2.kinn
1487	regular	Evap.Hex.LnC2.division2.y := Evap.Hex.LnC2.division2.u1 / Evap.Hex.LnC2.division2.u2
1488	regular	Evap.Hex.LnC2.division2.y := Evap.Hex.LnC2.division2.u1 / Evap.Hex.LnC2.division2.u2

Equation Operations

Operations
solved: Evap.Hex.LnC2.division2.y := Evap.Hex.LnC2.division2.u1 / Evap.Hex.LnC2.division2.u2
substitution: Evap.Hex.LnC2.division2.y := Evap.Hex.LnC2.division2.u1 / Evap.Hex.LnC2.division2.u2
substitution: Evap.Hex.LnC2.division2.y := Evap.Hex.LnC2.division2.u1 / Evap.Hex.LnC2.division2.u2
substitution: Evap.Hex.LnC2.division2.y := Evap.Hex.LnC2.division2.u1 / Evap.Hex.LnC2.division2.u2

Source Browser

```
extent={{-100,-100},{100,100}}, graphics={Rectangle(
  extent={{-100,-100},{100,100}},
  lineColor={0,0,255},
  fillColor={255,255,255},
  fillPattern=FillPattern.Solid),Line(points={{-100,6
0},{-40,60},{-30,
40}}, color={0,0,255}),Line(points={{-100,-60},
0},{-30,-40}},
color={0,0,255}),Line(points={{50,0},{100,0}},
0,255}),
Line(points={{-30,0},{30,0}},
0,0)),Line(points={{-15,
25.99},{15,-25.99}},
0,0)),Line(points={{-15,-25.99},{15,
25.99}}, color={0,0,0}),Ellipse(extent={{-50,50},
},
lineColor={0,0,255}});
product;

block Division "Output first input divided by second input"
extends Interfaces.SI2SO;

equation
y = u1/u2;
annotation (
Documentation(info="<html>
<p>
This block computes the output <b>y</b> (element-wise)
by <i>dividing</i> the corresponding elements of
the two inputs <b>u1</b> and <b>u2</b>:
</p>
<pre>
y = u1 / u2;
</pre>
</html>"),
Icon(coordinateSystem(
preserveAspectRatio=true,
extent={{-100,-100},{100,100}},
initialScale=0.1), graphics={
Line(points={{50,0},{100,0}}, color={0,0,127}),
Line(points={{-30,0},{30,0}},
Ellipse(fillPattern=FillPattern.Solid, extent={{-5,20},
{5,30}}),
Ellipse(fillPattern=FillPattern.Solid,
extent={{-5,-30},{5,-20}}),
Ellipse(lineColor={0,0,127}, extent={{-50,-50},
{50,50}}),
Text(
lineColor={0,0,255},
extent={{-150,110},{150,150}},
textString="<b>y</b> = <b>u1</b> / <b>u2</b>")
}
```

Pointing out the buggy equation  
 $y = u1/u2;$   
that gives division by zero



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

OMEdit - Transformational Debugger

/tmp/OpenModelica\_marsj/OMEdit/Debugging.Chattering.ChatteringEvents1\_info.xml

Variables

Variables Browser

Find Variables

☐ Case Sensitive Regular Expression

Expand All Collapse All

Variables	Comment	Line	Location
x		7	/hom...g.
y		8	/hom...g.
z		9	/hom...g.

Defined In Equations

Inc	Type	Equation
2	initial	(assignment) y = 2.0 * z
5	regular	(assignment) y = 2.0 * z

Used In Equations

Inc	Type	Equation
3	initial	(assignment) y = 2.0 * z
6	regular	(assignment) y = 2.0 * z

Source Browser

/home/marsj/trunk/testsuite/openmodelica

```
1 within ;
2 package Debugging "Test
3 cases for debugging of
4 declarative models"
5
6 package Chattering "Models
7 with chattering behaviour"
8 model ChatteringEvents1
9 "Exhibits chattering
10 after t = 0.5, with
11 generated events"
12 Real x(start=1,
13 fixed=true);
14 Real y;
15 Real z;
16 equation
17 z = if x > 0 then -1
18 else 1;
19 y = 2*z;
20 der(x) = y;
21 annotation
22 (Documentation(info="<html>
23 <p>After t = 0.5, chattering
24 takes place, due to the
25 feedback in the right
26 hand side of the
27 equation.</p>
28 <p>Chattering can be
29 detected because lots of
30 tightly spaced events are
31 generated. The feedback to
32 the user should allow to
33 identify the equation from
34 which the zero crossing
35 function that generates the
36 events originates.</p>
37 </html>"),
38 experiment(StopTime=1));
39 end ChatteringEvents1;
40
41 model ChatteringEvents2
42 "Exhibits chattering
43 after t = 0.422, with
44 generated events"
```

Equations

Equations Browser

Inc	Type	Equation
1	initial	(assignment) x = 1.0
2	initial	(assignment) y = 2.0 * z
3	initial	(assignment) y = 2.0 * z
4	initial	(assignment) der(x) = y
5	regular	(assignment) y = 2.0 * z
6	regular	(assignment) y = 2.0 * z
7	regular	(assignment) der(x) = y

Defines

Variable
z

Depends

Variable
x

Variable Operations

Operations

Equation Operations

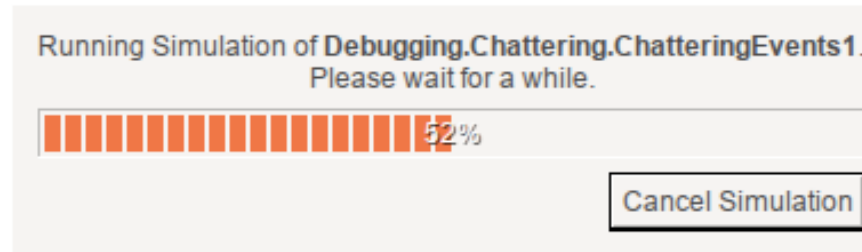
Operations

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;

equation

z = if x > 0 then -1 else 1;  
y = 2 \* z;

# Error Indication – Simulation Slows Down



# Performance Profiling for Faster Simulation

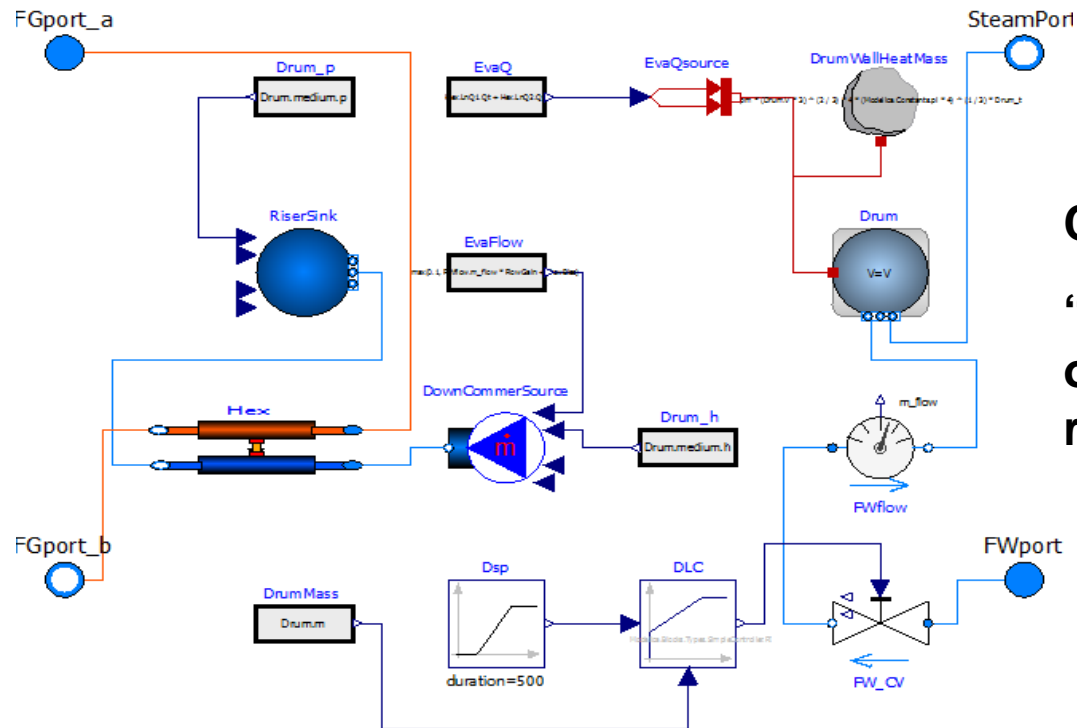
## (Here: Profiling all equations in MSL 3.2.1 DoublePendulum)

- Measuring **performance** of equation blocks to find bottlenecks
  - Useful as input before model simplification for real-time 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

Performance profiling DoublePendulum:

Equations Browser							Defines
Index	Type	Equation	Executi	Max time	Time	Fraction	Variable
+ 876	regular	linear, size 2	4602	0.000501	0.0134	75.7%	damper.a_rel revolute2.frame_b.f[2]
- 836	regular	(assignment) ...evolute2.phi)	1534	2.57e-05	0.000377	2.12%	
- 840	regular	(assignment) ...mper.phi_rel)	1534	1.38e-05	0.000237	1.33%	
- 837	regular	(assignment) ...evolute2.phi)	1534	8.38e-06	0.000235	1.32%	
- 841	regular	(assignment) ...mper.phi_rel)	1534	8.48e-06	0.000192	1.08%	
- 849	regular	(assignment) ...mper.phi_rel)	1534	8.04e-06	0.000146	0.824%	

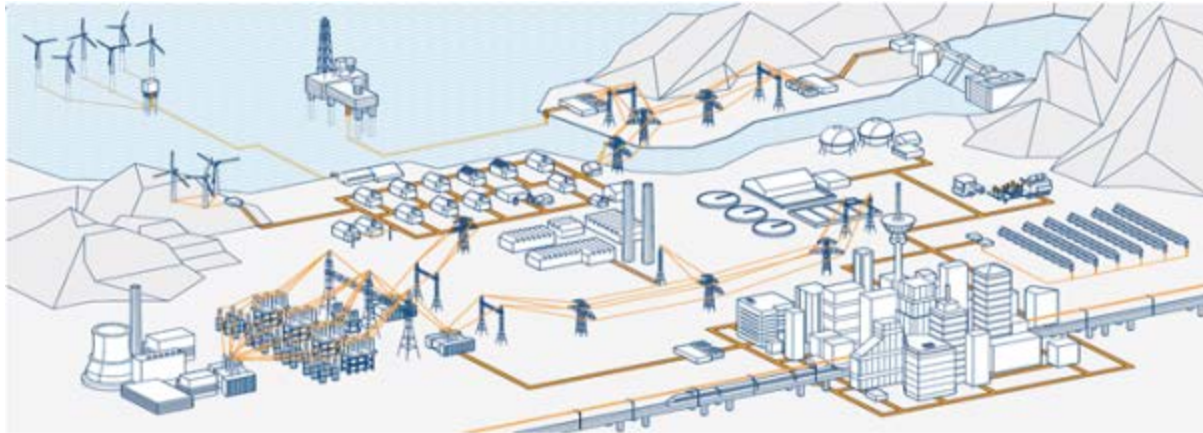
# Performance Profiling of Siemens Drum Boiler Model with Evaporator



**Conclusion from the evaluation:**  
“...the profiler makes the process of performance optimization radically shorter.”

# ABB Industry Use of OpenModelica FMI 2.0 and Debugger

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



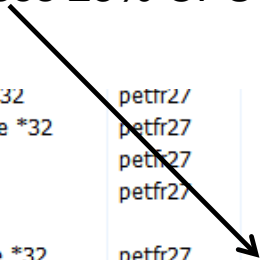
- ABB: *“ABB uses several compatible Modelica tools, including OpenModelica, depending on specific application needs.”*
- ABB: *“OpenModelica provides outstanding debugging features that help to save a lot of time during model development.”*

# 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 ChatteringEvents1;
```

Uses 25% CPU



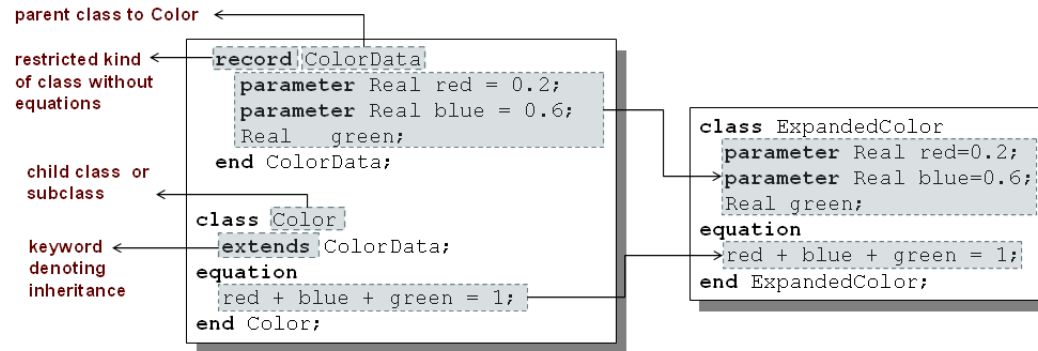
acrotray.exe *32	pettr2/	00	9/6 K	A
AdobeARM.exe *32	petfr27	00	1,136 K	A
Bootcamp.exe	pettr27	00	1,448 K	B
conhost.exe	petfr27	00	1,300 K	C
csrss.exe		00	3,000 K	
DCSHelper.exe *32	petfr27	00	660 K	D
Debugging.Chattering...	petfr27	25	1,436 K	D
dllhost.exe	petfr27	00	2,224 K	C

- 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

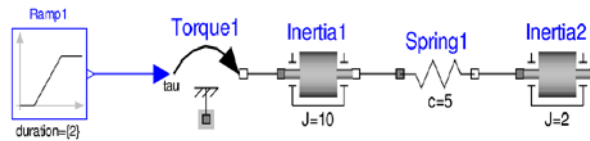
Hybrid  
Modeling

# Acausal Modeling

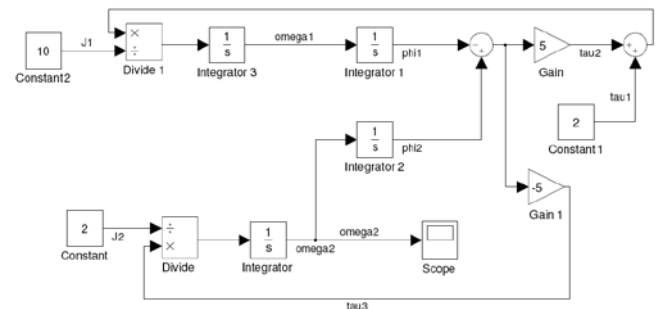
The order of computations is not decided at modeling time

Visual  
Component  
Level

Acausal



Causal



Equation  
Level

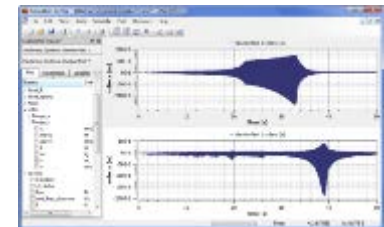
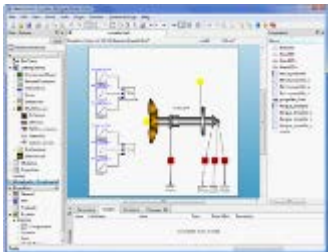
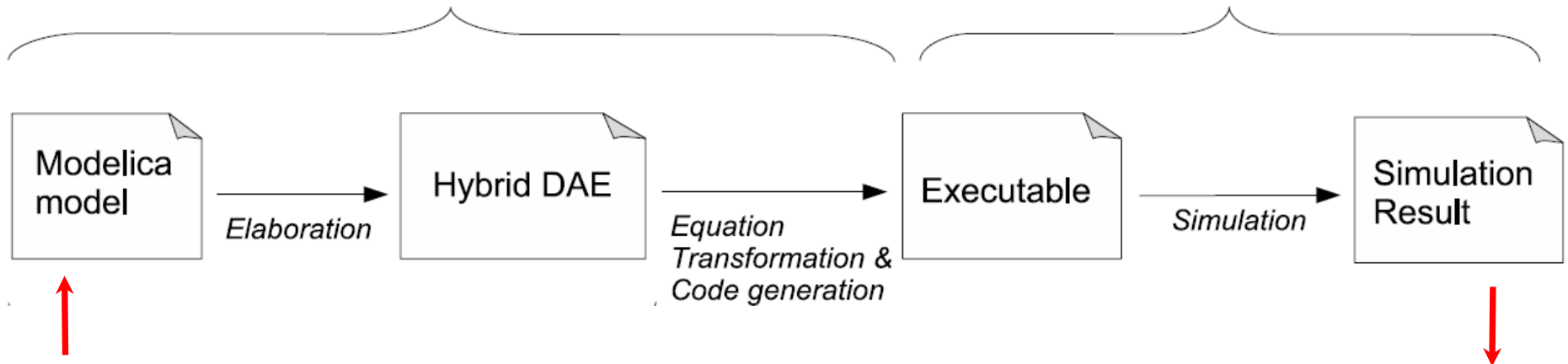
A resistor *equation*:  
 $R \cdot i = v;$

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:  $x(0) = 1$

Name of model

Initial condition

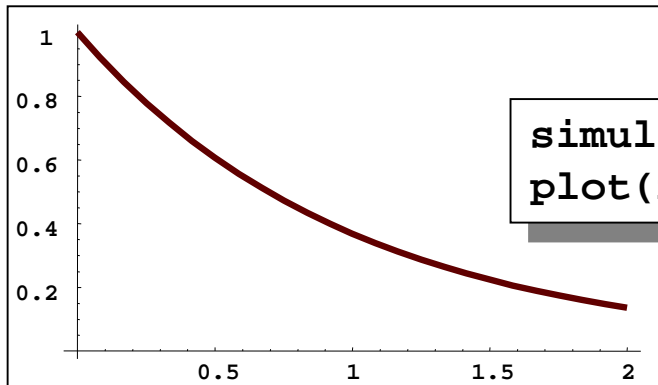
Continuous-time  
variable

Parameter, constant  
during simulation

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

Differential equation

## Simulation in OpenModelica environment



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

# Modelica Variables and Constants

- Built-in primitive data types

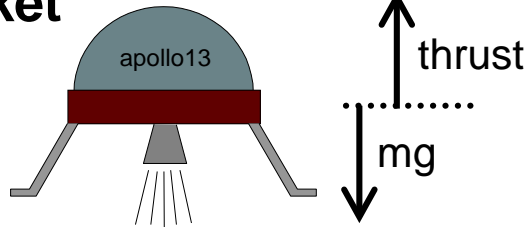
<b>Boolean</b>	<b>true</b> or <b>false</b>
<b>Integer</b>	Integer value, e.g. <b>42</b> or <b>-3</b>
<b>Real</b>	Floating point value, e.g. <b>2.4e-6</b>
<b>String</b>	String, e.g. <b>"Hello world"</b>
<b>Enumeration</b>	Enumeration literal e.g. <b>ShirtSize.Medium</b>

- 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

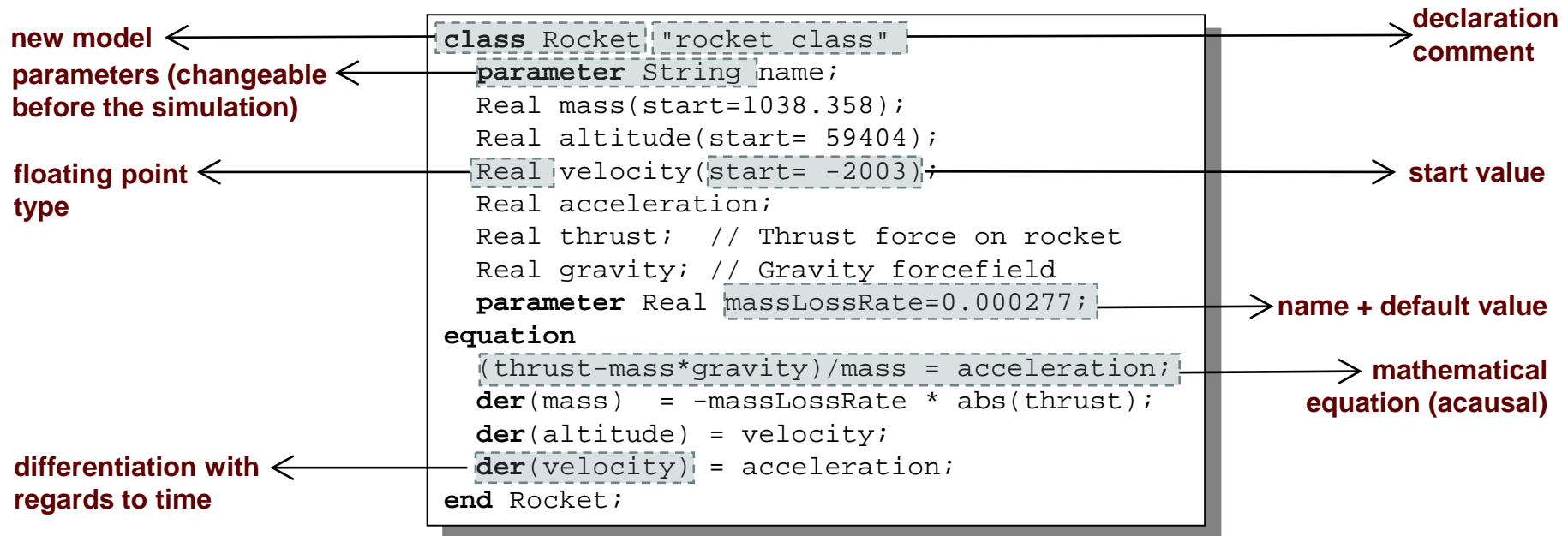


$$acceleration = \frac{thrust - mass \cdot gravity}{mass}$$

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

$$altitude' = velocity$$

$$velocity' = acceleration$$





# 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;
```

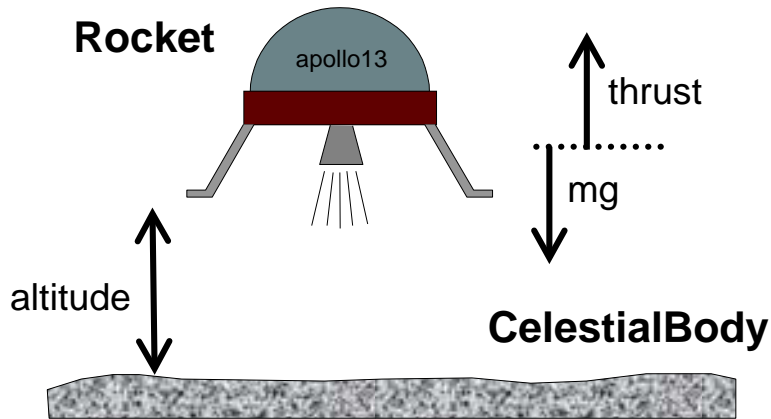


An *instance* of the class can be declared by *prefixing* the type name to a variable name

```
...
CelestialBody moon;
...
```

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

# Moon Landing



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

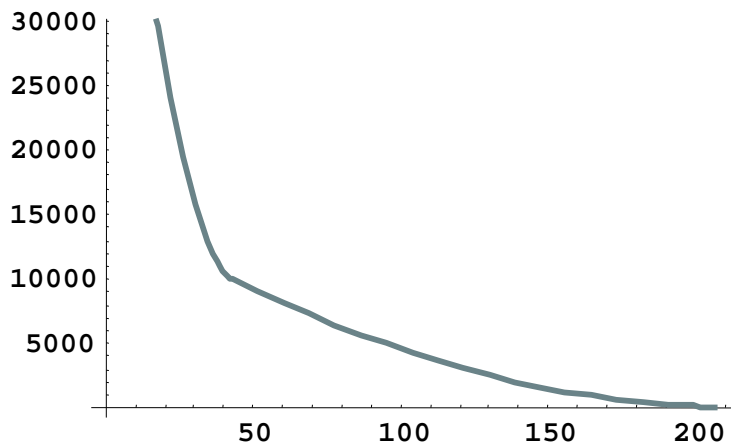
only access  
inside the class

access by dot  
notation outside  
the class

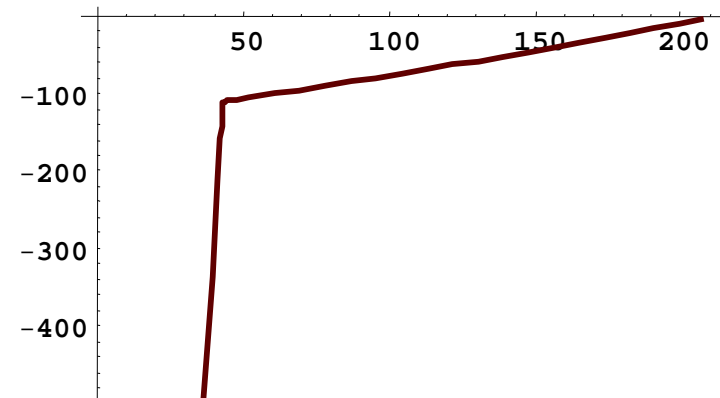
```
class MoonLanding
  parameter Real force1 = 36350;
  parameter Real force2 = 1308;
  protected
    parameter Real thrustEndTime = 210;
    parameter Real thrustDecreaseTime = 43.2;
  public
    Rocket apollo(name="apollo13");
    CelestialBody moon(name="moon",mass=7.382e22,radius=1.738e6);
  equation
    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)
```

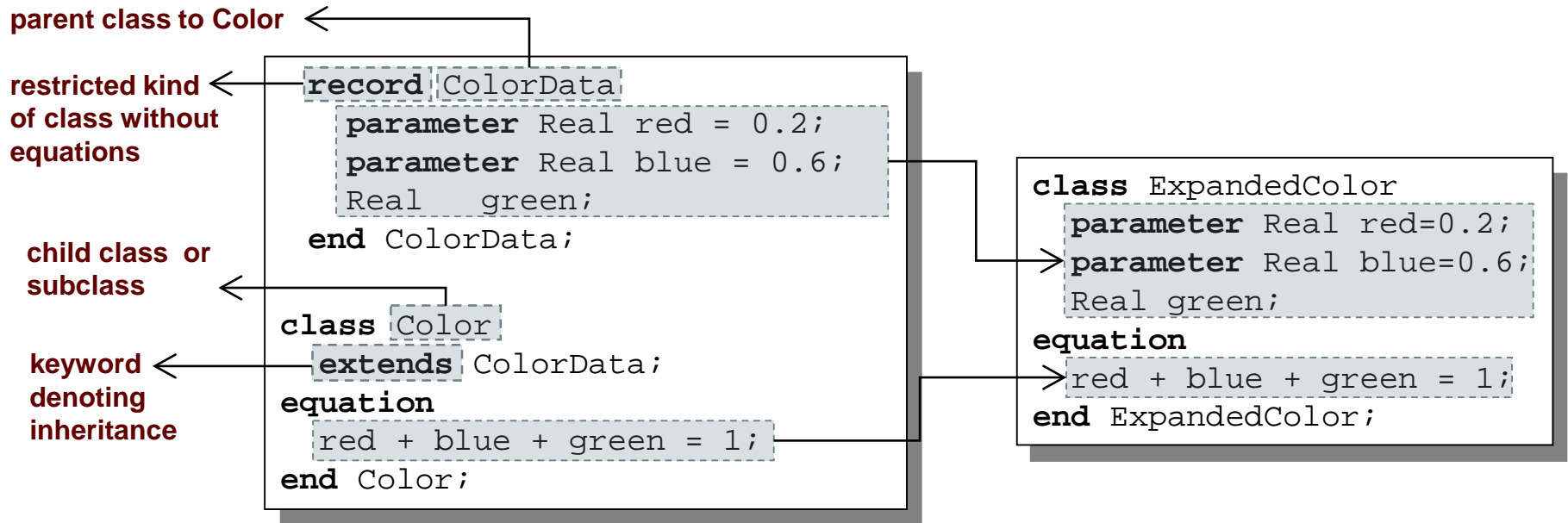
$\{1, 2, 3, 4\}$  becomes  
the value of the  
coefficient vector A, and  
21 becomes the value of  
the formal parameter x.

```
function PolynomialEvaluator  
  input Real A[:]; // array, size defined  
                        // at function call time  
  input Real x := 1.0; // default value 1.0 for x  
  output Real sum;  
protected  
  Real xpower; // local variable xpower  
algorithm  
  sum := 0;  
  xpower := 1;  
  for i in 1:size(A,1) loop  
    sum := sum + A[i]*xpower;  
    xpower := xpower*x;  
  end for;  
end PolynomialEvaluator;
```

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



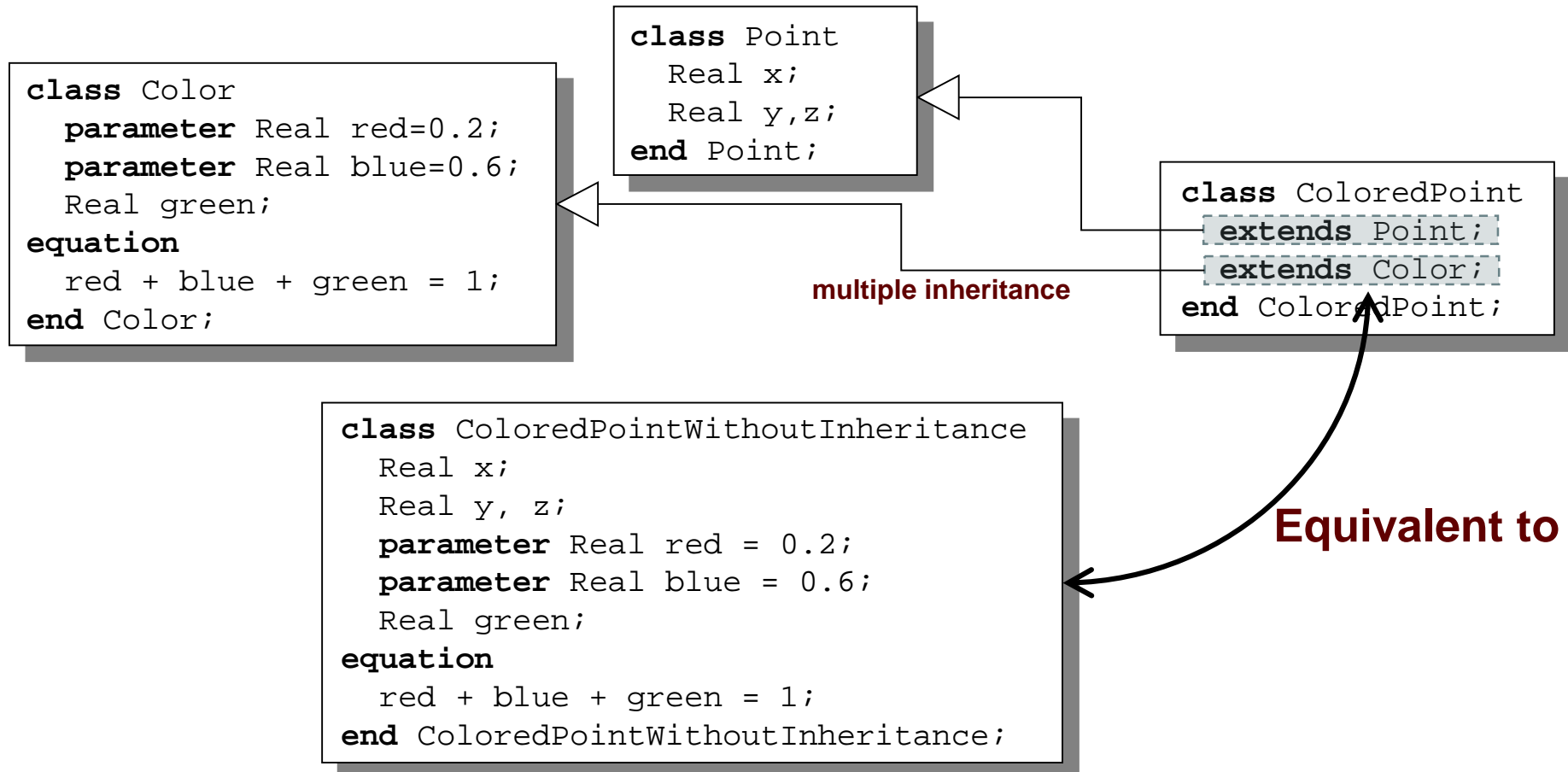
# Inheritance



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

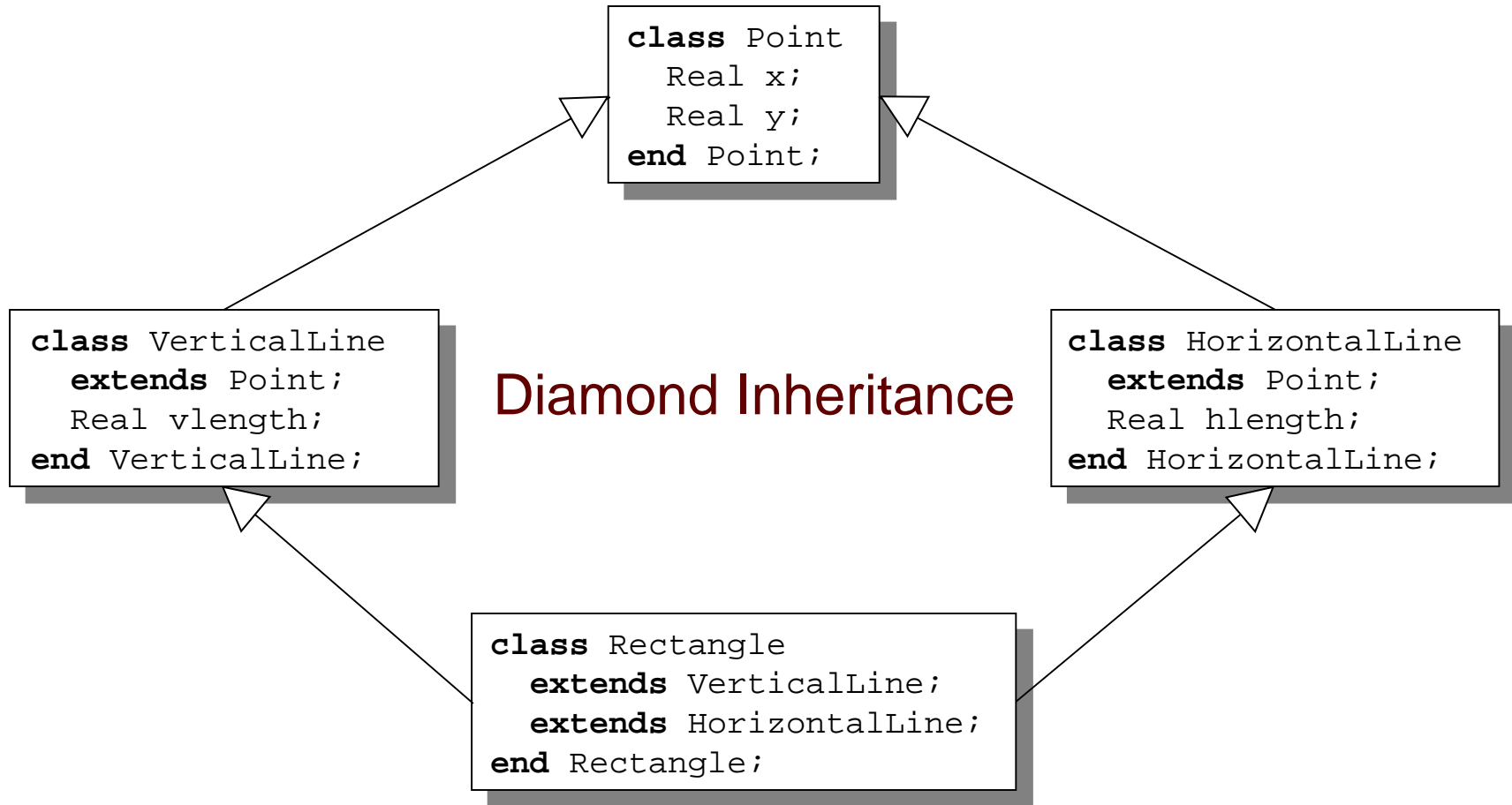
# Multiple Inheritance

Multiple Inheritance is fine – inheriting both geometry and color



# Multiple Inheritance cont'

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



# Simple Class Definition

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

```
class SameColor = Color;
```

Equivalent to:

inheritance ←

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

- Often used for introducing new names of types:

```
type Resistor = Real;
```

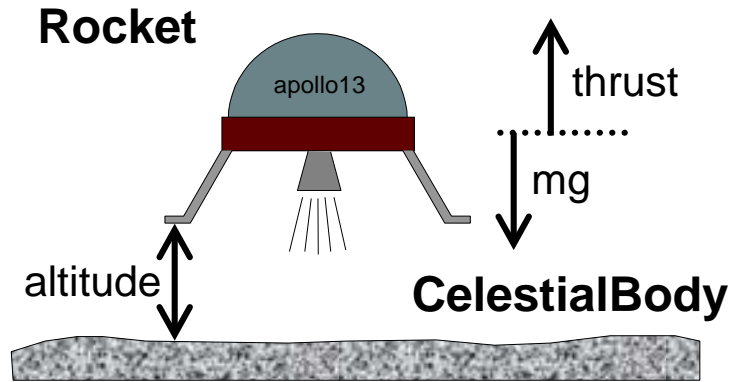
```
connector MyPin = Pin;
```

# 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;

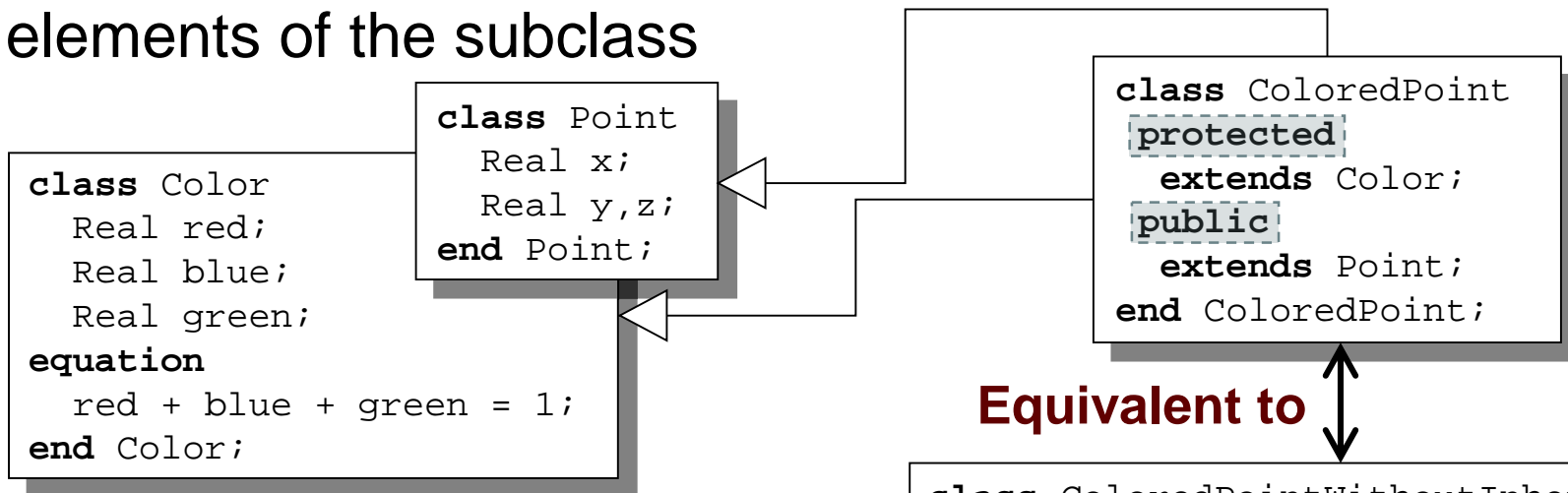
```

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!

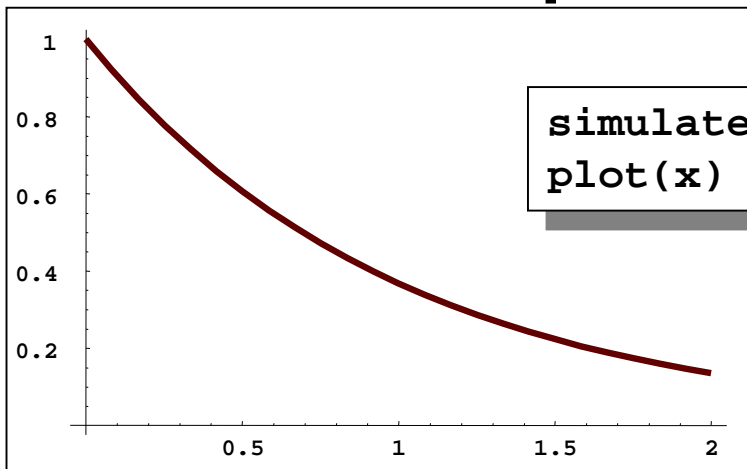
## A Modelica “Hello World” model

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; (*xxxxx s*)  
end HelloWorld;
```

## Simulation in OpenModelica environment

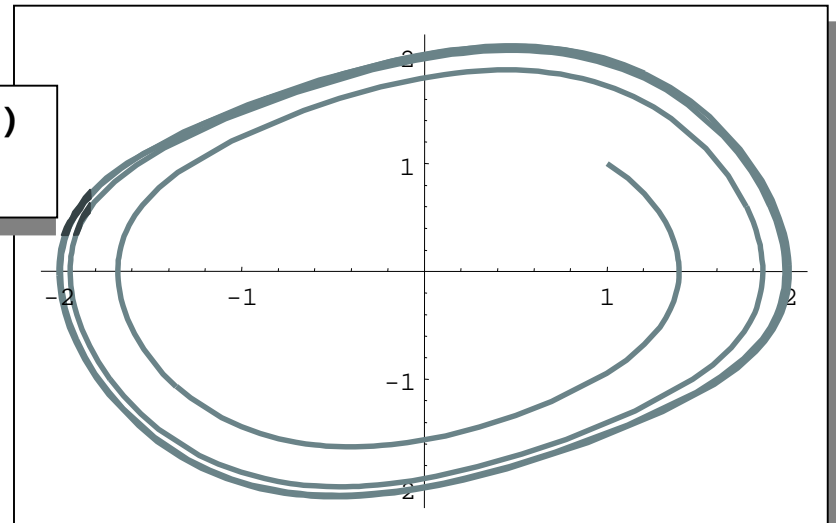


```
simulate(HelloWorld, stopTime = 2)  
plot(x)
```

## (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

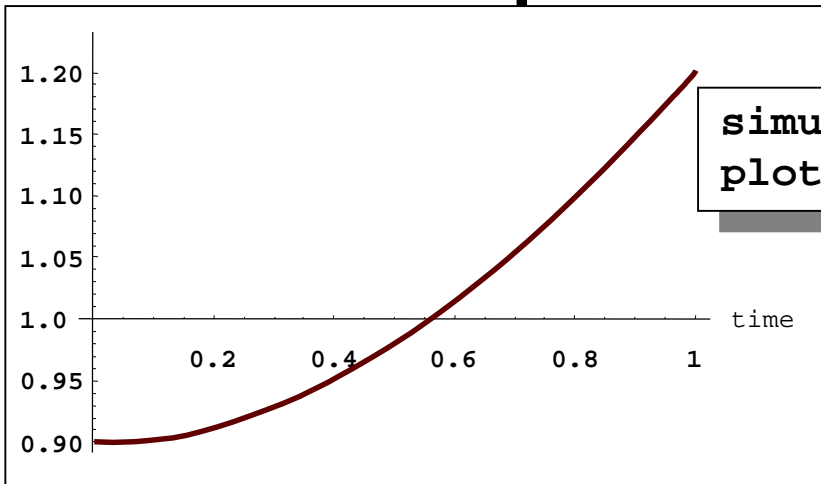
### Include algebraic equation

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

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

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

$$x(0) = 2$$

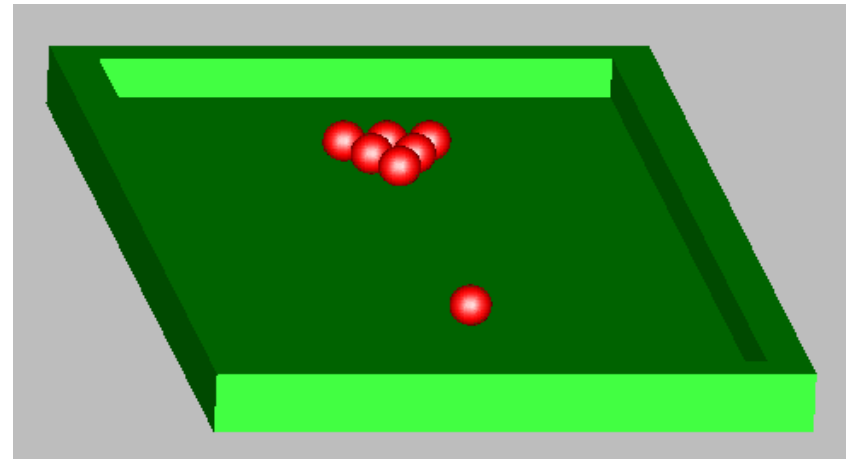
$$y(0) = 3$$

## (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

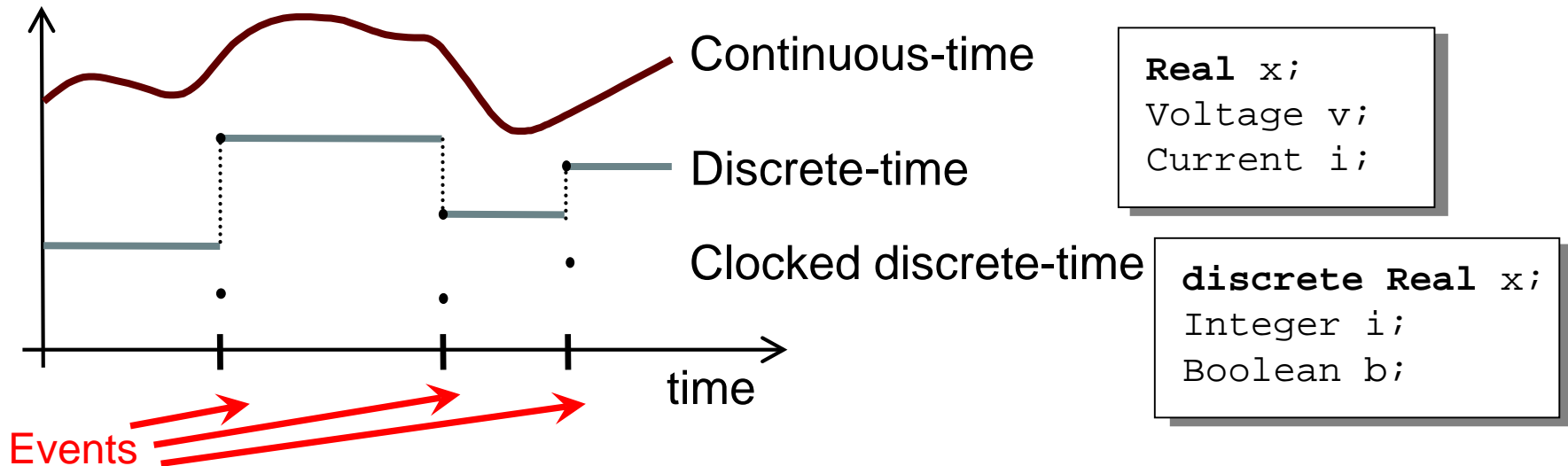
## Discrete Events and Hybrid Systems



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  $v$

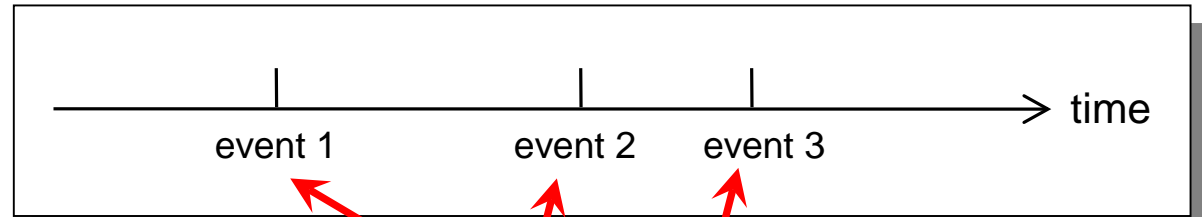
If-expression

# 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

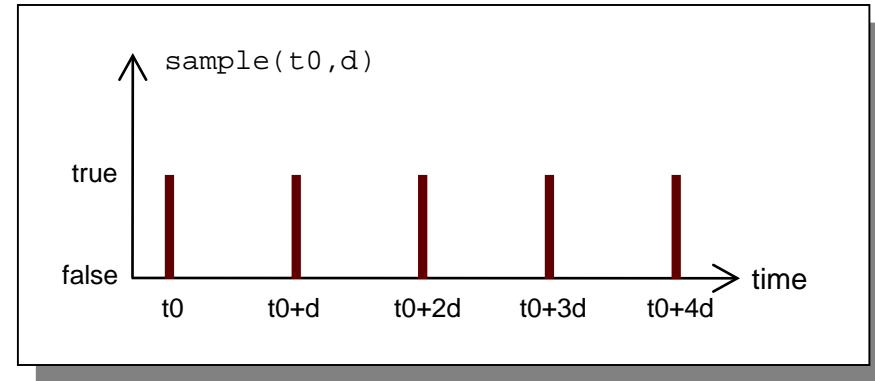
## State event

```
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  $i = 0, 1, \dots$

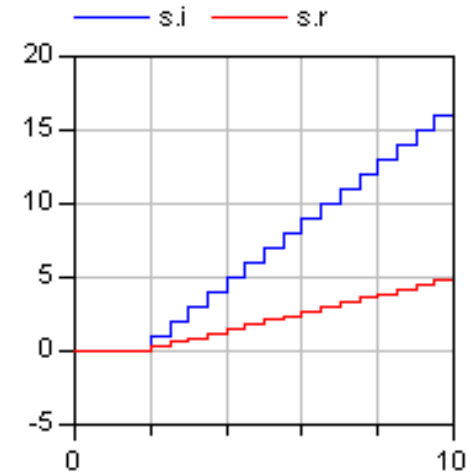


Variables need to be discrete

```
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.



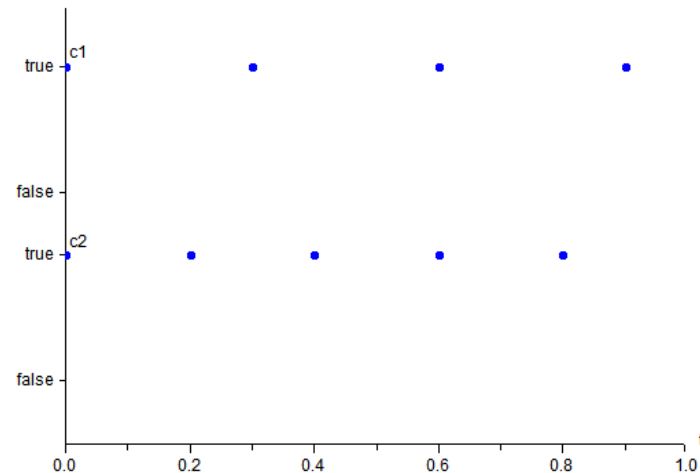


# Generating Clock Tick Events using Clock()

## (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(c=c, solverMethod=solverMethod)** – solver clock

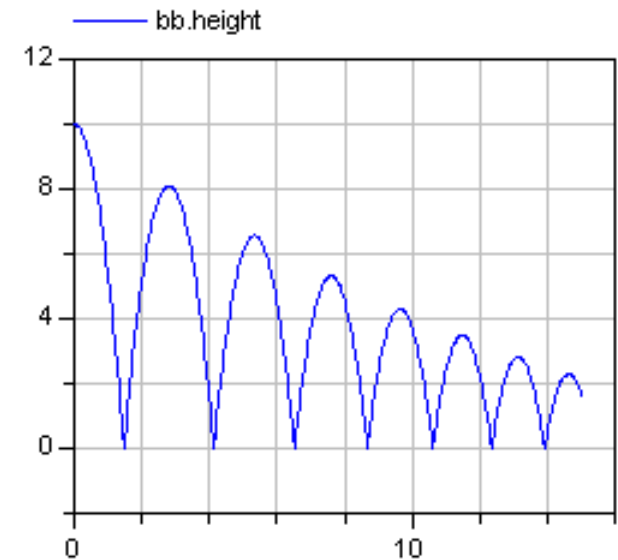
```
class ClockTicks
  // Integer quotient rational number interval clock
  Clock c1 = Clock(3,10);      // ticks: 0, 3/10, 6/10, ..
  // Clock with real value interval between ticks
  Clock c2 = Clock(0.2);      // ticks: 0.0, 0.2, 0.4, ...
end ClockTicks;
```



# 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;
```

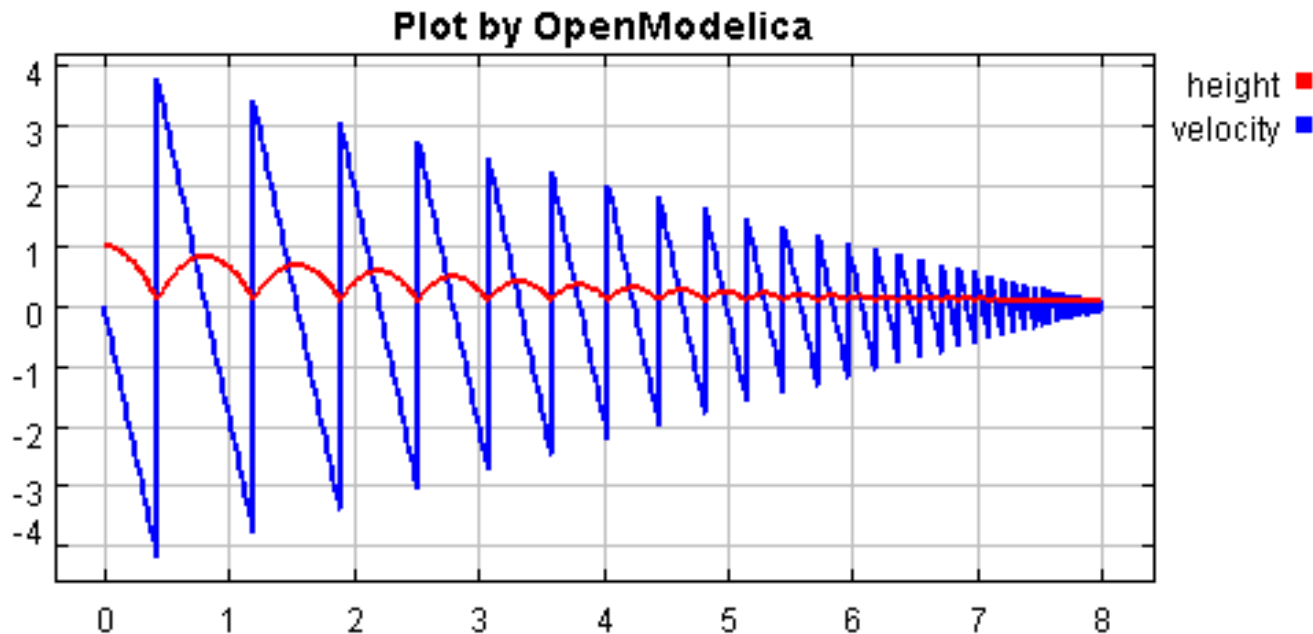


Initial conditions

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**

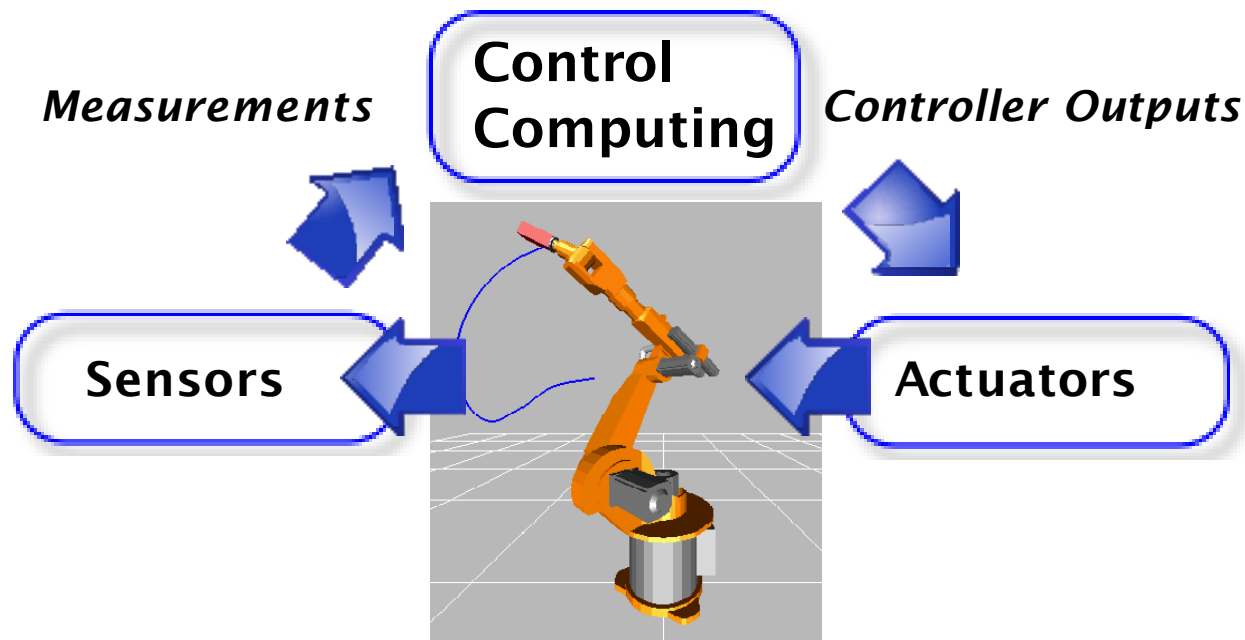
# **Clocked Synchronous Models and State Machines**

## **and Applications for Digital Controllers**

# Control System Applications

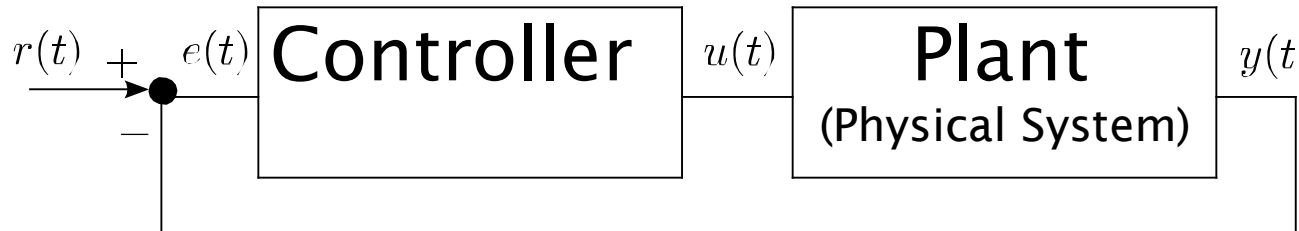
## Control System

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

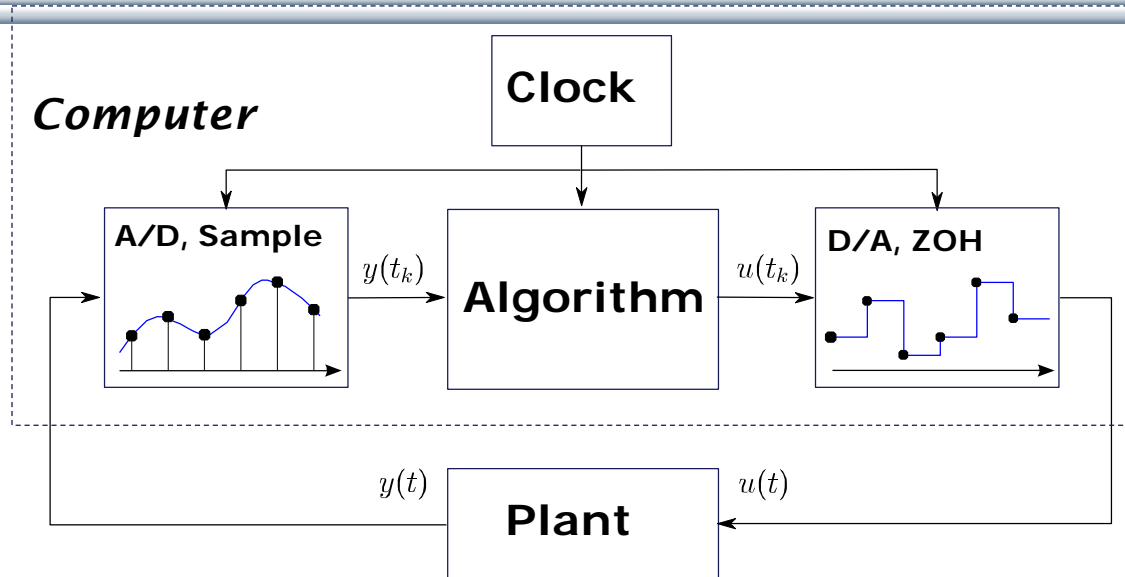


- $r(t)$  reference (setpoint)
- $e(t)$  error
- $y(t)$  measured process variable (plant output)
- $u(t)$  control output variable (plant input)

### Usual Objective

Plant output should follow the reference signal.

# Embedded Real-Time Control System

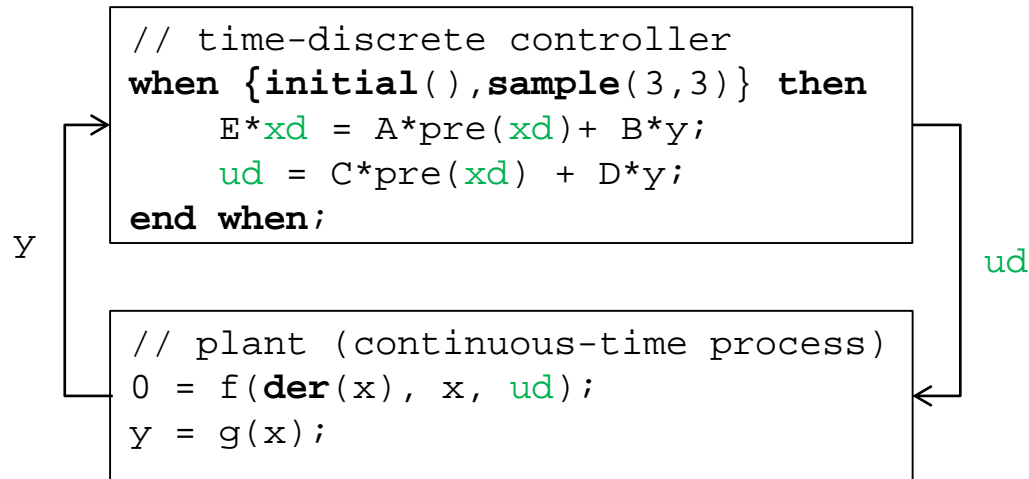


1. **Discrete-time** controller + **continuous-time** plant  $\equiv$  *hybrid system* or *sampled-data system*
2. Interface between digital and analog world: Analog to Digital and Digital to Analog Converters (ADC and DAC).
3. ADC  $\rightarrow$  Algorithm  $\rightarrow$  DAC is synchronous (zero-delay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



# Controller with Sampled Data-Systems

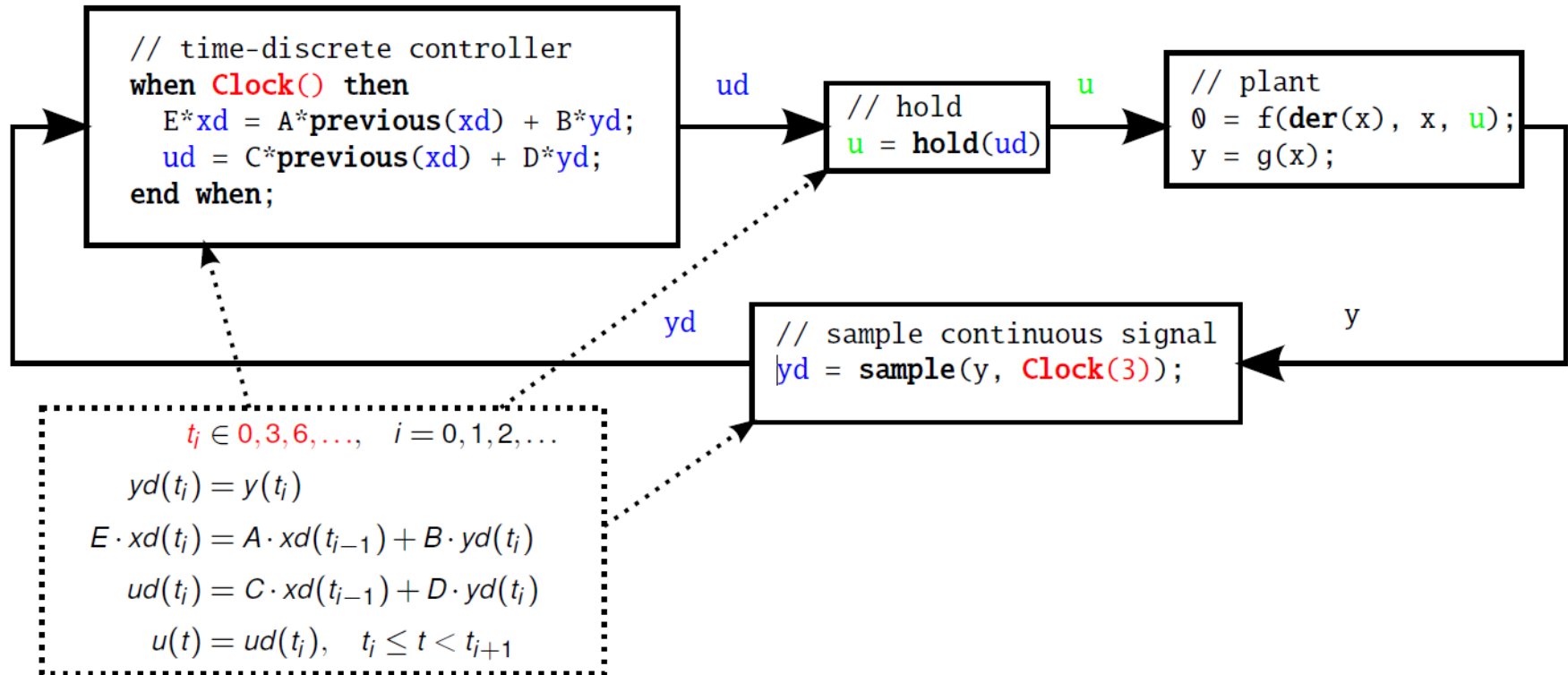
(unlocked models, using `pre()` and `sample()` )



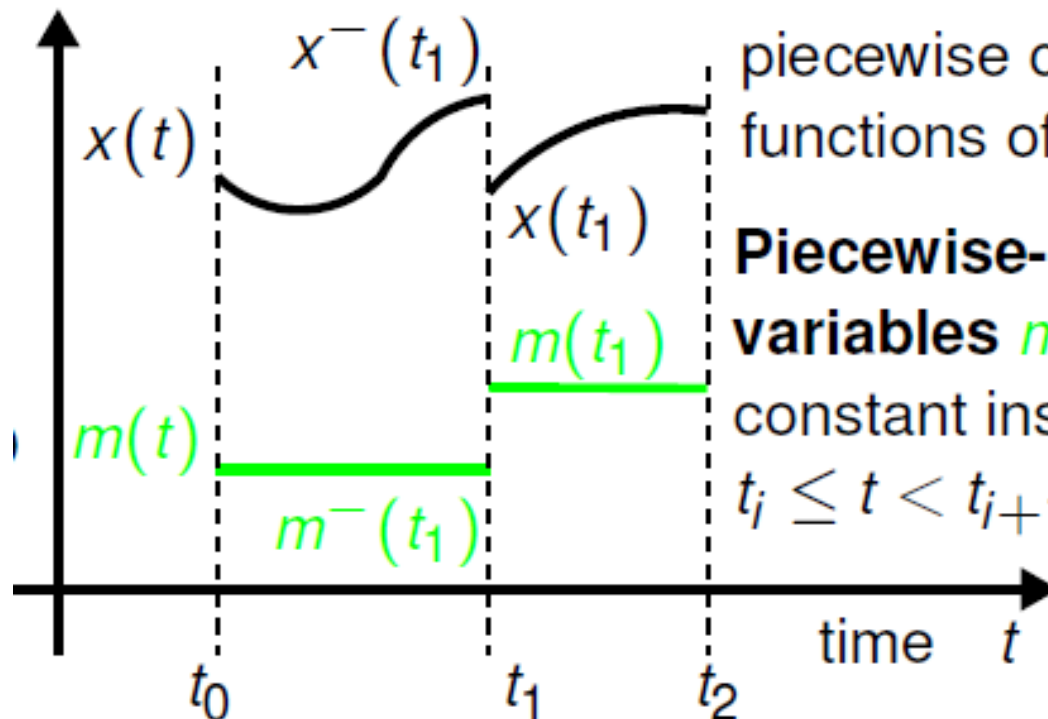
- `y` is automatically sampled at  $t = 3, 6, 9, \dots$ ;
- `xd`, `u` are piecewise-constant variables that change values at sampling events (implicit zero-order hold)
- `initial()` triggers event at initialization ( $t=0$ )

# Controller with Clocked Synchronous Constructs

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



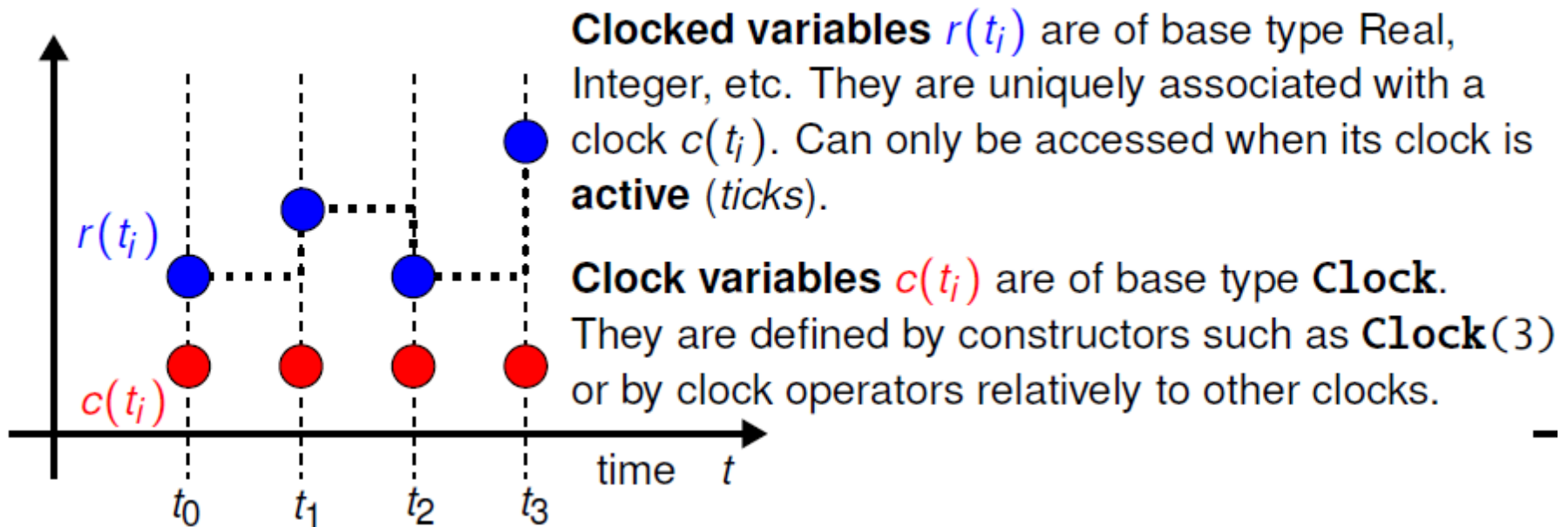
# Unlocked Variables in Modelica 3.2



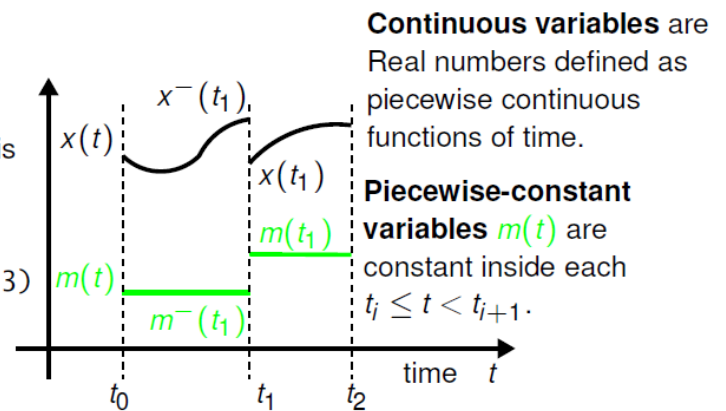
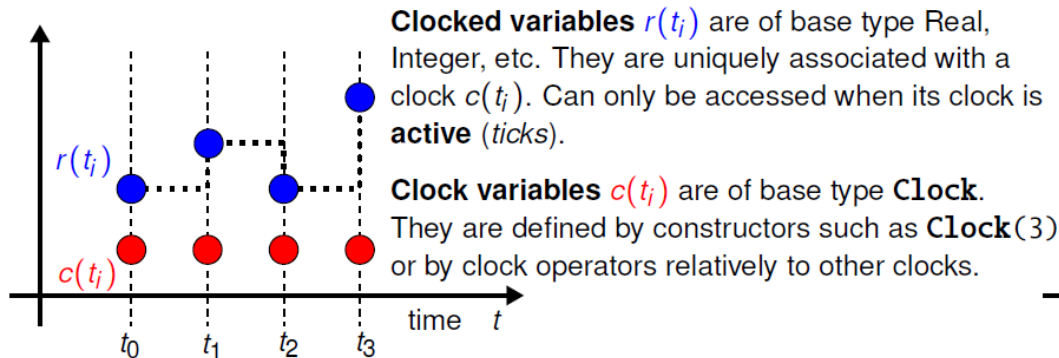
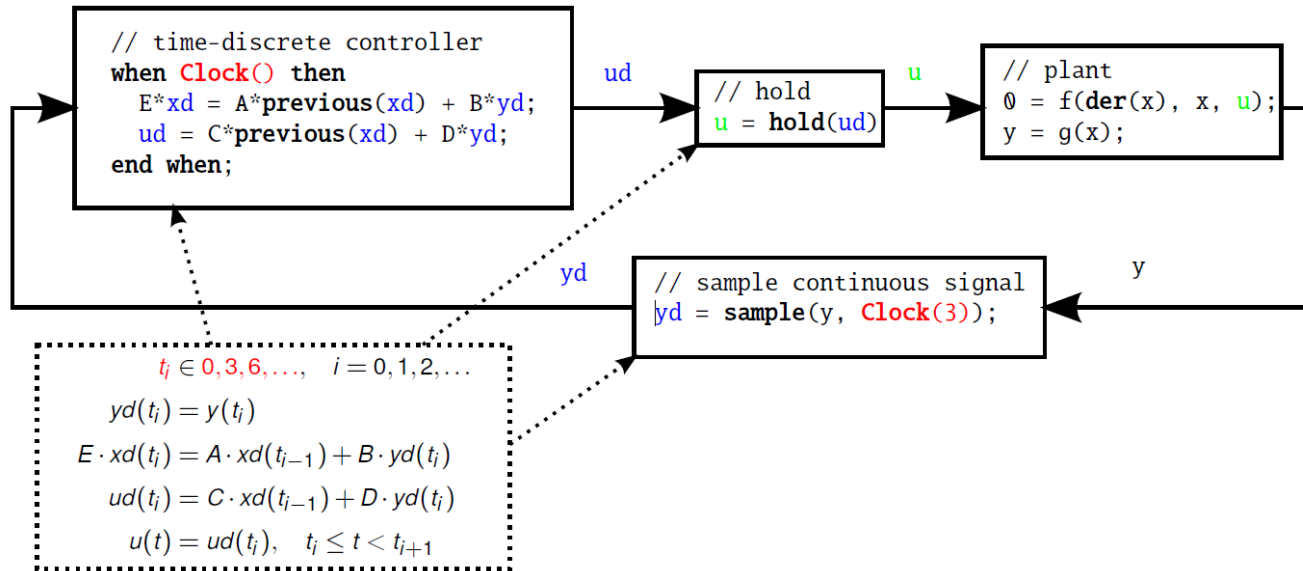
**Continuous variables** are Real numbers defined as piecewise continuous functions of time.

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

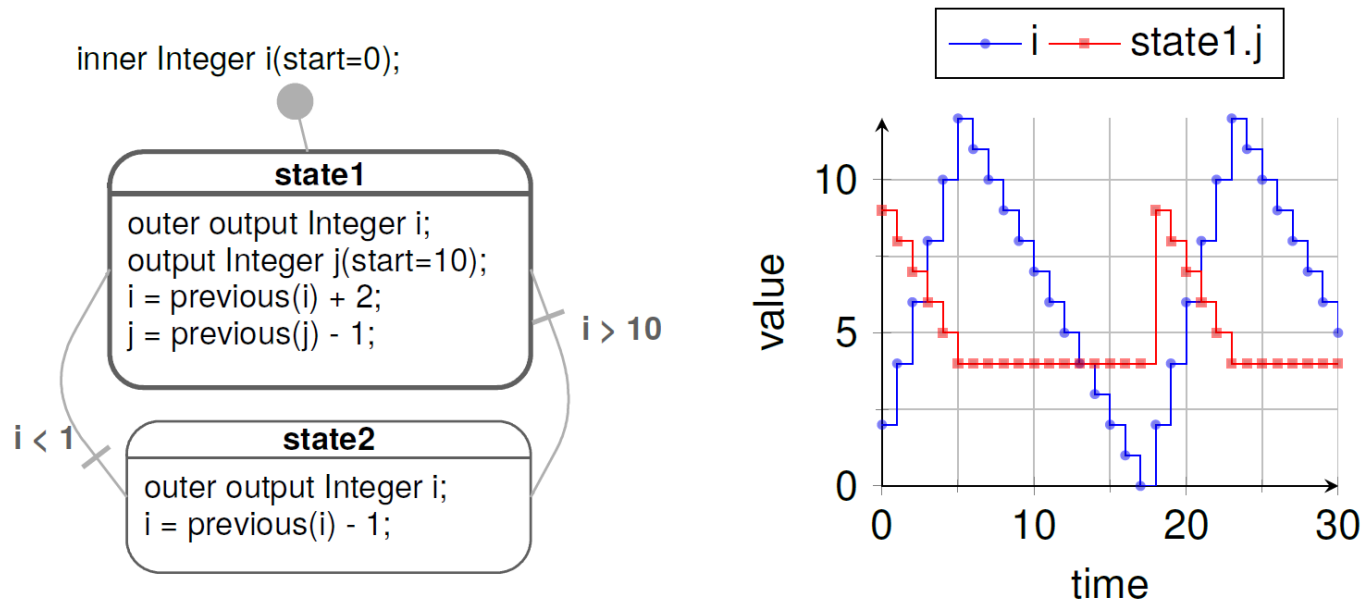
# Clock variables (Clock) and Clocked Variables (Real) (in Modelica 3.3)



# Clocked Synchronous Extension in Modelica 3.3



# State Machines in Modelica 3.3: Simple Example



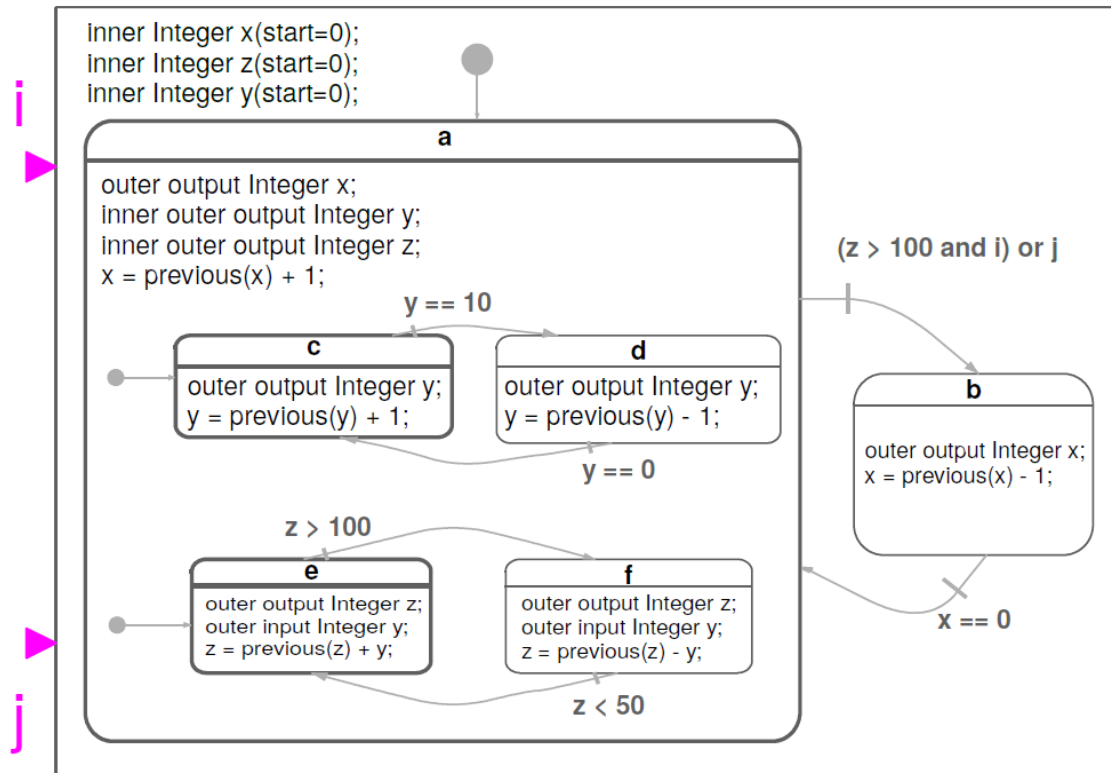
- Equations are active if corresponding *clock* ticks. Defaults to periodic clock with 1.0 s sampling period
- “i” is a shared variable, “j” is a local variable. Transitions are “*delayed*” and enter states by “*reset*”

# Simple Example: Modelica Code

```
model Simple_NoAnnotations "Simple state machine"
  inner Integer i(start=0);
  block State1
    outer output Integer i;
    output Integer j(start=10);
  equation
    i = previous(i) + 2;
    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;
```

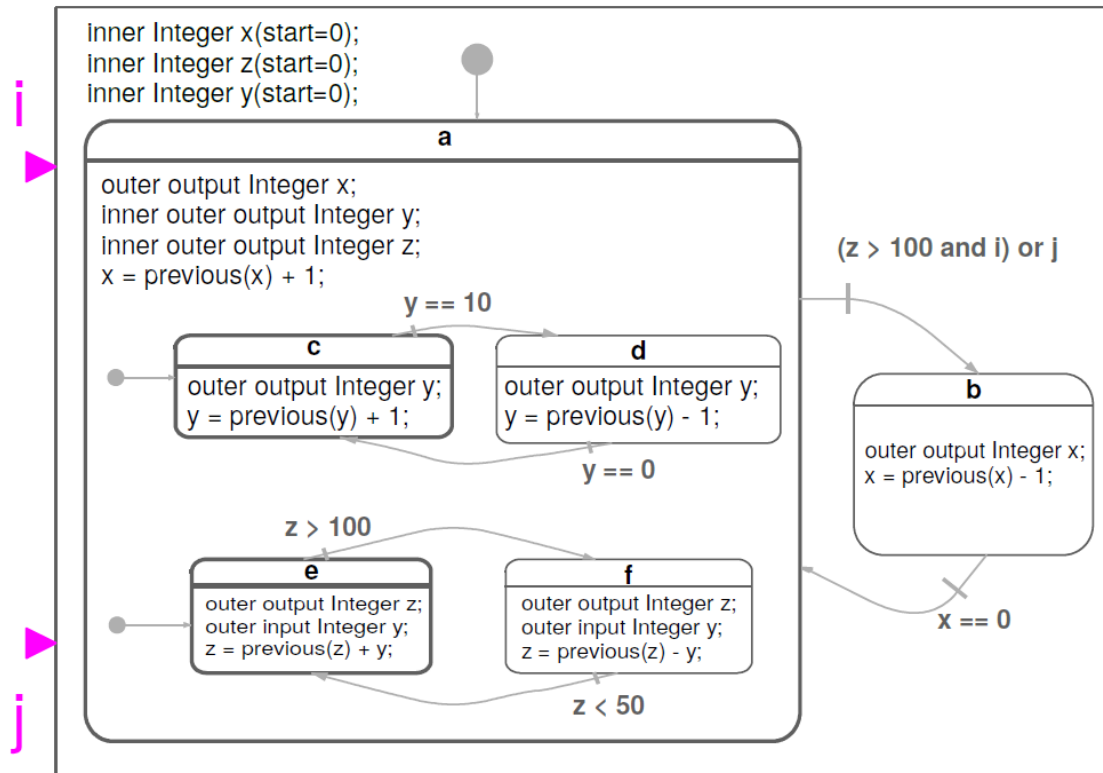


# Hierarchical and Parallel Composition of Modelica State Machine Models



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.

# Hierarchical and Parallel Composition



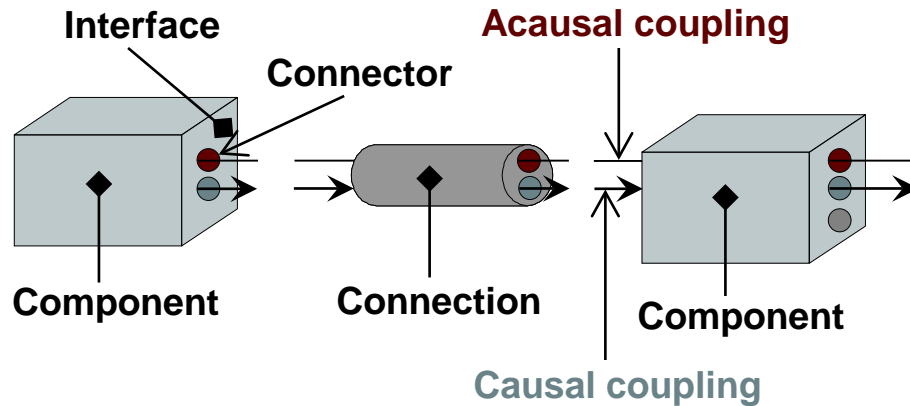
Semantics of Modelica state machines (and example above) inspired by Florence Maraninchi & Yann Rémond's "Mode-Automata" and by Marc Pouzet's Lucid Synchrone 3.0.

---

## Part IV

# Components, Connectors and Connections – Modelica Libraries and Graphical Modeling

# Software Component Model



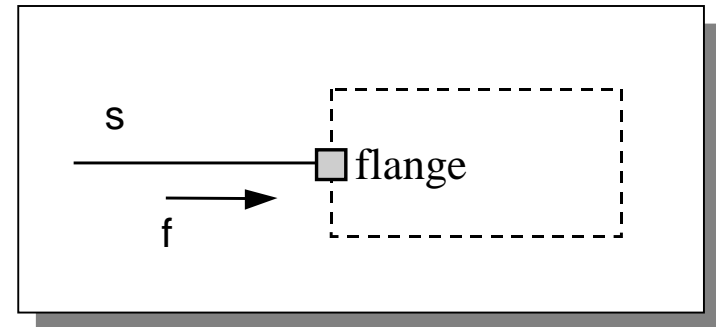
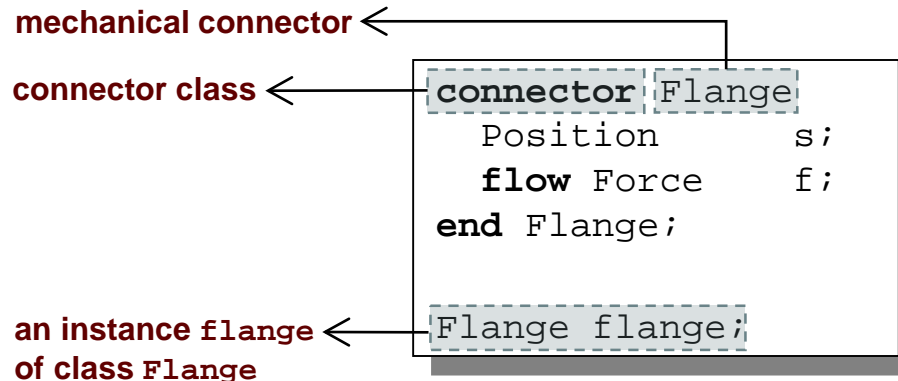
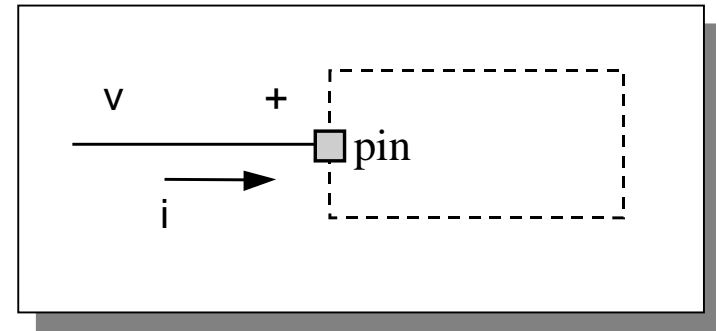
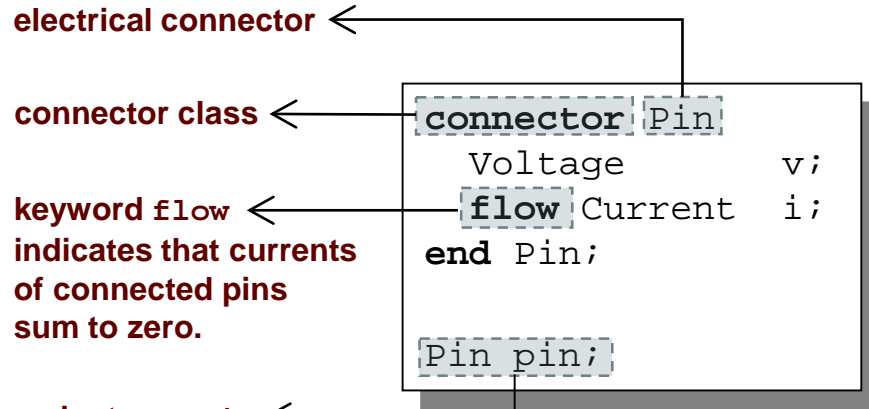
A component class should be defined *independently of the environment*, very essential for *reusability*

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:

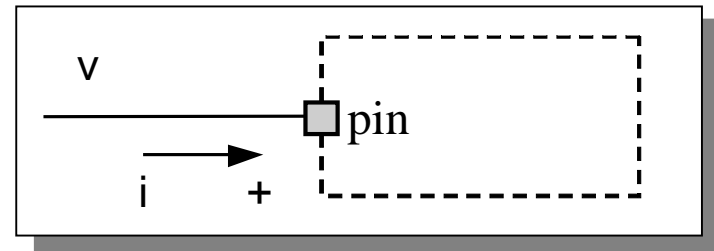
- *Potential variables* *potential* or energy level
- *Flow variables* represent some kind of flow
- *Stream variables* represent fluid flow in convective transport

## 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	Magnetic
Hydraulic	Pressure	Volume flow	Volume	OpenHydraulics
Heat	Temperature	Heat flow	Heat	HeatFlow1D
Chemical	Chemical potential	Particle flow	Particles	Chemical
Pneumatic	Pressure	Mass flow	Air	PneuLibLight

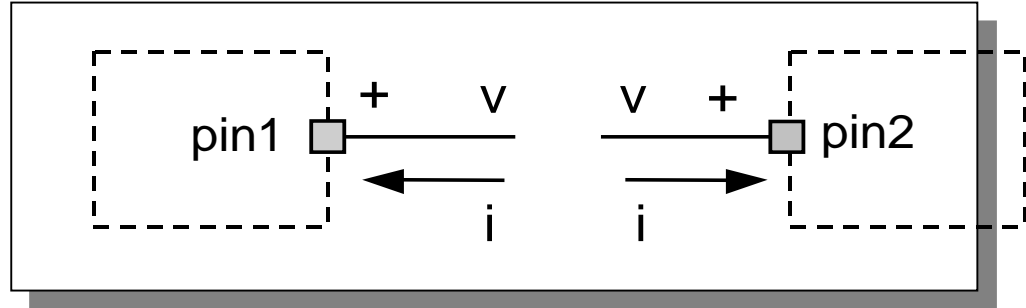


# connect-equations

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:

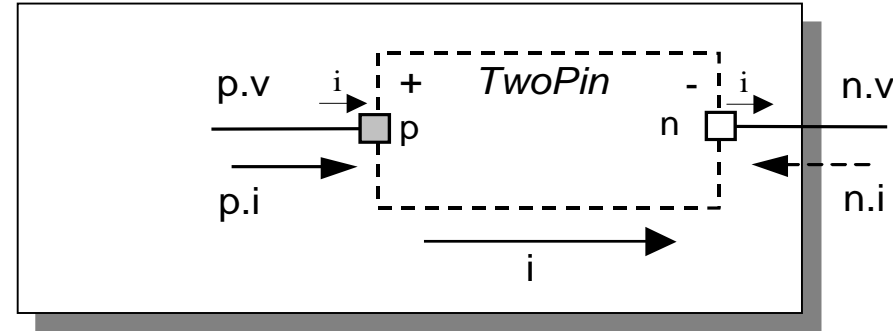
$$v_1 = v_2 = v_3 = \dots v_n$$

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;
```

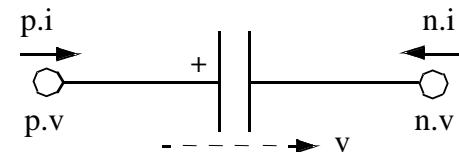
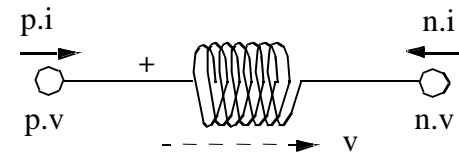
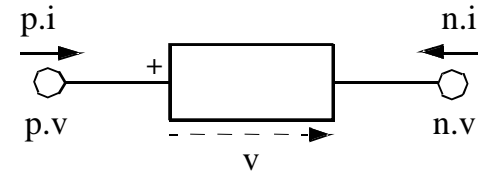
electrical connector class

# Electrical Components

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

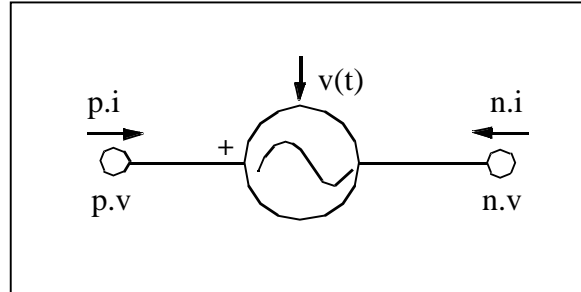
```
model Inductor "Ideal electrical inductor"
  extends TwoPin;
  parameter Real L "Inductance";
equation
  L*der(i) = v;
end Inductor;
```

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

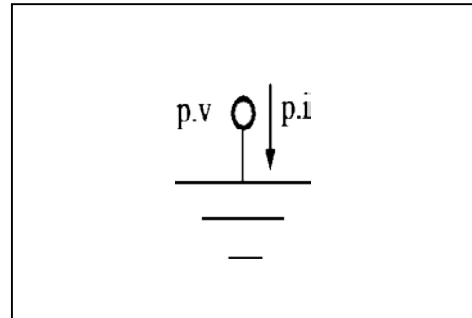


# Electrical Components cont'

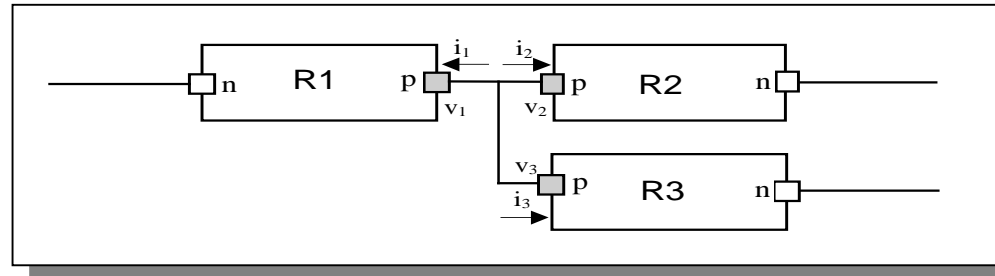
```
model Source
  extends TwoPin;
  parameter Real A,w;
equation
  v = A*sin(w*time);
end Resistor;
```



```
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 non-commercial purposes under the conditions of *The Modelica License*.
- Modelica libraries are available online including documentation and source code from <https://modelica.org/libraries.html>



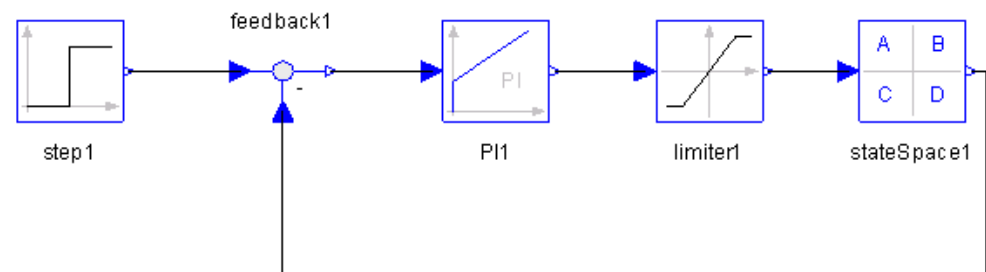
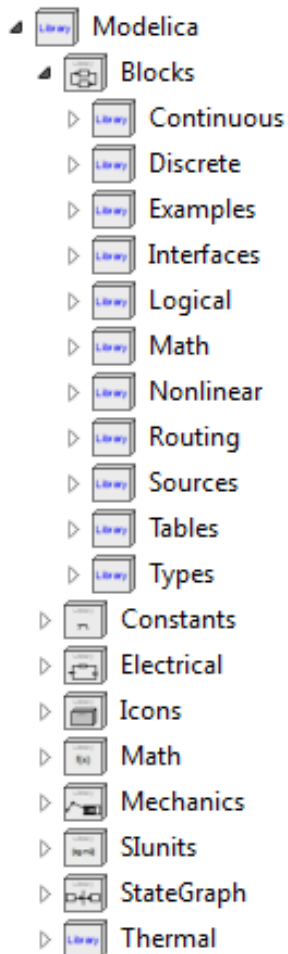
# Modelica Standard Library cont'

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

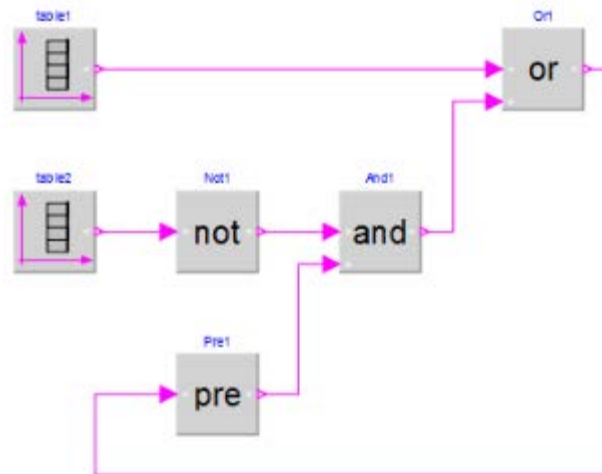
- **Blocks**            Library for basic input/output control blocks
- **Constants**       Mathematical constants and constants of nature
- **Electrical**       Library for electrical models
- **Icons**            Icon definitions
- **Fluid**            1-dim Flow in networks of vessels, pipes, fluid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

# Modelica.Blocks

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

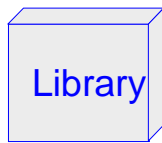


Examples:



# Modelica.Electrical

Electrical components for building analog, digital, and multiphase circuits



Analog



Digital

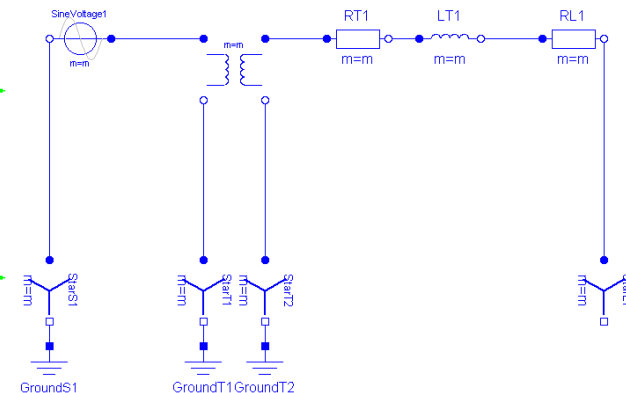
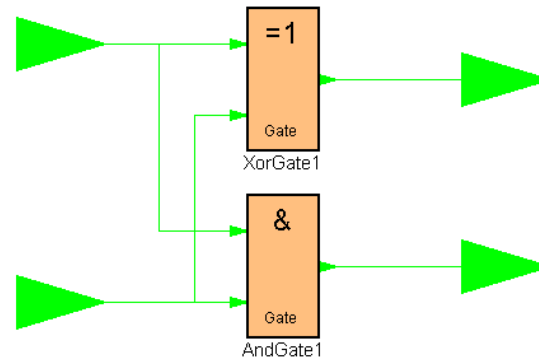
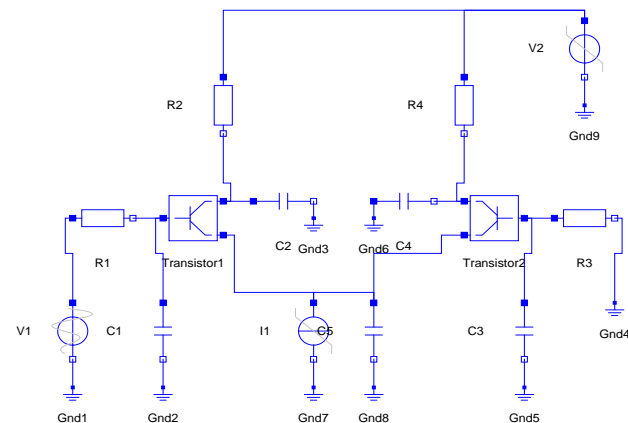


Machines



MultiPhase

Examples:

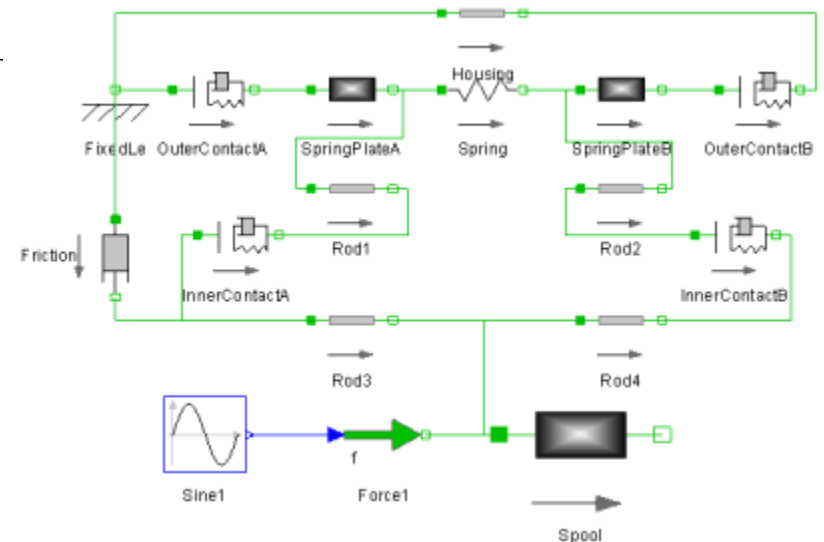
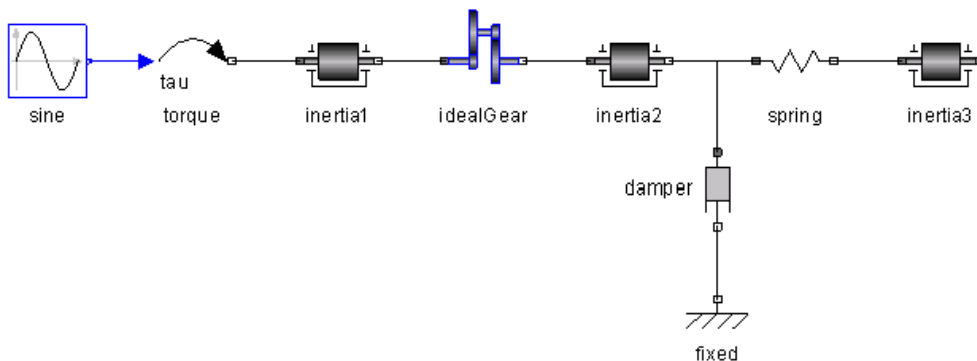


# Modelica.Mechanics

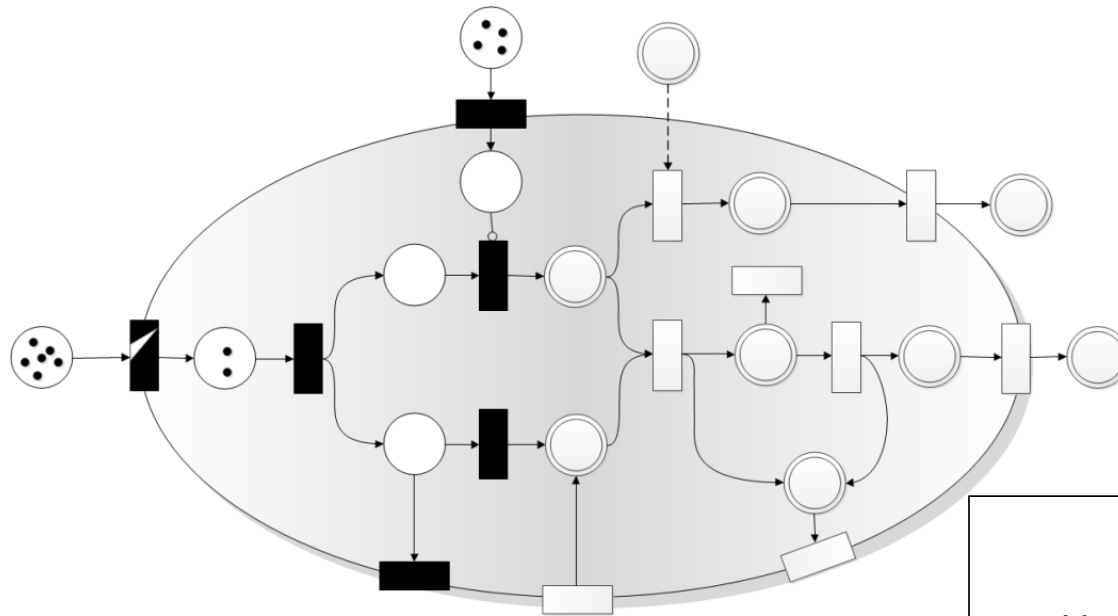
Package containing components for mechanical systems

Subpackages:

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






# PNlib - An Advanced Petri Net Library for Hybrid Process Modeling





## xHPN: Extended Hybrid Petri Nets



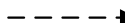

### Transitions

-  (time-)discrete process (event)
-  stochastic process (random event)
-  continuous process (flow)

### Places

-  (time-)discrete state (integer quantity)
-  continuous state (real quantity)

### Arcs

-  „normal“ arc
-  inhibitor arc
-  test arc
-  read arc

# Other Free Libraries

Up to date list at: <https://www.modelica.org/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
- PowerSystems Power systems in transient and steady-state mode
- ...

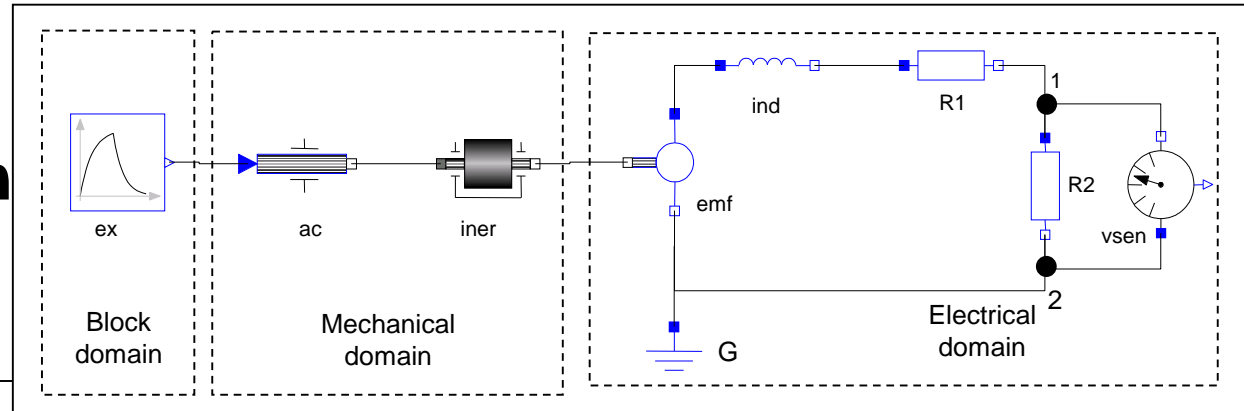
# Some Commercial Libraries

Up to date list at: <https://www.modelica.org/libraries>

- Air Conditioning
- Electric Power
- Fuel Cell
- Heat Exchanger
- Hydro Power
- Liquid Cooling
- Thermal Power
- Vapor Cycle
- Battery
- Belts
- Engine
- ...
- Powertrain
- SmartElectricDrives
- VehicleDynamics
- Hydraulics
- Pneumatics
- Engine Dynamics
- Environmental Control
- CombiPlant
- ...
- (there are many more)

# 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);
```

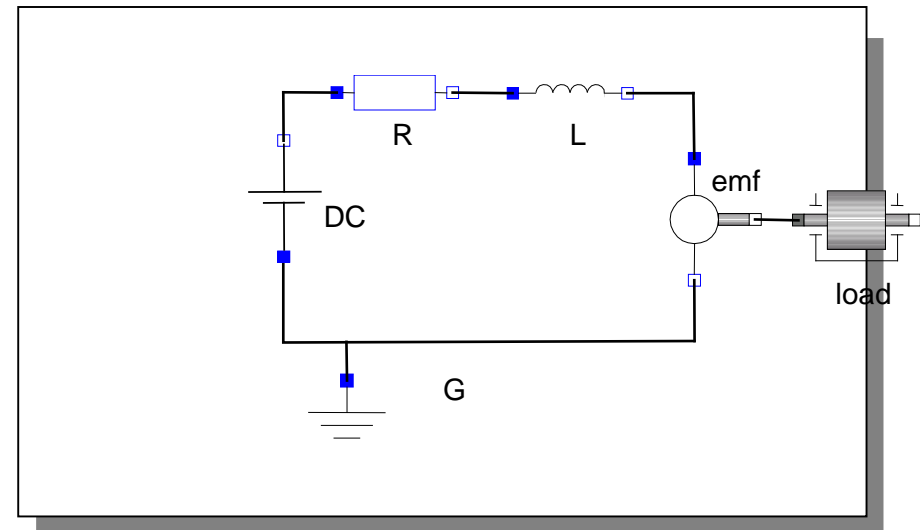
```
end Generator;
```



# 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 IV**

## **Sensitivity Analysis**

### **using**

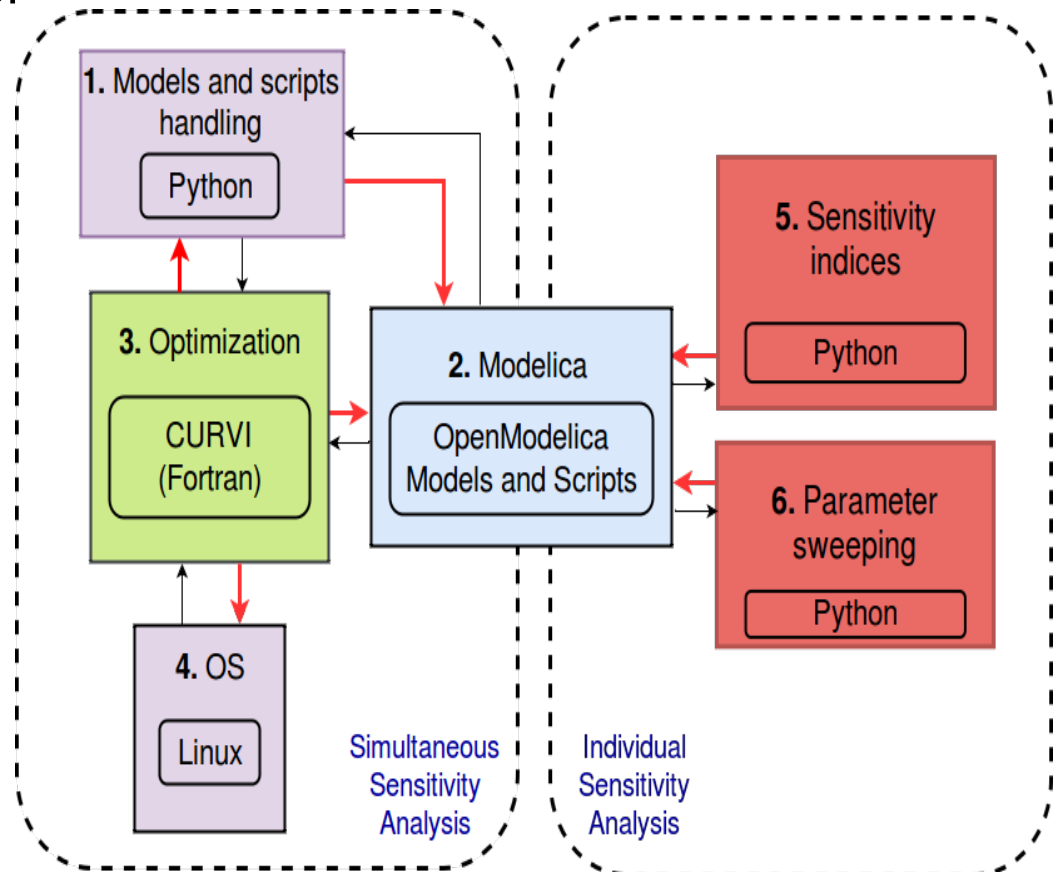
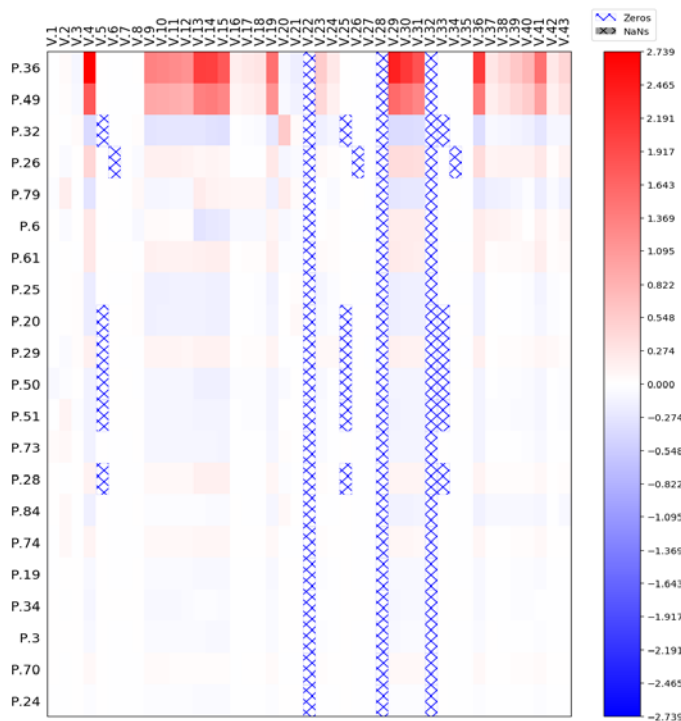
## **OpenModelica**

# OMSens – Multi-Parameter Sensitivity Analysis

- Individual and simultaneous multi-parameter analysis
- Optimization-based simultaneous analysis
- Robust derivative free optimizer

Tool architecture

Heatmap visualization



# Introduction to Sensitivity Analysis

- Sensitivity of nonlinear systems in the form of ODEs
  - Undergo noticeable **dynamic changes** in response to **small perturbations** in the parameters.
- OO-languages (Modelica)
  - Systematic treatment of the problem
  - Clear, unambiguous access to parameters, variables and simulation configuration.
  - Reusable frameworks to manipulate models as black boxes.
- Varied options to use internal knowledge about model structure



# Approaches to Sensitivity Analysis

- **Individual** analysis:
  - **One parameter** perturbed at a time
  - Ignores combinations of perturbations
- **Simultaneous** analysis:
  - All possible combinations not feasible
    - Would give combinatorial explosion of parameter settings
  - Find “**optimal**” **combinations** of perturbations
    - “Smallest simultaneous perturbations that produce largest deviations”
    - Typically: **optimization-based** strategies

# CURVIF: robust derivative-free optimization algorithm

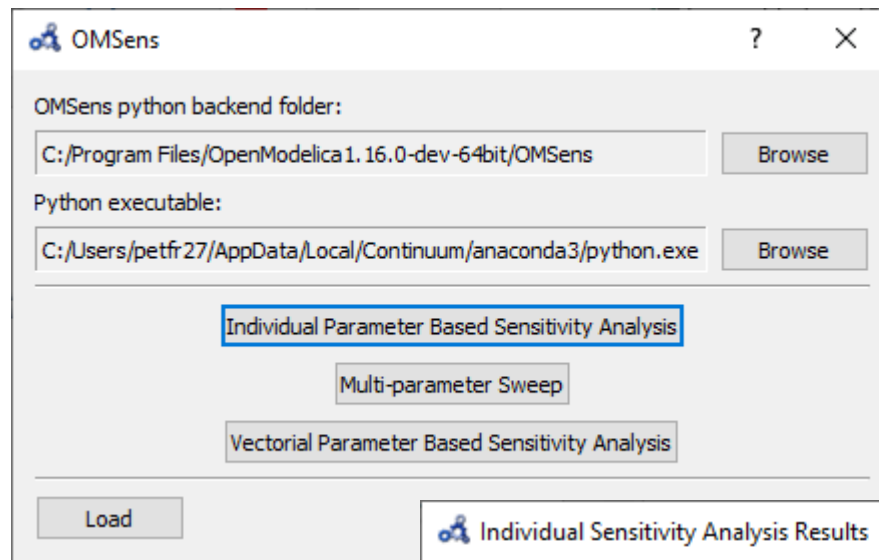
- The CURVI family
  - Curvilinear search approach
- Three versions: CURVIF, CURVIG, CURVIH
  - Function values, function values plus Gradients, and the latter plus Hessians.
  - Globally convergent
  - In general uses **fewer evaluations** than other algorithms
- CURVIF: the flavor adopted for OMSens
  - Trade-off: favor **robustness**, sacrifice some efficiency
  - Derivative-free methods can either be robust - at the cost of using many function evaluations, e.g. direct searches - or may present convergence problems

# LotkaVolterra – A Simple Model to be Used for Sensitivity Analysis Exercises

```
model LotkaVolterra "This is the typical equation-oriented model"
  parameter Real alpha=0.1 "Reproduction rate of prey";
  parameter Real beta=0.02 "Mortality rate of predator per prey";
  parameter Real gamma=0.4 "Mortality rate of predator";
  parameter Real delta=0.02 "Reproduction rate of predator per prey";
  parameter Real prey_pop_init=10 "Initial prey population";
  parameter Real pred_pop_init=10 "Initial predator population";
  Real prey_pop(start=prey_pop_init) "Prey population";
  Real pred_pop(start=pred_pop_init) "Predator population";
initial equation
  prey_pop = prey_pop_init;
  pred_pop = pred_pop_init;
equation
  der(preypop) = prey_pop*(alpha-beta*pred_pop);
  der(predpop) = pred_pop*(delta*prey_pop-gamma);
end LotkaVolterra ;
```

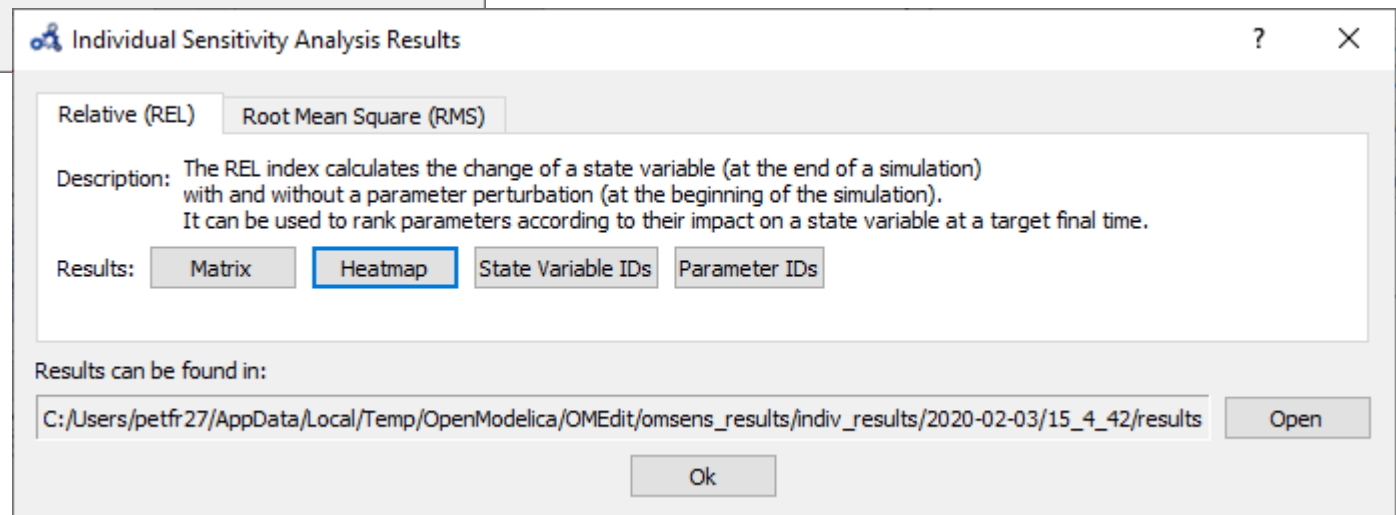
# OMSens Exercise – Locate Python

## Select Analysis type – OpenModelica 1.16.0 or later



### Installation instructions:

<https://github.com/OpenModelica/OMSens#omsens>

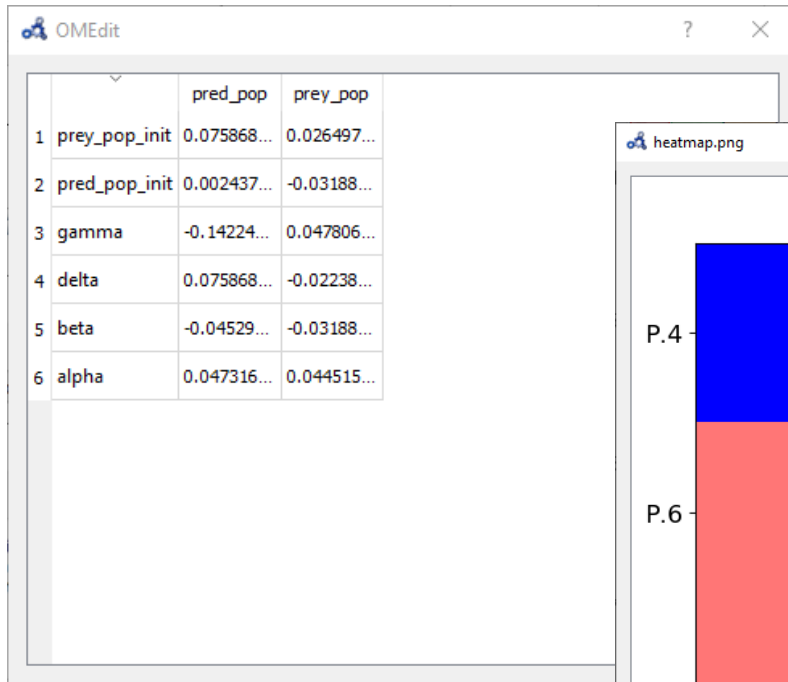




# OMSens Exercise –results from individual analysis

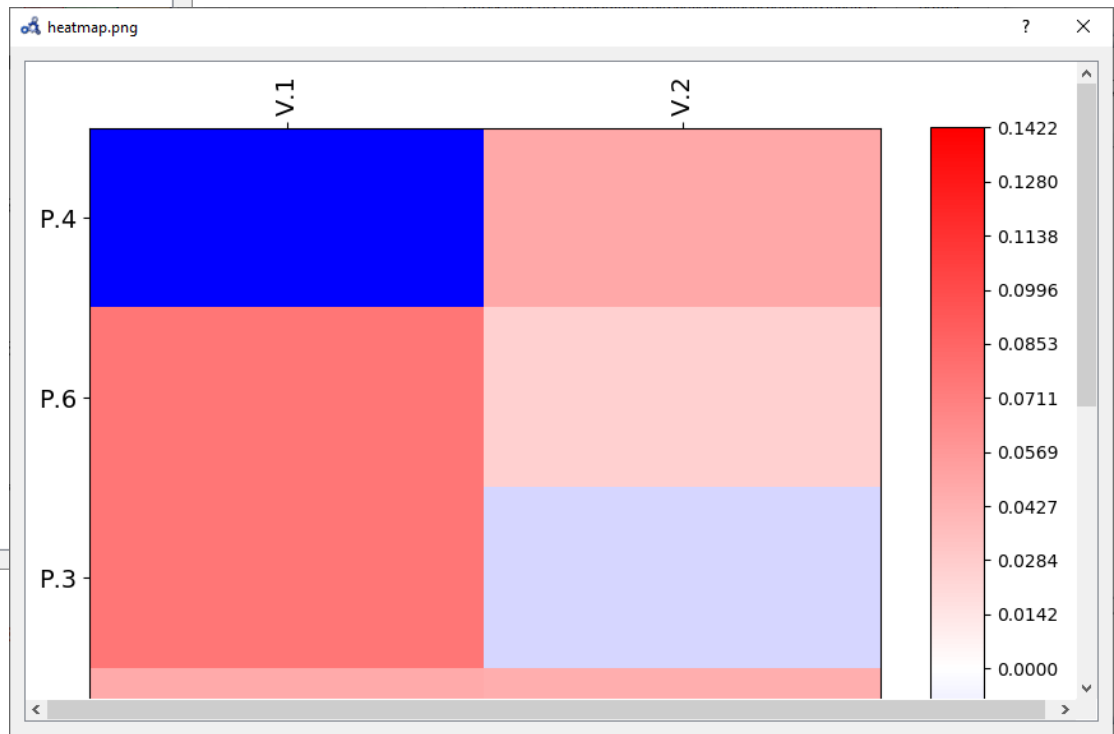
More info in the file:

OMSens Example\_Exercise\_ Lotka-Volterra.pdf



OMEdit

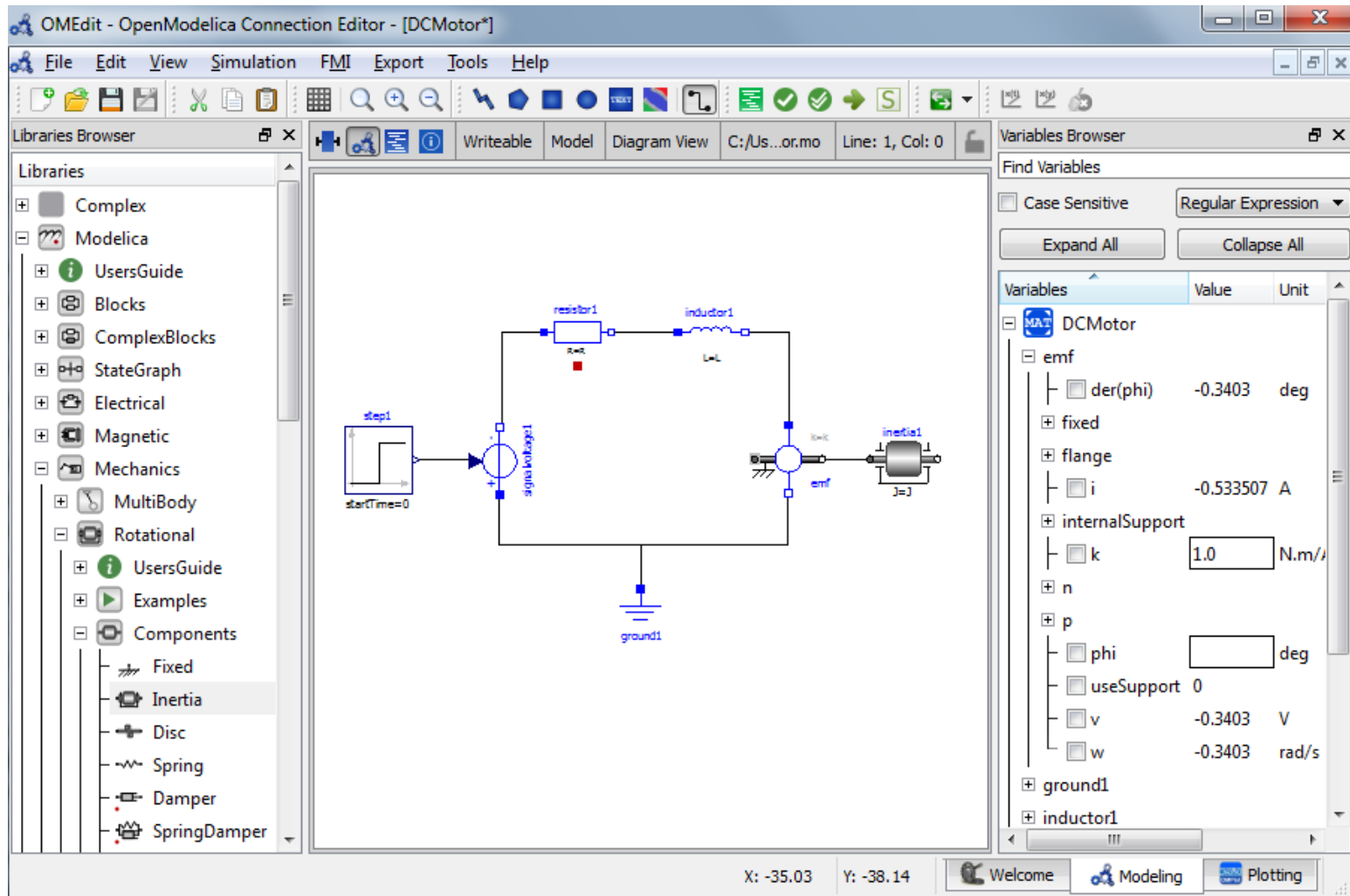
		pred_pop	prey_pop
1	prey_pop_init	0.075868...	0.026497...
2	pred_pop_init	0.002437...	-0.03188...
3	gamma	-0.14224...	0.047806...
4	delta	0.075868...	-0.02238...
5	beta	-0.04529...	-0.03188...
6	alpha	0.047316...	0.044515...



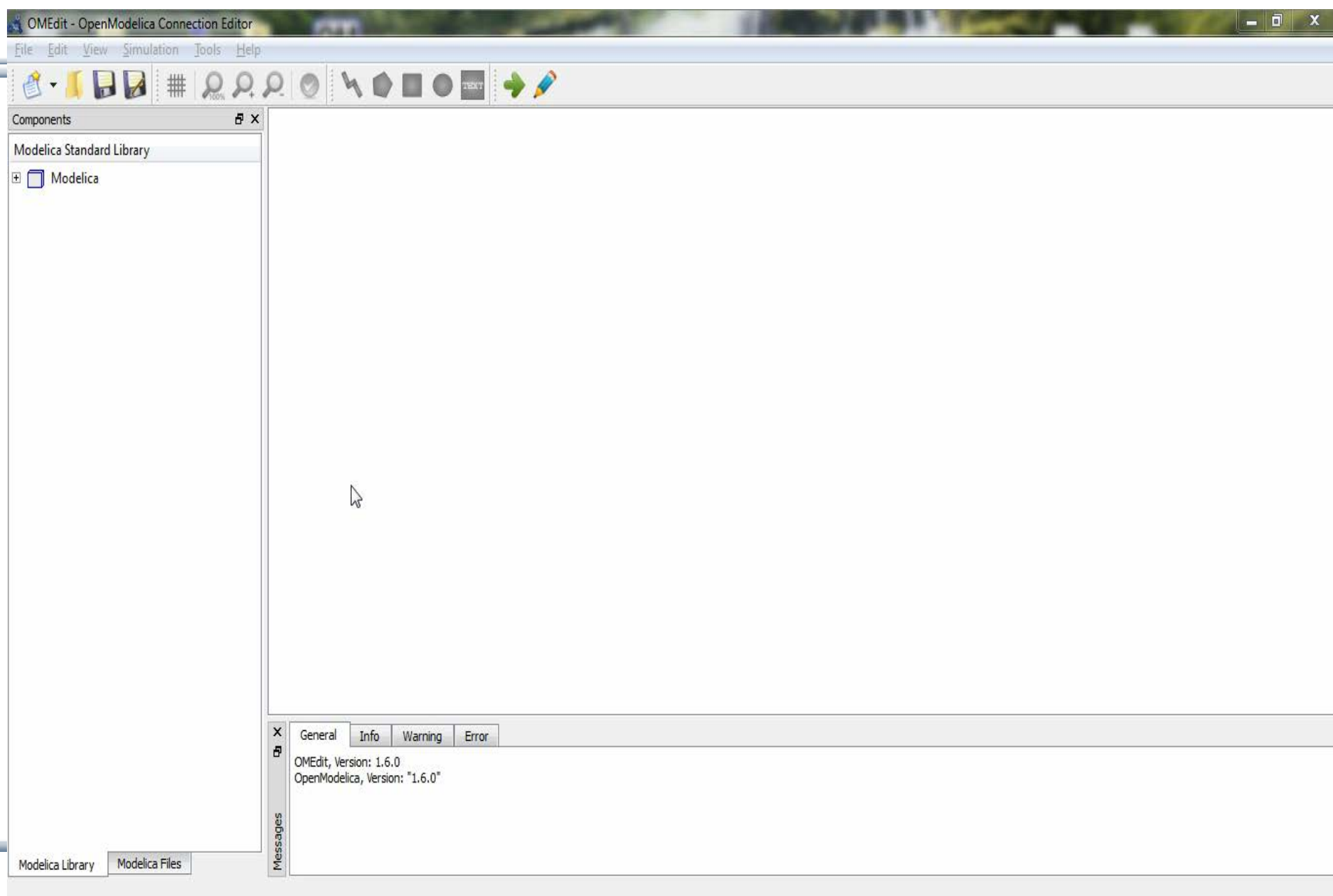
---

**Part Vb**  
**More**  
**Graphical Modeling Exercises**  
  
**using**  
**OpenModelica**

# Graphical Modeling - Using Drag and Drop Composition



# Graphical Modeling Animation – DCMotor



# Multi-Domain (Electro-Mechanical) Modelica Model

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

**model** DCMotor

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

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

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

```
Ground G;
```

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

```
Inertia load;
```

**equation**

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

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

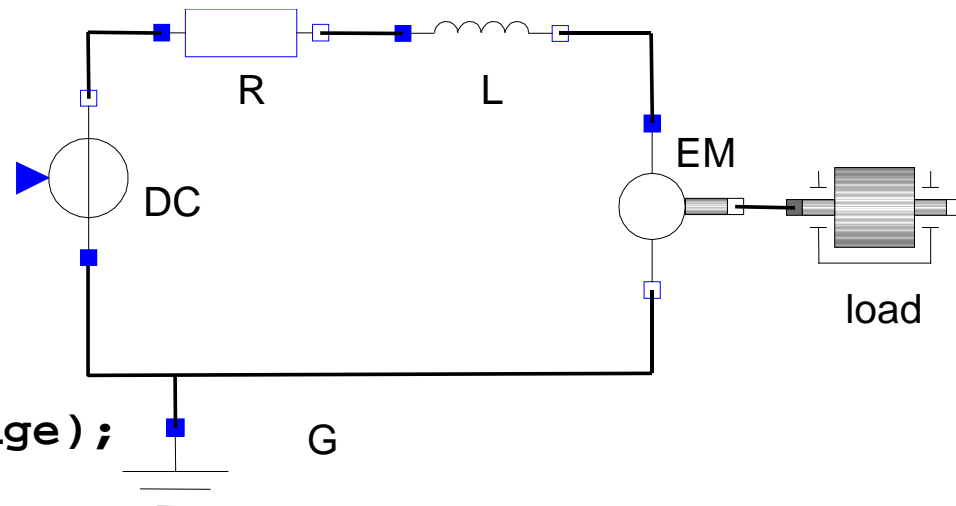
```
connect(L.p, EM.n);
```

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

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

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

**end** DCMotor



# Corresponding DCMotor Model Equations

The following equations are automatically derived from the Modelica model:

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

(load component not included)

Automatic transformation to ODE or DAE for simulation:

$$\frac{dx}{dt} == f[x, u, t]$$

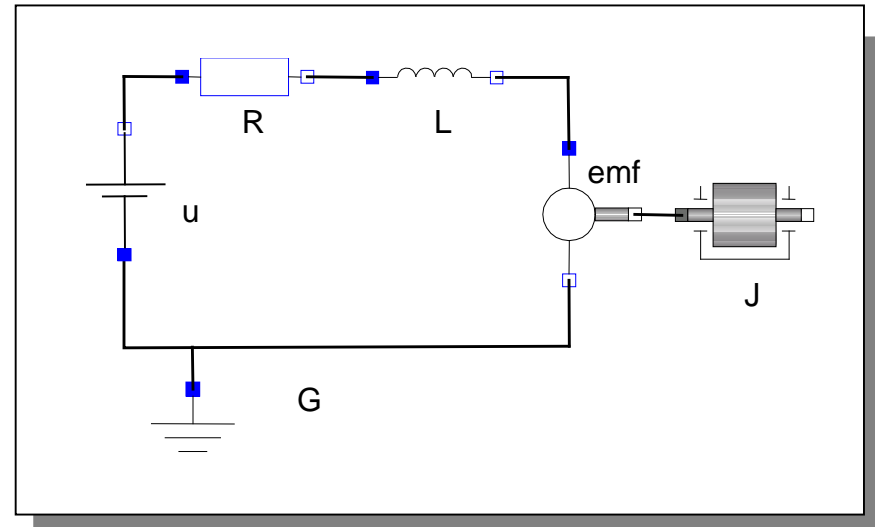
$$g\left[\frac{dx}{dt}, x, u, t\right] == 0$$

## Exercise 3.1

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

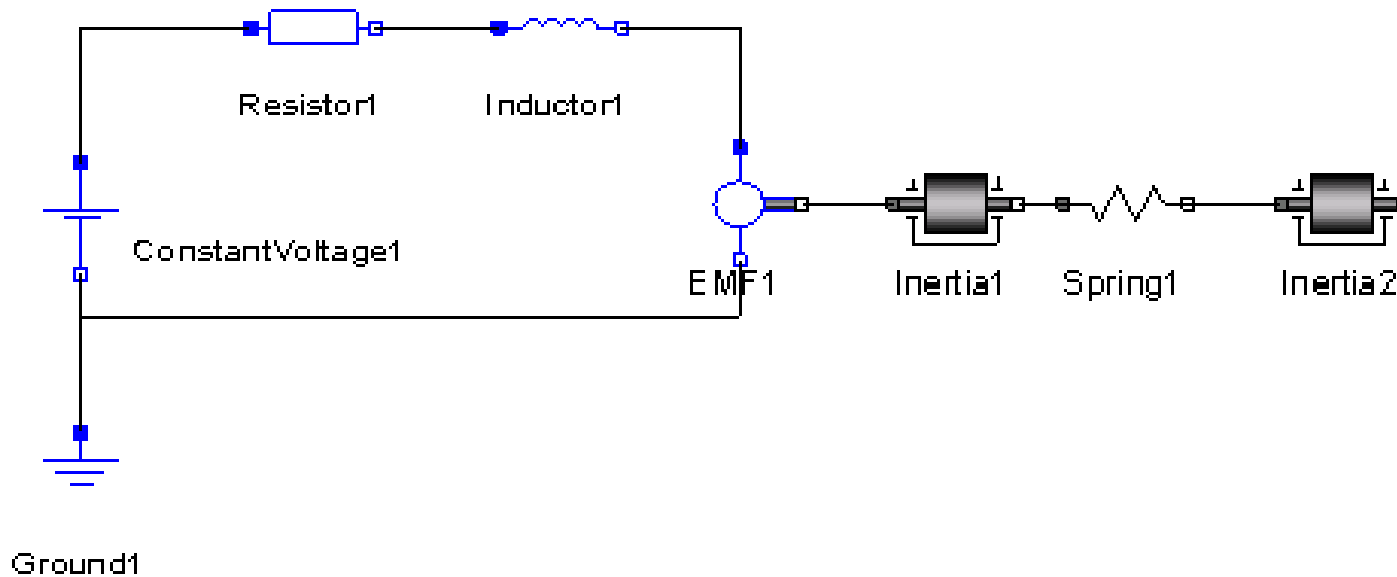
`Mechanics.Rotational.Components,`  
`Electrical.Analog.Basic,`  
`Electrical.Analog.Sources`

- 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

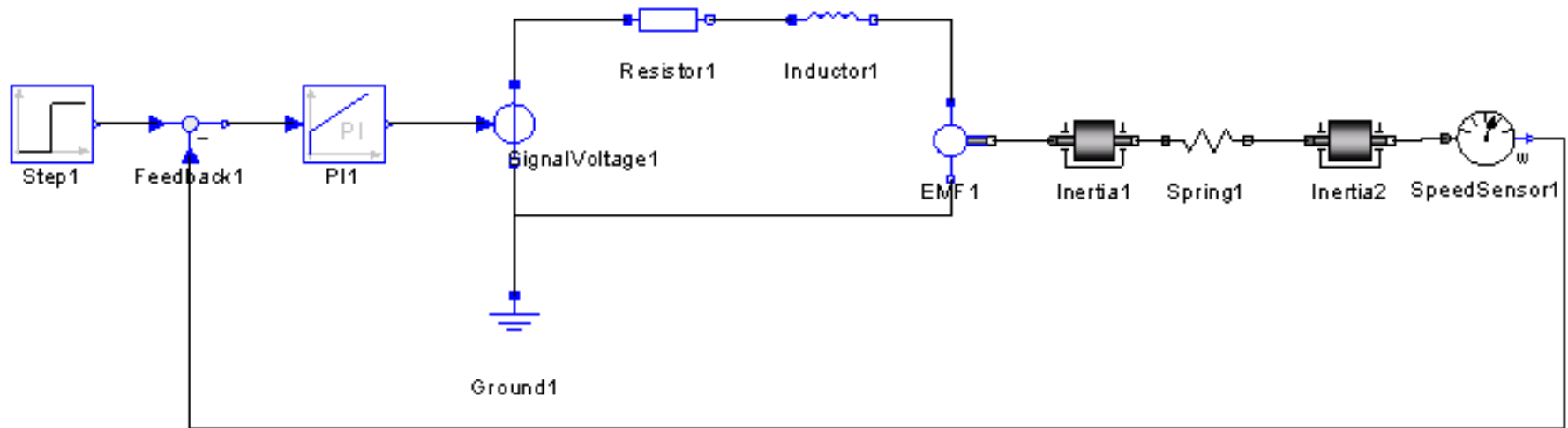
- If there is enough time: Add a torsional spring to the outgoing shaft and another inertia element. Simulate again and see the results. Adjust some parameters to make a rather stiff spring.



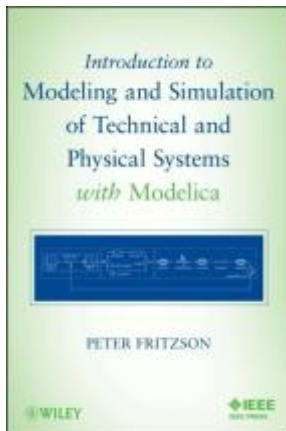
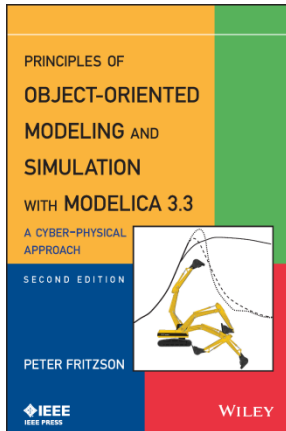


## Exercise 3.3

- 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.



# Learn more...



- OpenModelica
  - [www.openmodelica.org](http://www.openmodelica.org)
- Modelica Association
  - [www.modelica.org](http://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

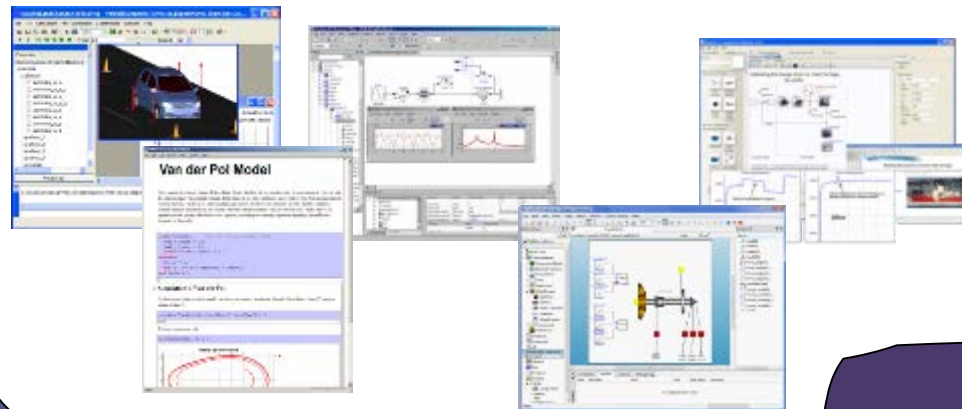
# Summary

Multi-Domain  
Modeling

  
M O D E L I C A

Visual Acausal  
Component  
Modeling

[www.OpenModelica.org](http://www.OpenModelica.org)  
[OpenModelica@ida.liu.se](mailto:OpenModelica@ida.liu.se)



Typed  
Declarative  
Textual Language

**Thank you for listening!**

Hybrid  
Modeling