

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

Tutorial 2025-09-08 at Modelica2025 in Lucerne

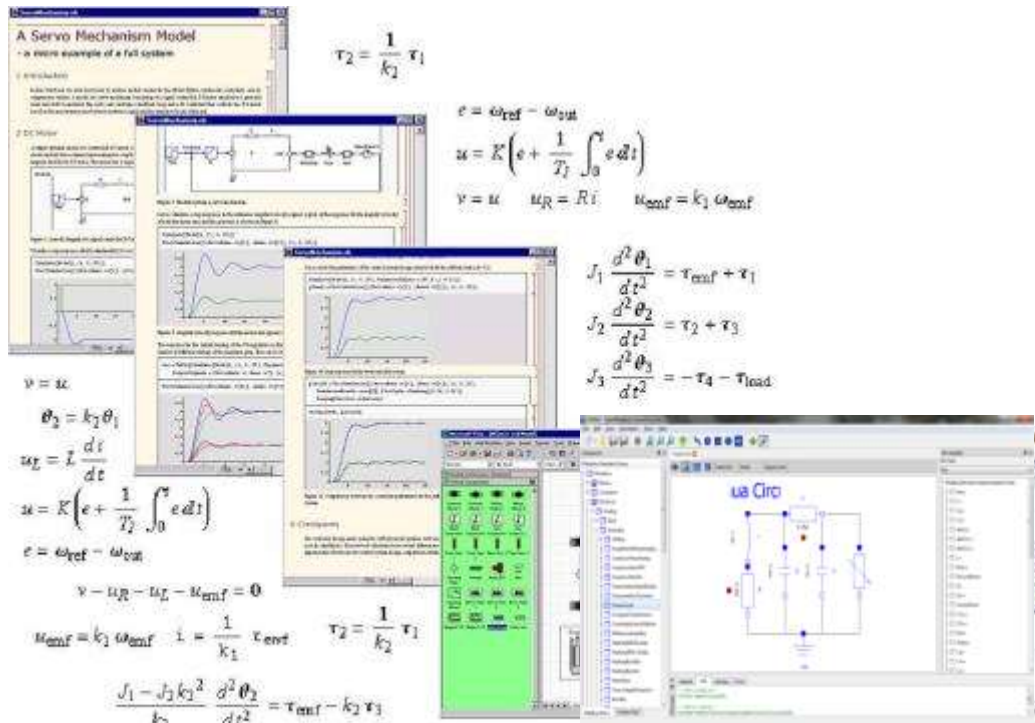
Peter Fritzson

Professor em. at Linköping University, 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



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

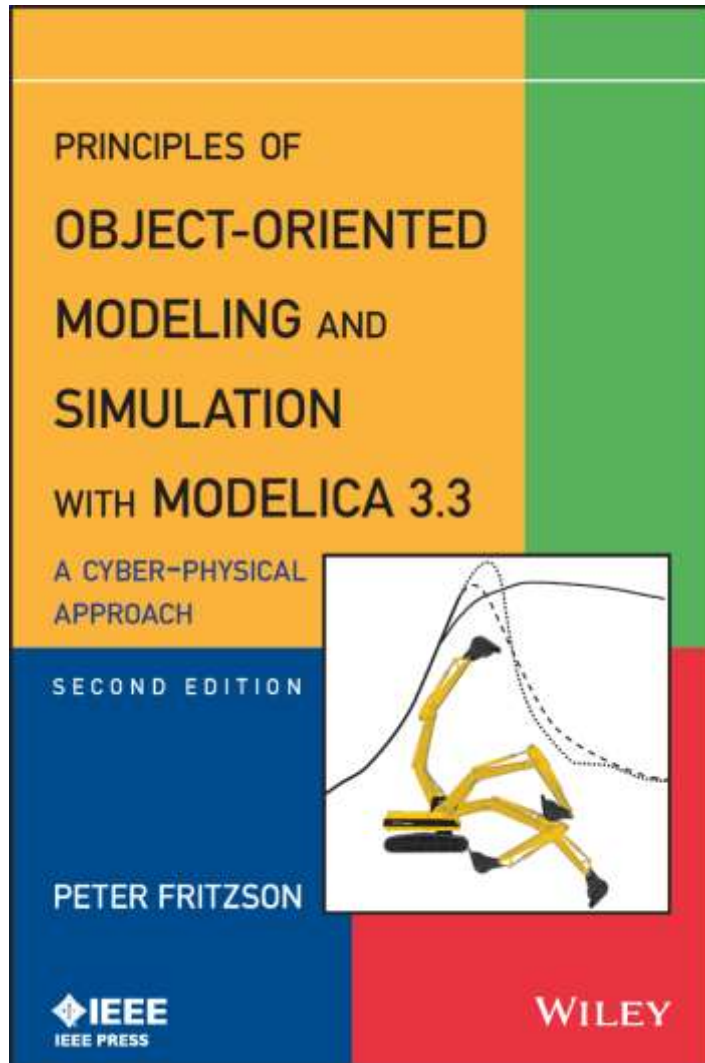
Contributions 2011 by Peter F., Mohsen T., Adeel Asghar,

Contributions 2012-2018 by Peter Fritzson, Lena Buffoni, Mahder Gebremedhin, Bernhard Thiele, Lennart Ochel

Contributions 2019-2025 by Peter Fritzson, Arunkumar Palanisamy, Bernt Lie, Adrian Pop, John Tinnerholm

Tutorial Based on Book, December 2014

Download OpenModelica Software



Peter Fritzson

Principles of Object Oriented Modeling and Simulation with Modelica 3.3

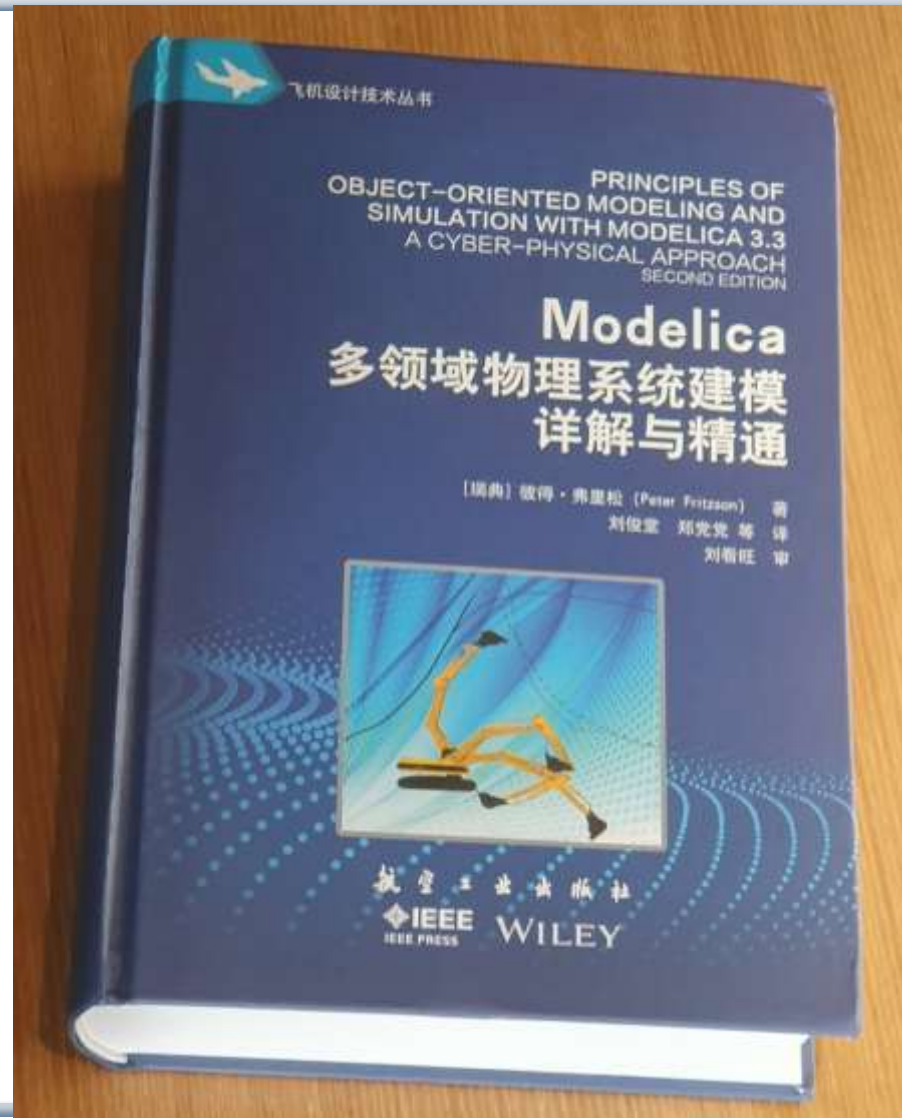
A Cyber-Physical Approach

Can be ordered from Wiley or Amazon

Wiley-IEEE Press, 2014, 1250 pages

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

Chinese Translation Published year 2021



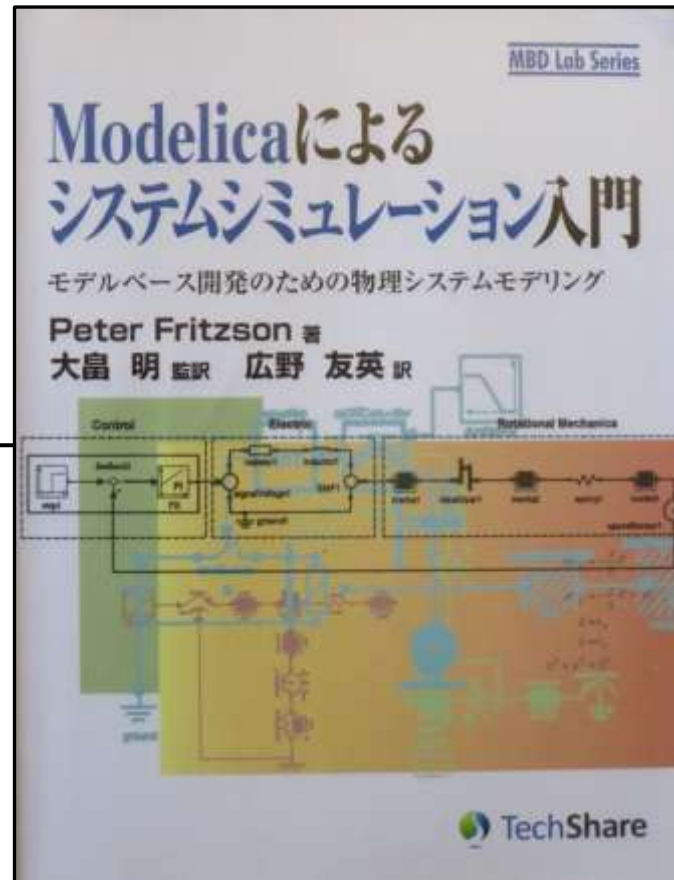
Introductory Modelica Book

September 2011
232 pages

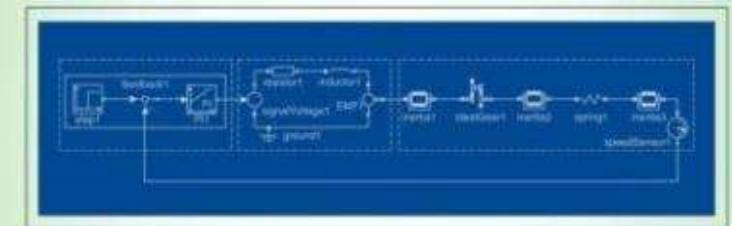
Translations
available in
Chinese,
Japanese,
Spanish

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
- Thanks to Emma Larsdotter Nilsson, Peter Bunus, David Broman, Jan Brugård, Mohsen-Torabzadeh-Tari, Adeel Asghar, Lena Buffoni, Adrian Pop, Arunkumar Palanisamy, John Tinnerholm 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
- OpenModelica: www.openmodelica.org

Outline

Part I

Introduction to Modelica and a demo example



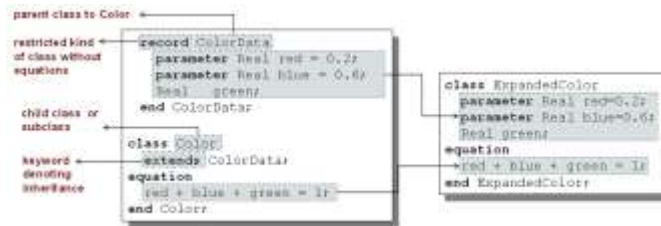
Part II

Modelica environments



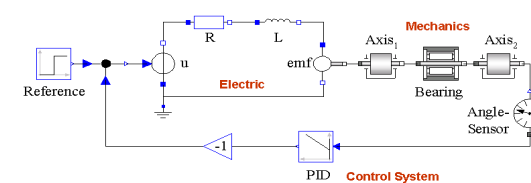
Part III

Modelica language concepts and textual modeling



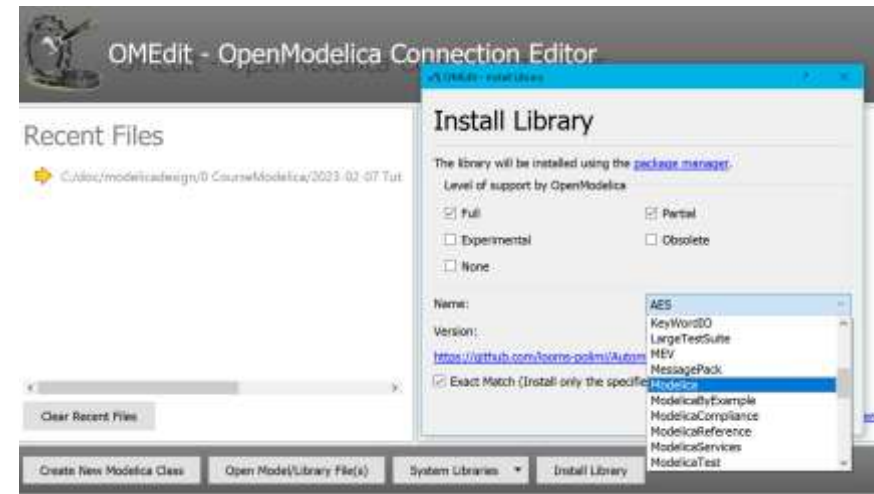
Part IV

Graphical modeling and the Modelica standard library



Software Installation - Windows

- Start the software installation
- Install OpenModelica-1.25.2 or later Download from www.openmodelica.org (takes about 20min)
- The Modelica library is typically loaded by default
- Otherwise you also need to load the Modelica standard library if not already loaded:
(push the load library button and select Modelica)



Software Installation – Linux (requires internet connection)

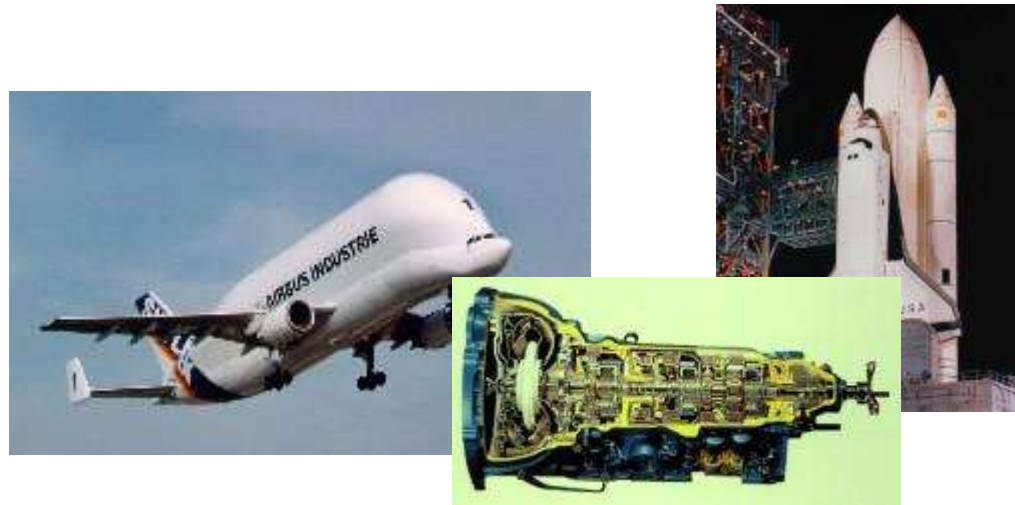
- Go to <https://openmodelica.org/index.php/download/download-linux> and follow the instructions.

Software Installation – MAC (requires internet connection)

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

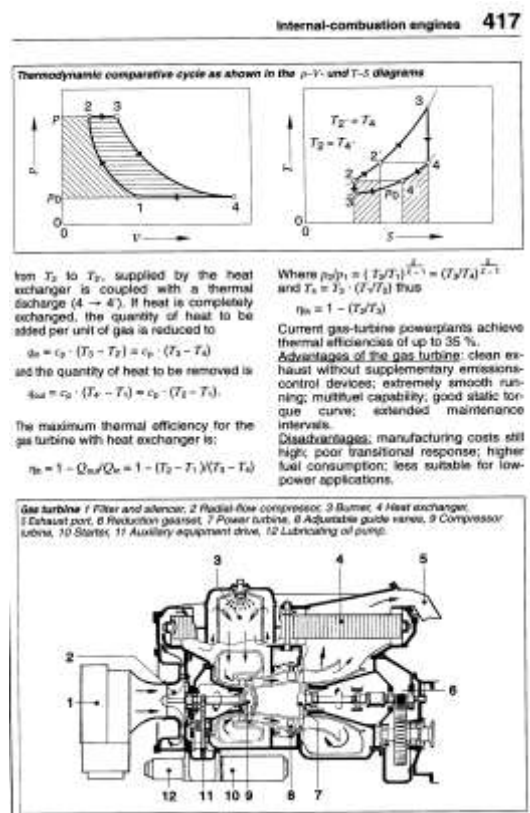
Part I

Introduction to Modelica and a demo example



Modelica Background: Stored Knowledge

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



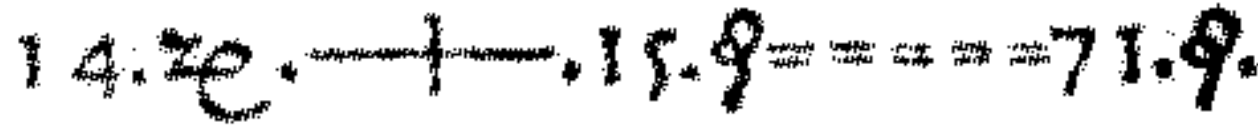
“The change of motion is proportional to the motive force impressed”
– Newton

Lex. II.

Mutationem motus proportionalem esse vi motrici impressae, & 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

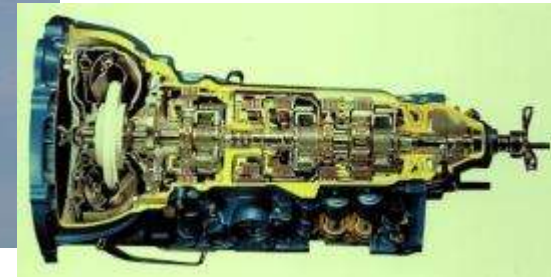
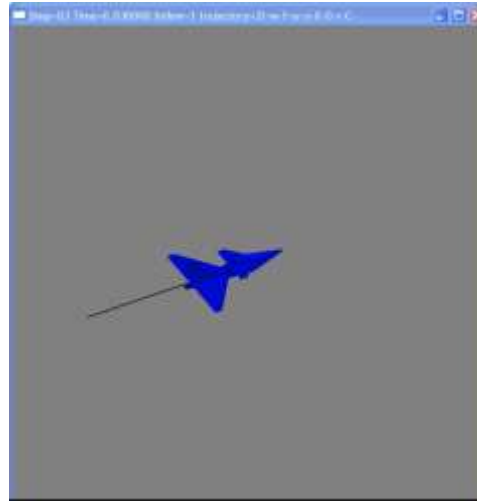
$v = \text{INTEG}(F) / m$

Programming languages usually do not allow equations!

What is Modelica?

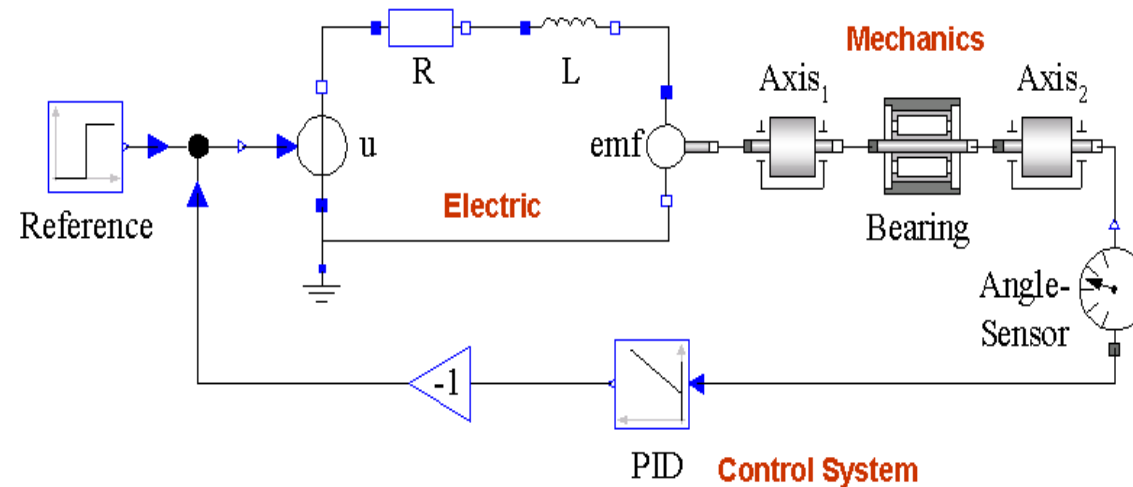
A language for modeling of **complex cyber-physical systems**

- 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:



Available at: www.modelica.org

*Developed and standardized
by Modelica Association*

**There exist one free and several commercial tools,
for example:**

- **OpenModelica from OSMC**
(also in ABB Optimax, Bosch-Rexroth 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 Simplorer, Rickardo tool, etc.)
- MWORKS from Tongyang Sw & Control
- IDA Simulation Env, from Equa

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*:

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

can be used in three ways:

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

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

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

What is Special about Modelica?

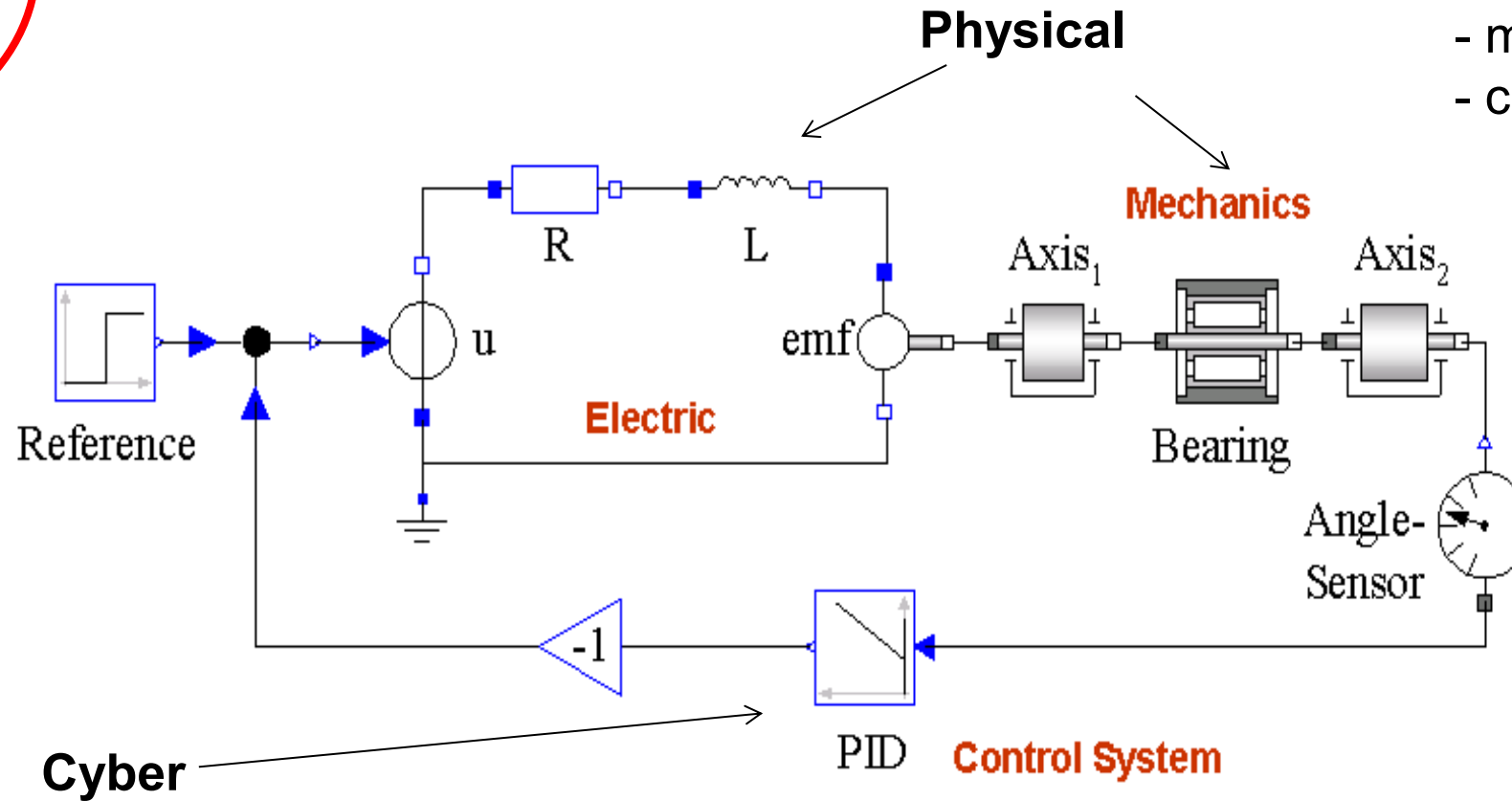
- Multi-Domain Modeling
- Visual acausal hierarchical component modeling
- Typed declarative equation-based textual language
- Hybrid modeling and simulation

What is Special about Modelica?

Multi-Domain
Modeling

Cyber-Physical Modeling

3 domains
- electric
- mechanics
- control

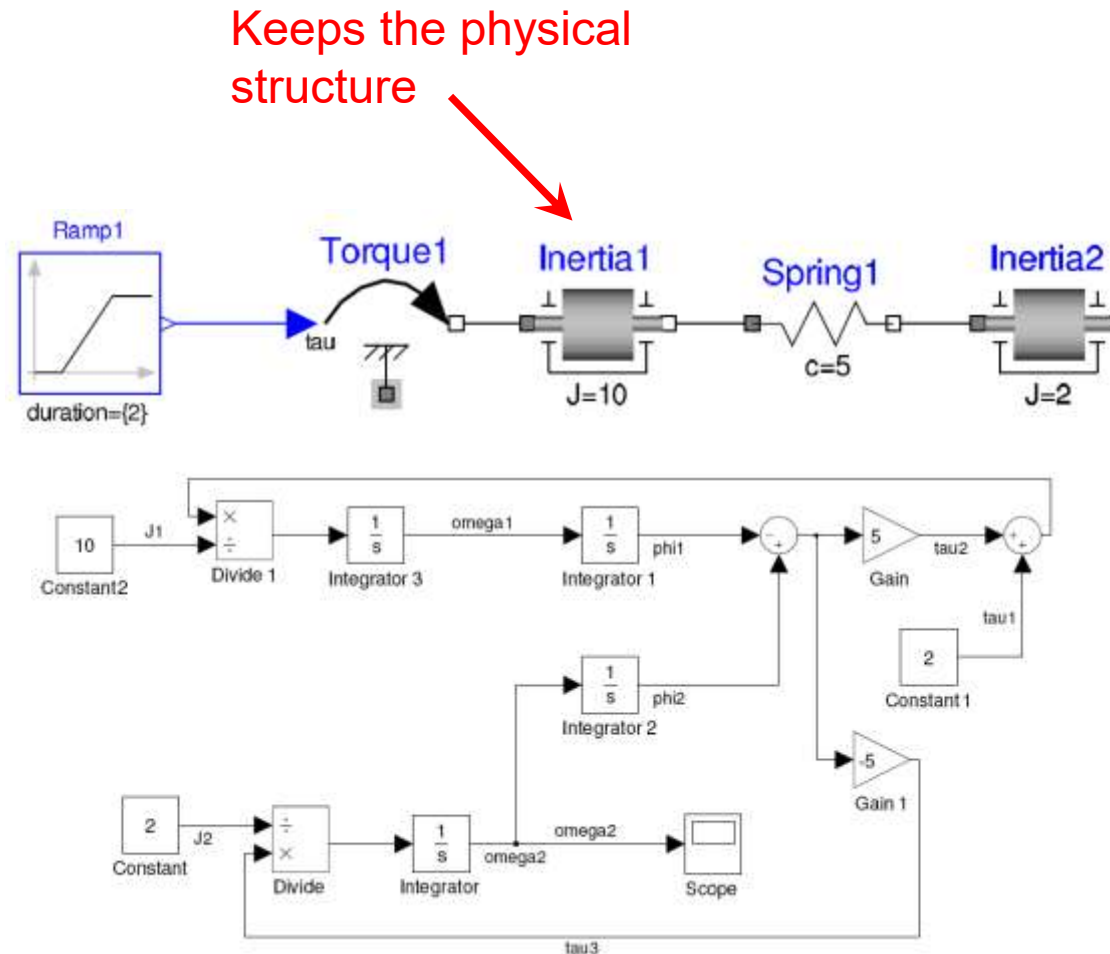


What is Special about Modelica?

Multi-Domain
Modeling

Acausal model
(Modelica)

Causal
block-based
model
(Simulink)



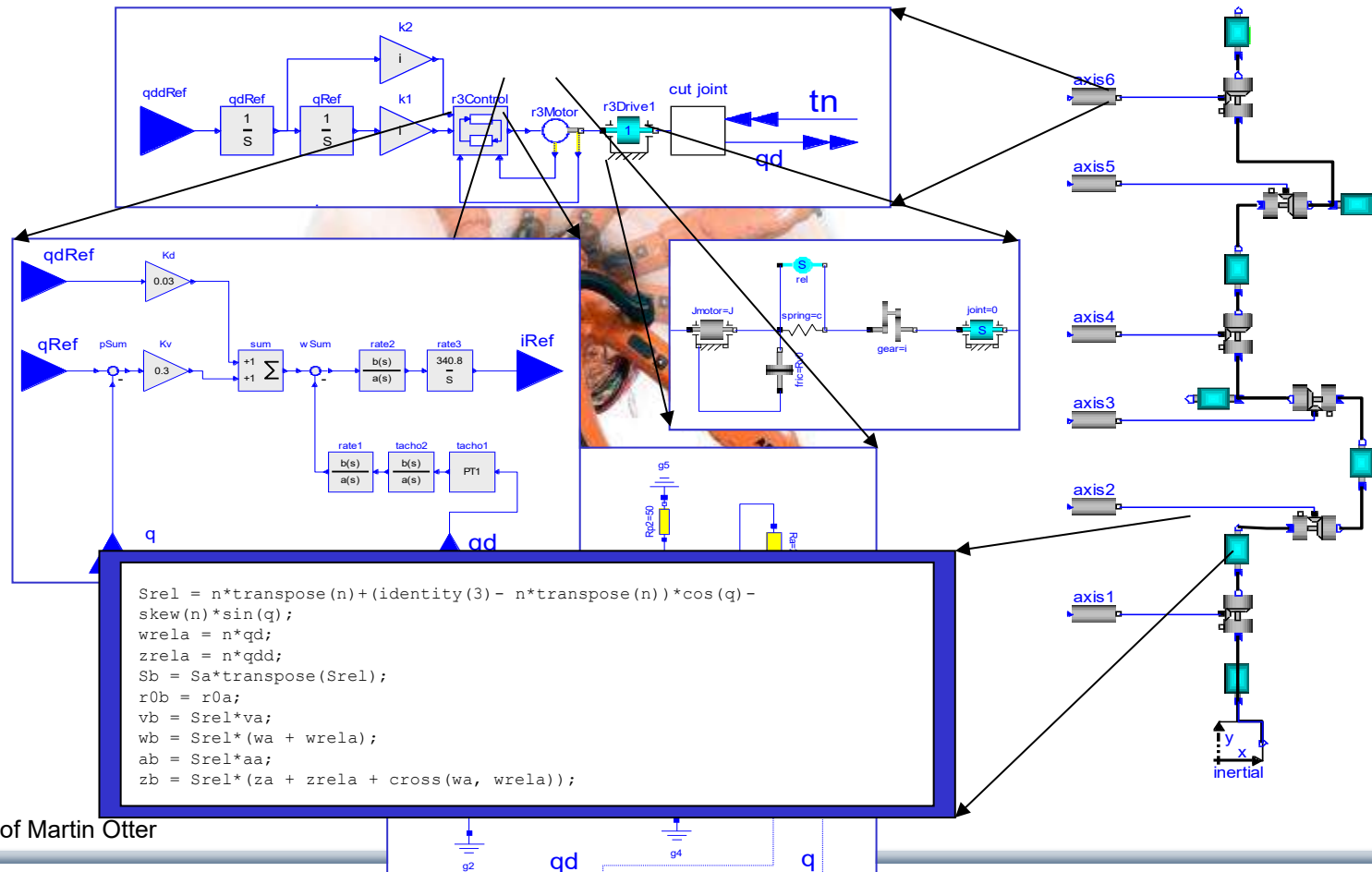
Visual Acausal
Hierarchical
Component
Modeling

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

Visual Acausal
Hierarchical
Component
Modeling

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

Behavior described declaratively using

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

Variable
declarations

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

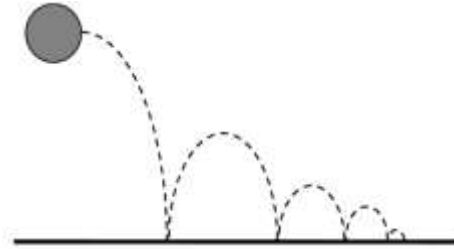
Differential equations

Typed
Declarative
Equation-based
Textual Language

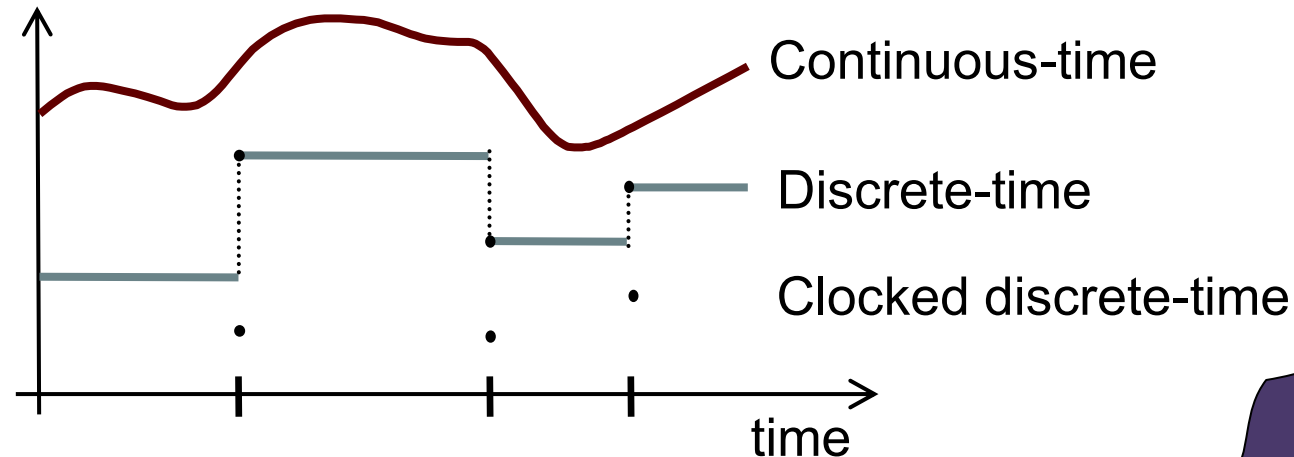
What is Special about Modelica?

Multi-Domain
Modeling

Visual Acausal
Component
Modeling



Hybrid modeling =
continuous-time + discrete-time modeling

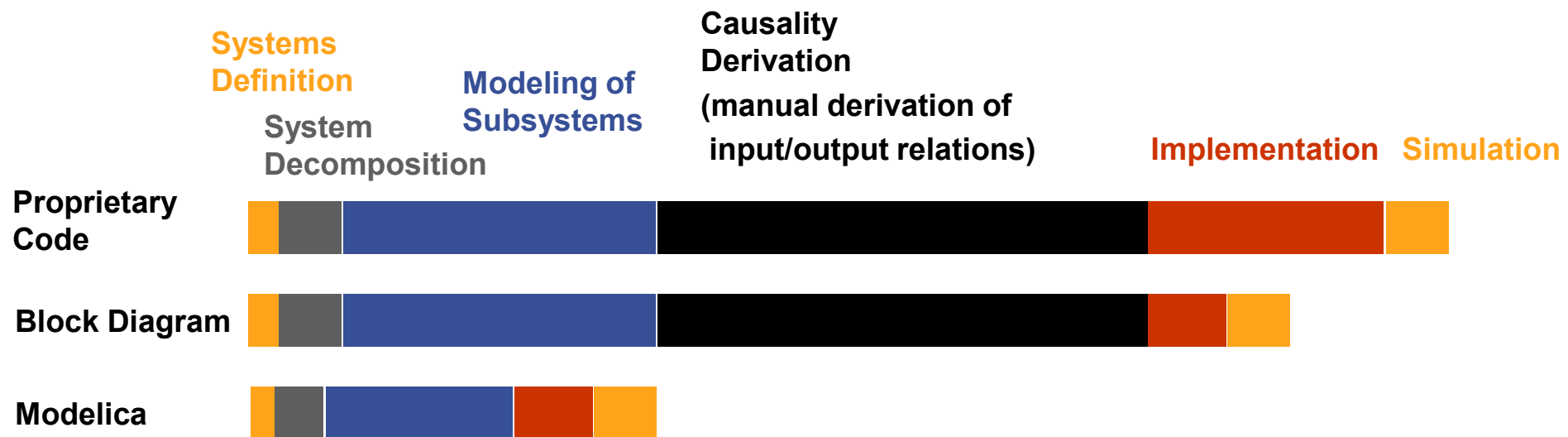


Typed
Declarative
Equation-based
Textual Language

Hybrid
Modeling

Modelica – Faster Development, Lower Maintenance than with Traditional Tools

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

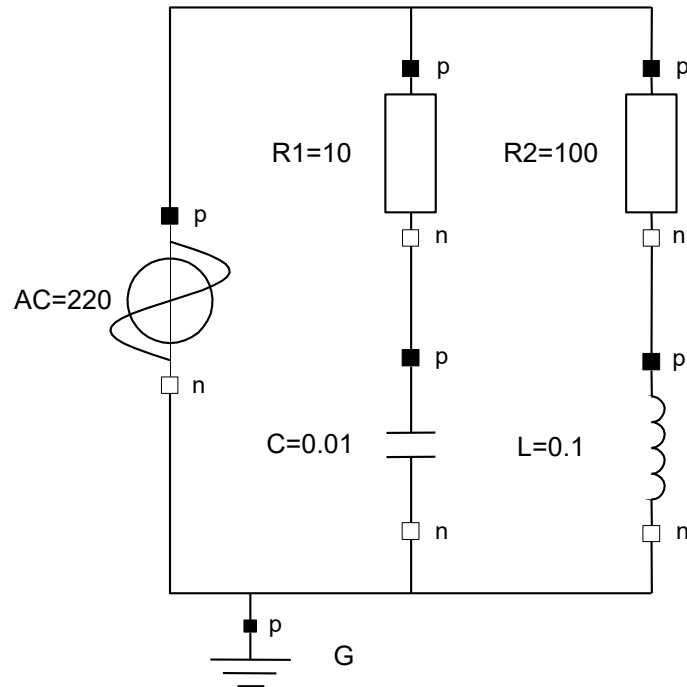


Modelica vs Simulink Block Oriented Modeling

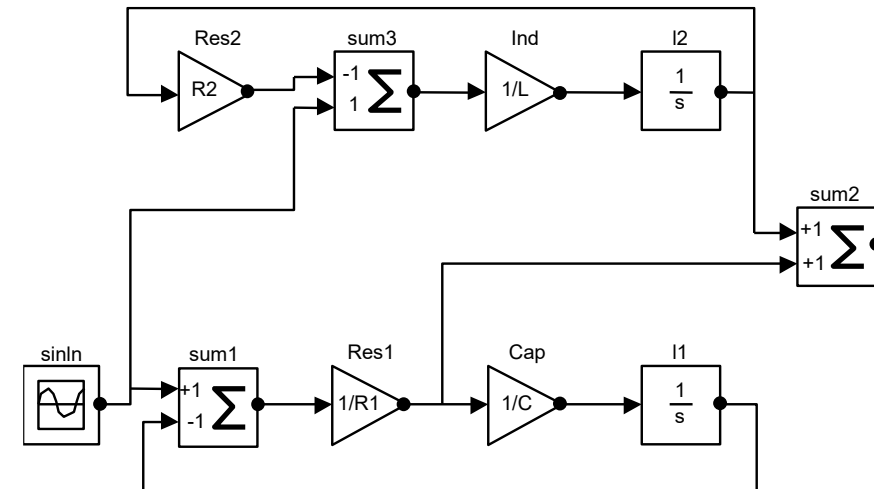
Simple Electrical Model

Modelica:
Physical model –
easy to understand

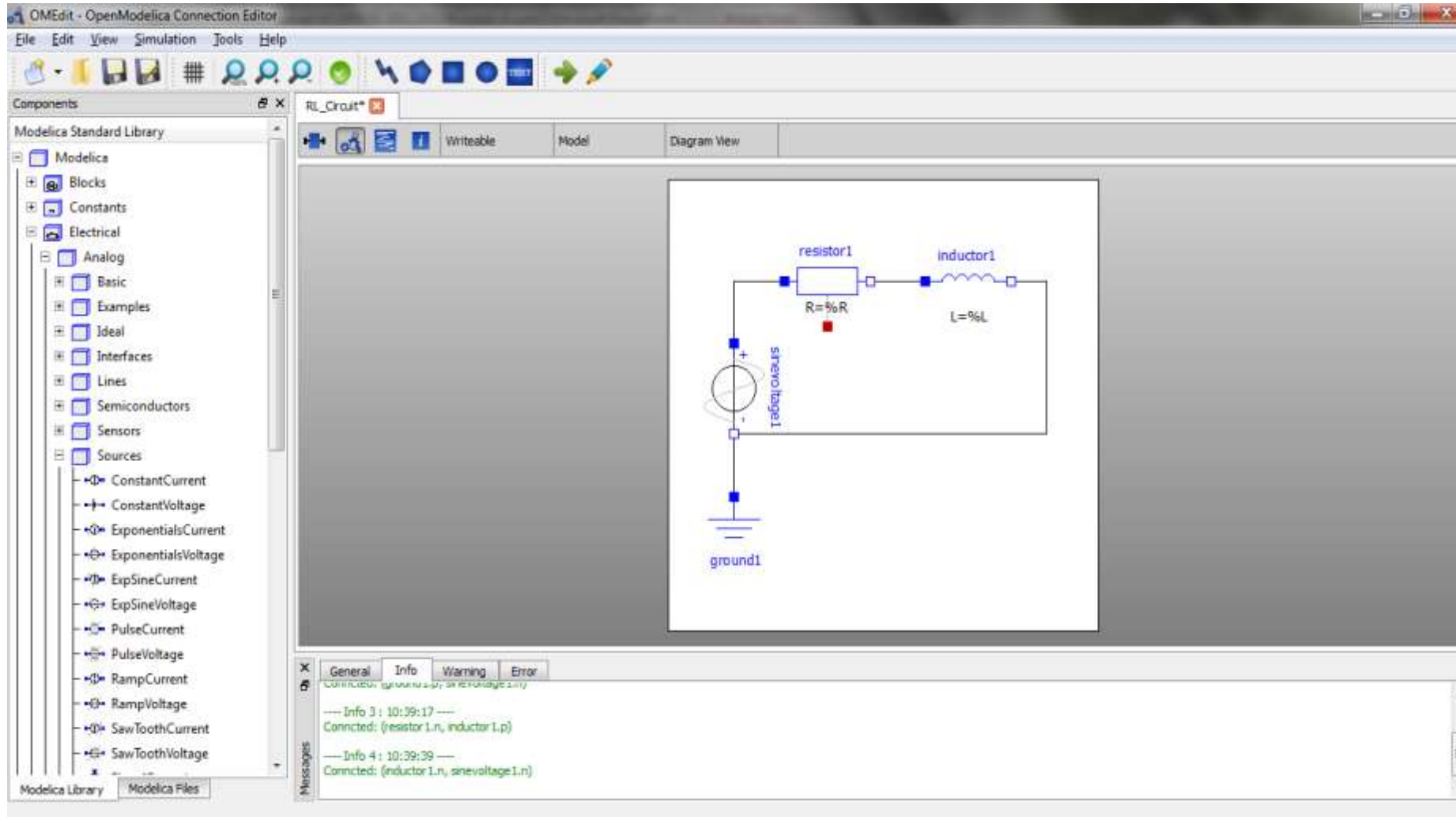
Keeps the
physical
structure



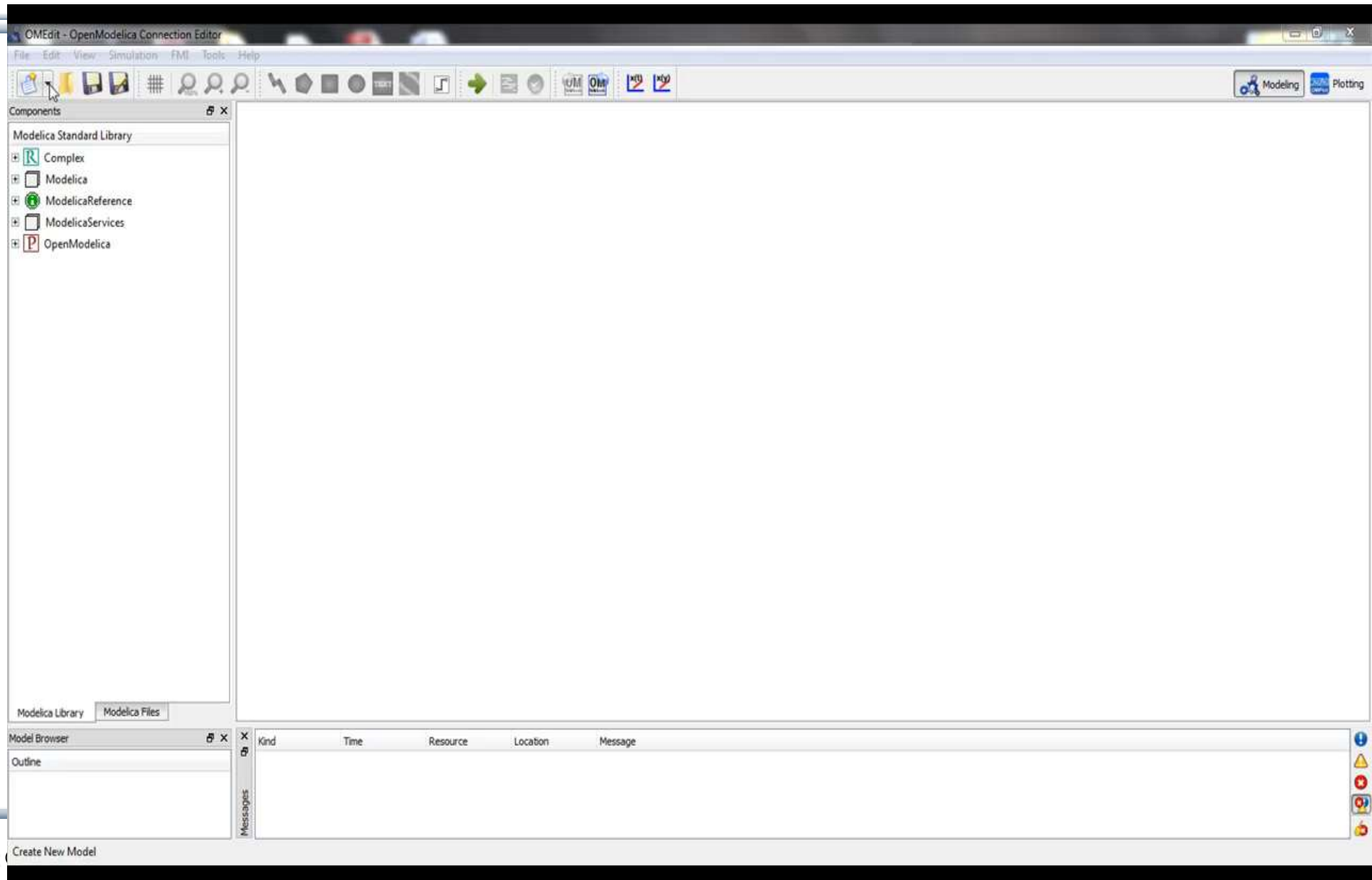
Simulink:
Signal-flow model – hard to
understand



Graphical Modeling - Using Drag and Drop Composition



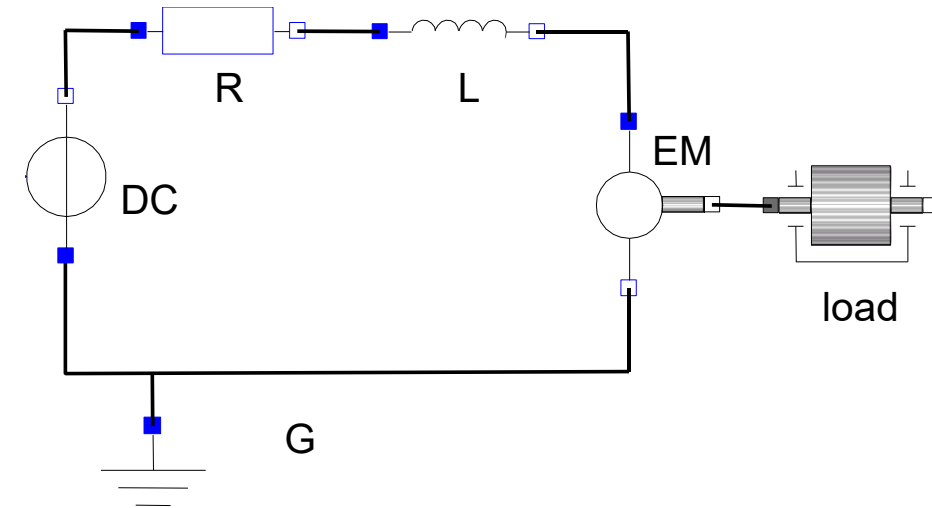
Graphical Modeling with OpenModelica Environment



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:

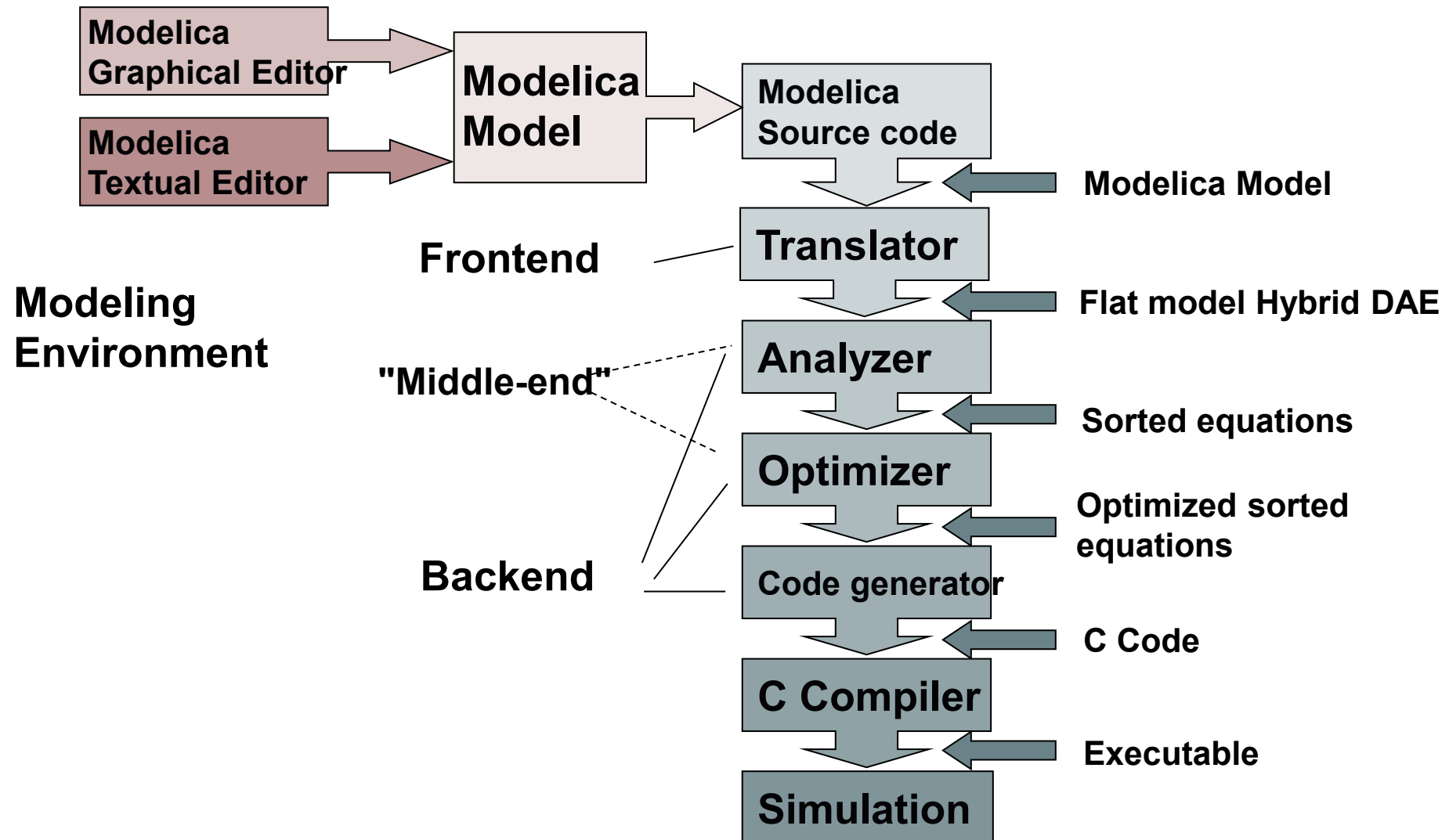
0 == DC.p.i + R.n.i	EM.u == EM.p.v - EM.n.v	R.u == R.p.v - R.n.v
DC.p.v == R.n.v	0 == EM.p.i + EM.n.i	0 == R.p.i + R.n.i
	EM.i == EM.p.i	R.i == R.p.i
0 == R.p.i + L.n.i	EM.u == EM.k * EM.ω	R.u == R.R * R.i
R.p.v == L.n.v	EM.i == EM.M / EM.k	
	EM.J * EM.ω == EM.M - EM.b * EM.ω	L.u == L.p.v - L.n.v
0 == 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:

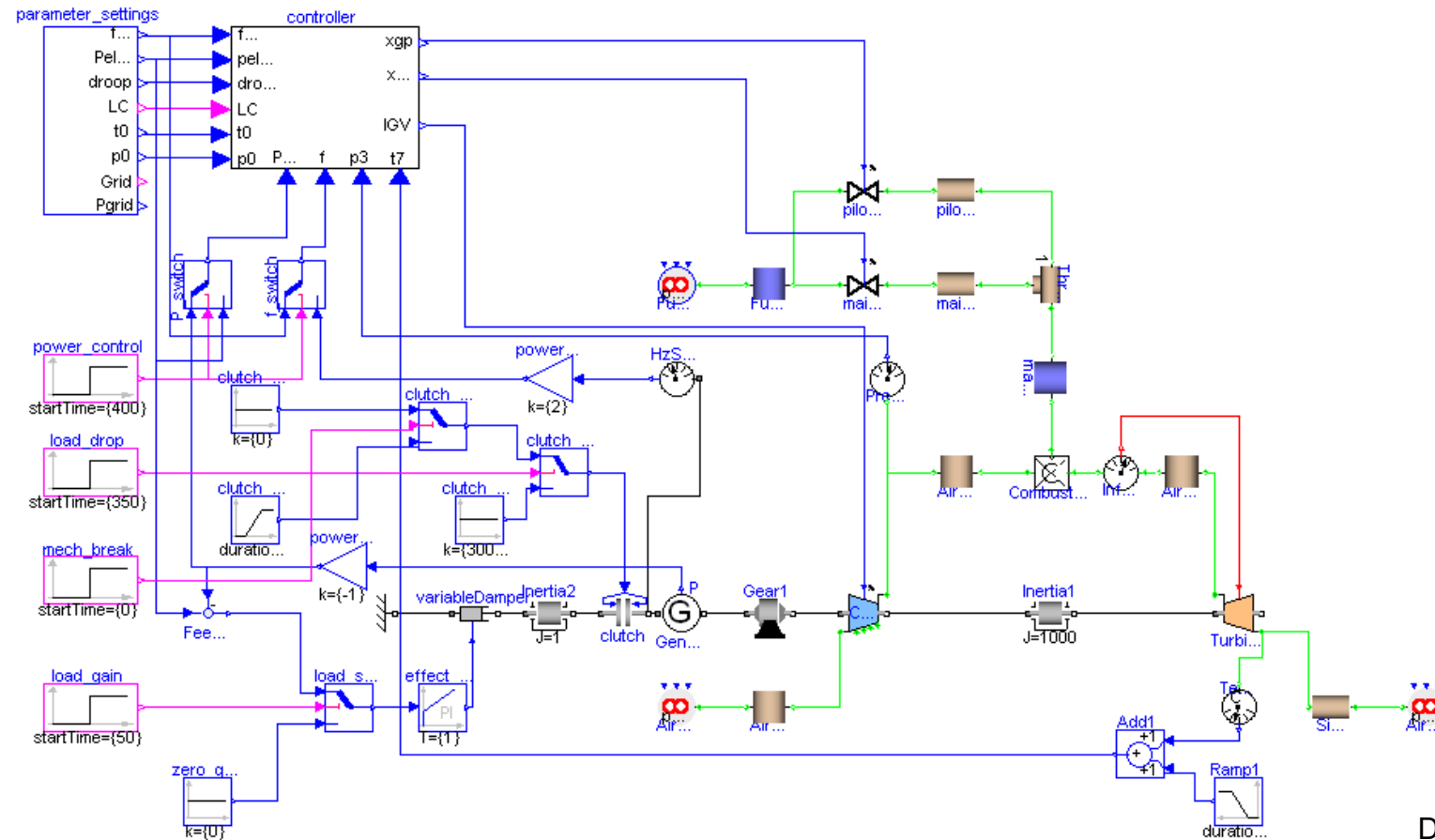
$$\frac{dx}{dt} == f[x, u, t] \quad g\left[\frac{dx}{dt}, x, u, t\right] == 0$$

Model Translation Process to Hybrid DAE to Code



Modelica in Power Generation

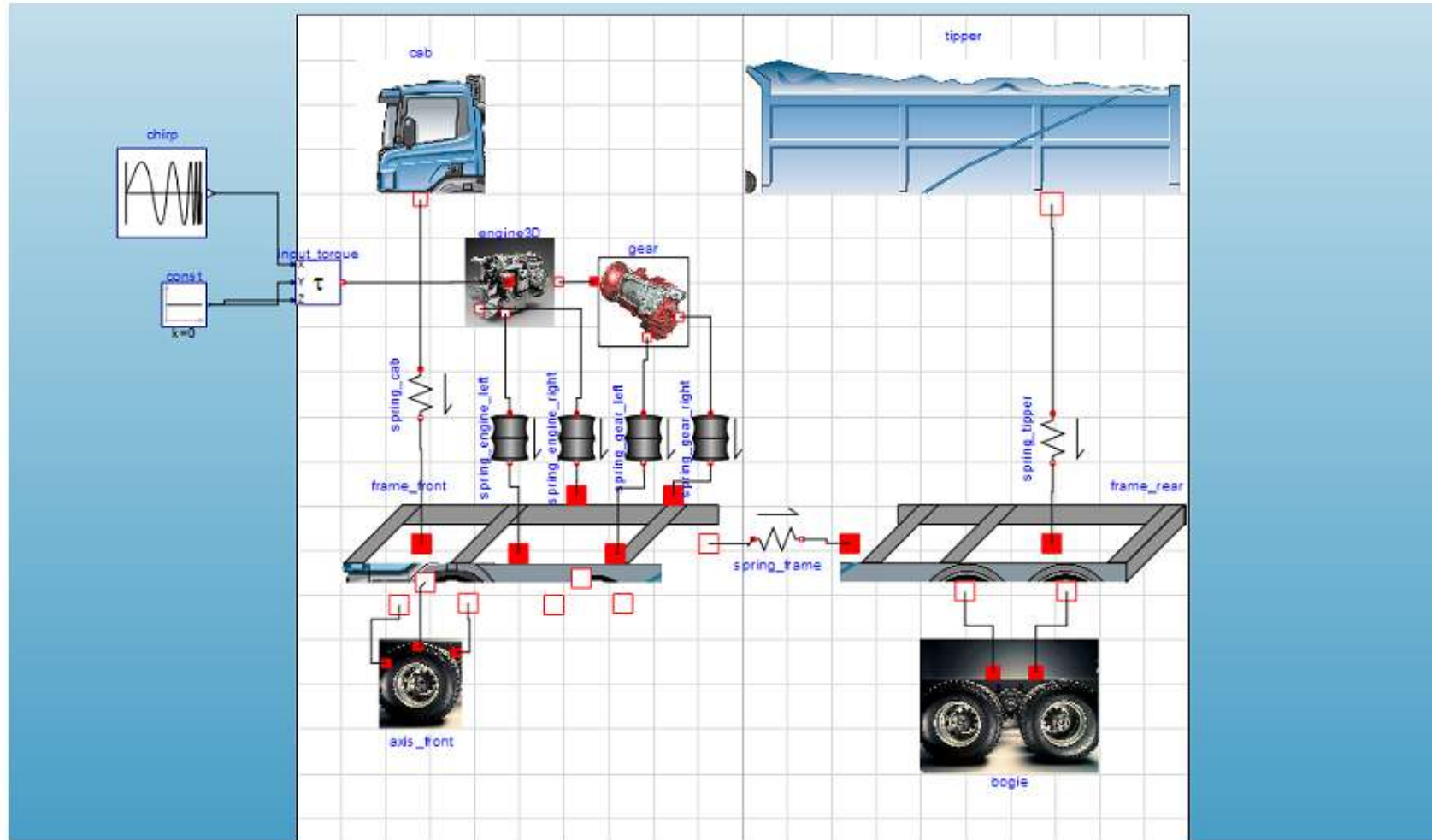
GTX Gas Turbine Power Cutoff Mechanism



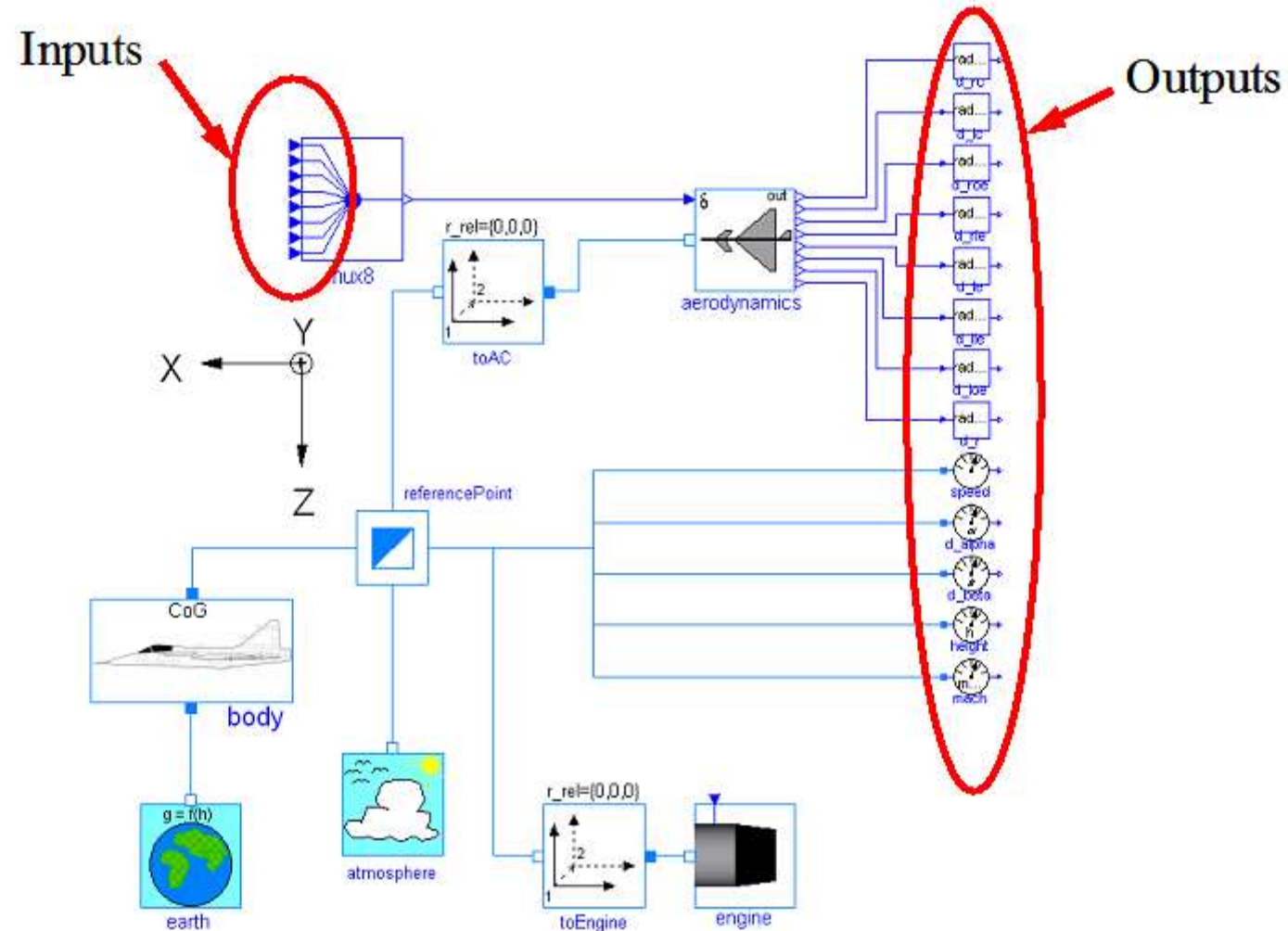
Developed
by MathCore
for Siemens

Courtesy of Siemens Industrial Turbomachinery AB

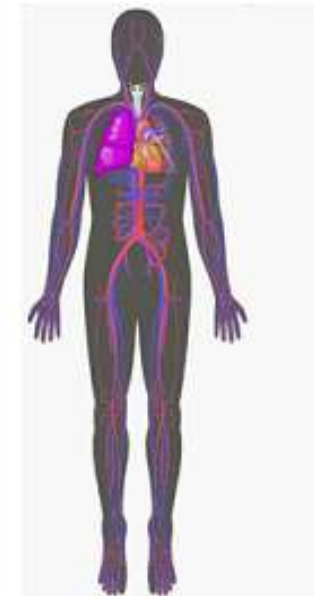
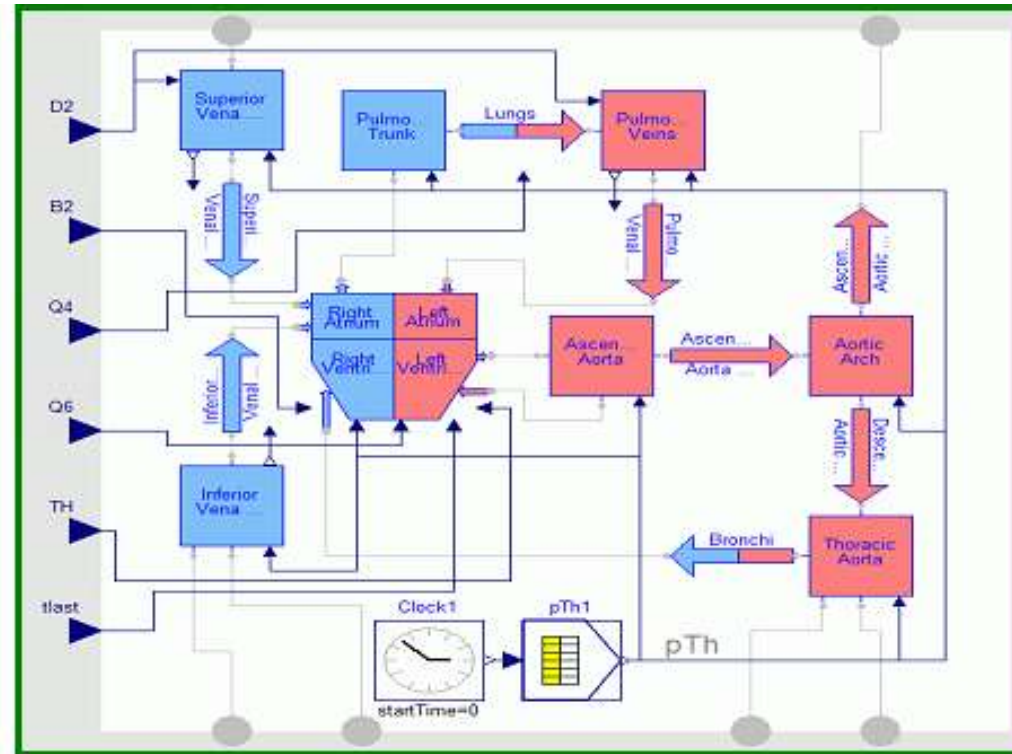
Modelica in Automotive Industry



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

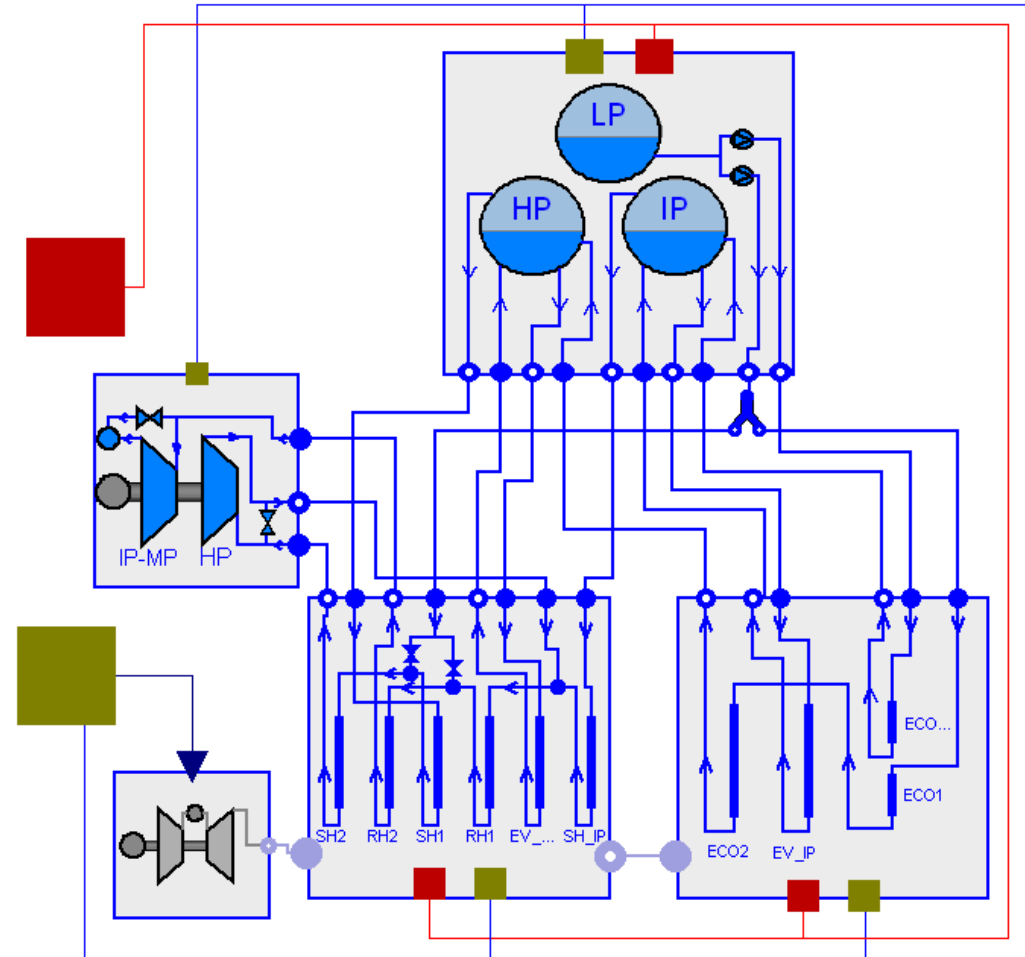


Combined-Cycle Power Plant

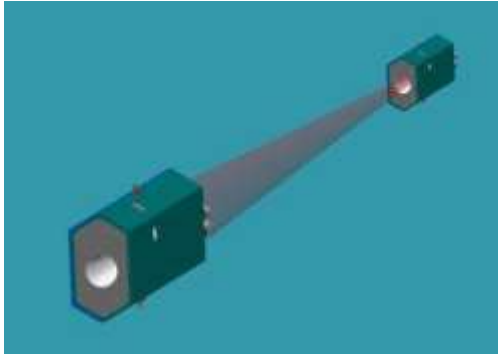
Plant model – system level

- 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

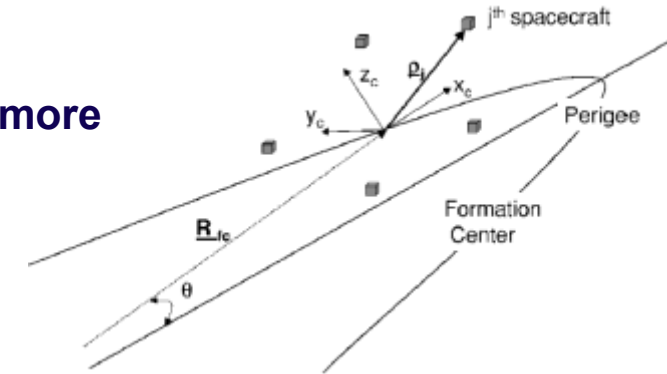
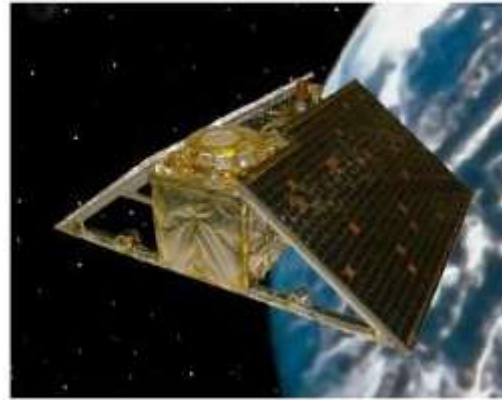


Modelica Spacecraft Dynamics Library



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

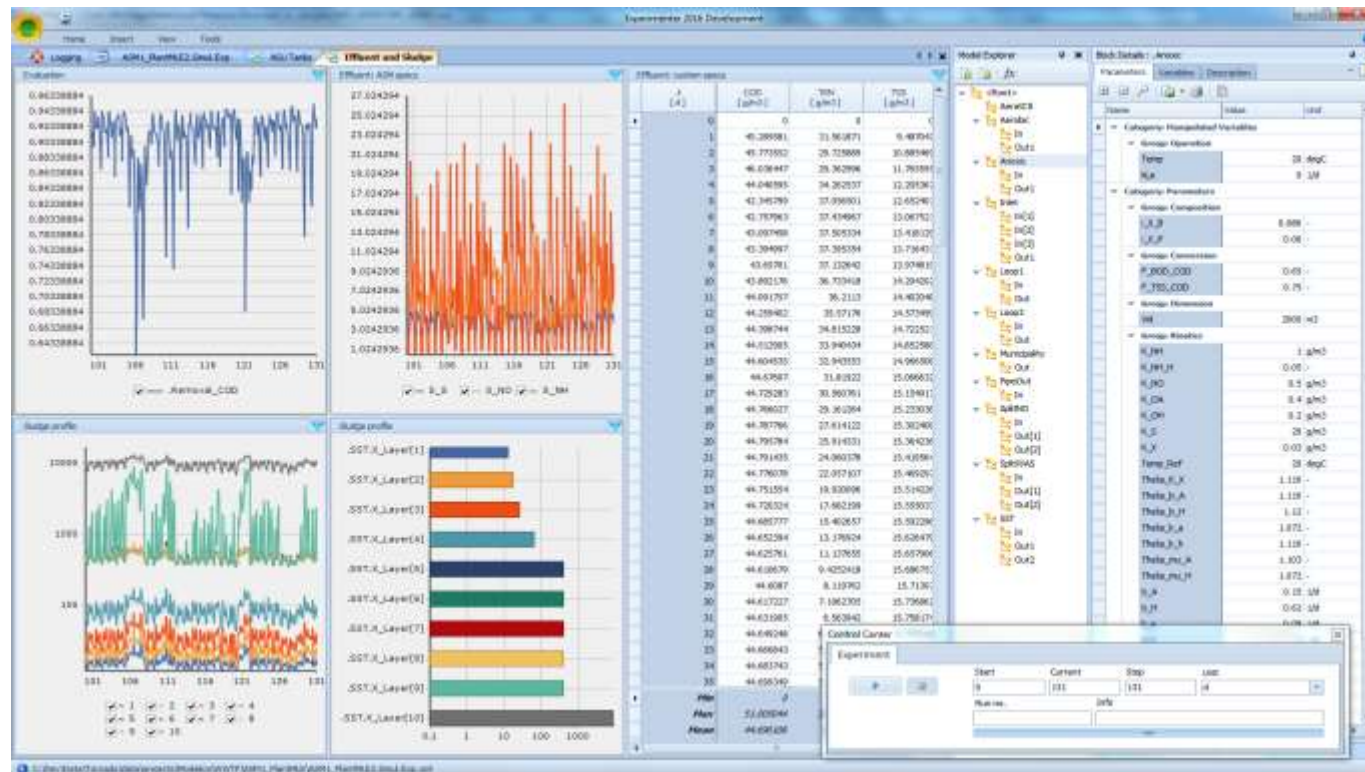
- 2012: initial delivery (for 50 plants)
- 2013: SW extension (500 plants)
- 2014: HW+SW extension (> 2000)
- 2015: HW+SW extension, incl. OpenModelica generating optimizing controller code in FMI 2.0 form

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

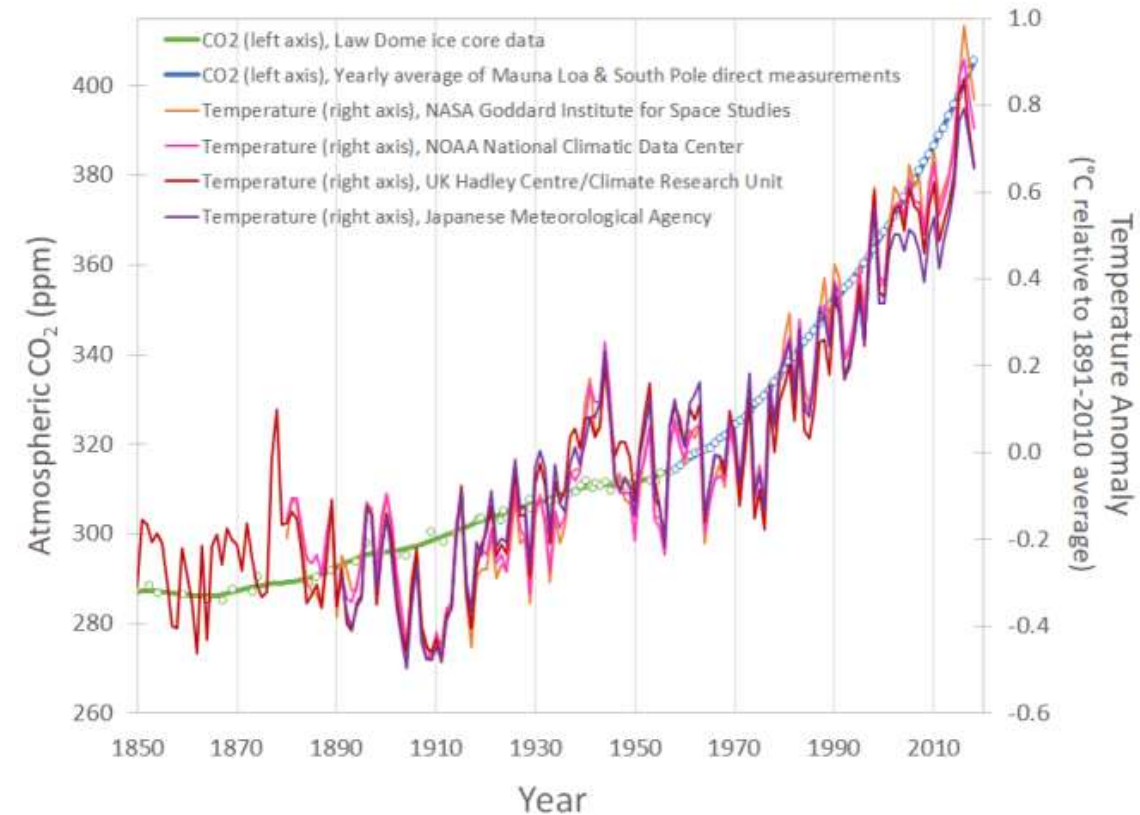
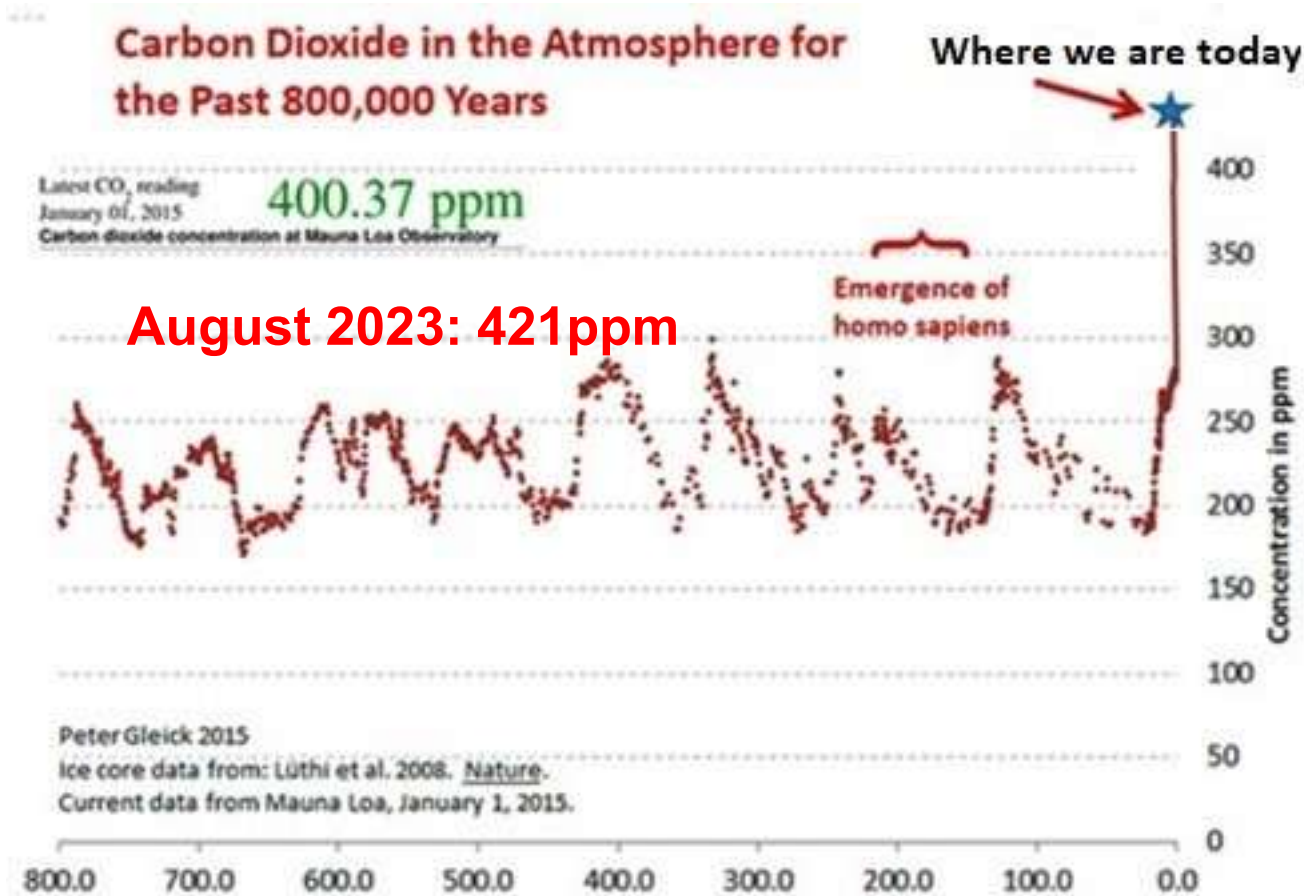
- **MIKE by DHI**, www.mikebydhi.com, **WEST Water Quality** modeling and simulation environment
- Includes a large part of the OpenModelica compiler using the OEM license.
- Here a water treatment effluent and sludge simulation.



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

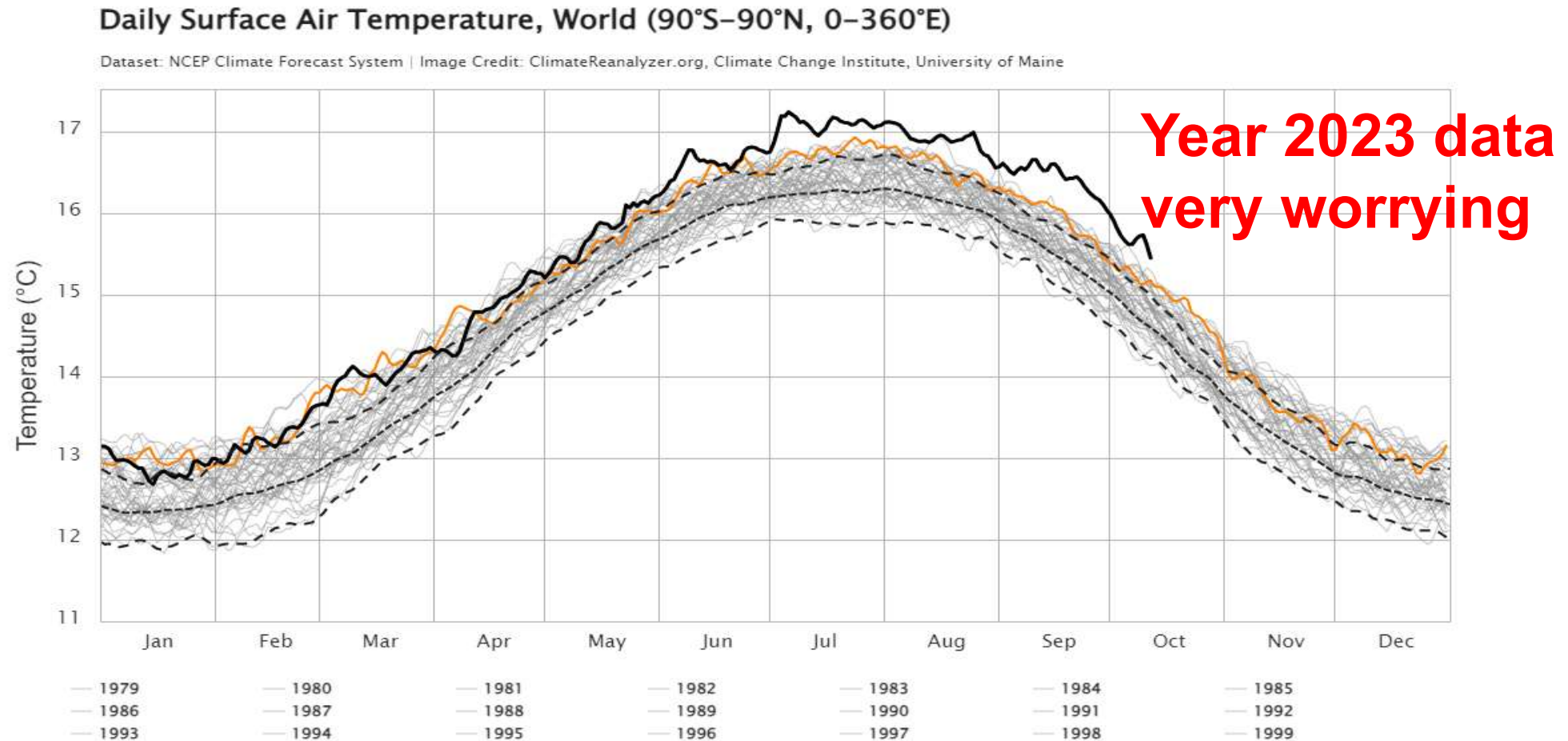
**Use Modelica in to model and optimize
sustainable technical innovations,
and a sustainable circular economy**

Carbon dioxide in the atmosphere and earth temperature



There is no doubt that man is affecting the climate

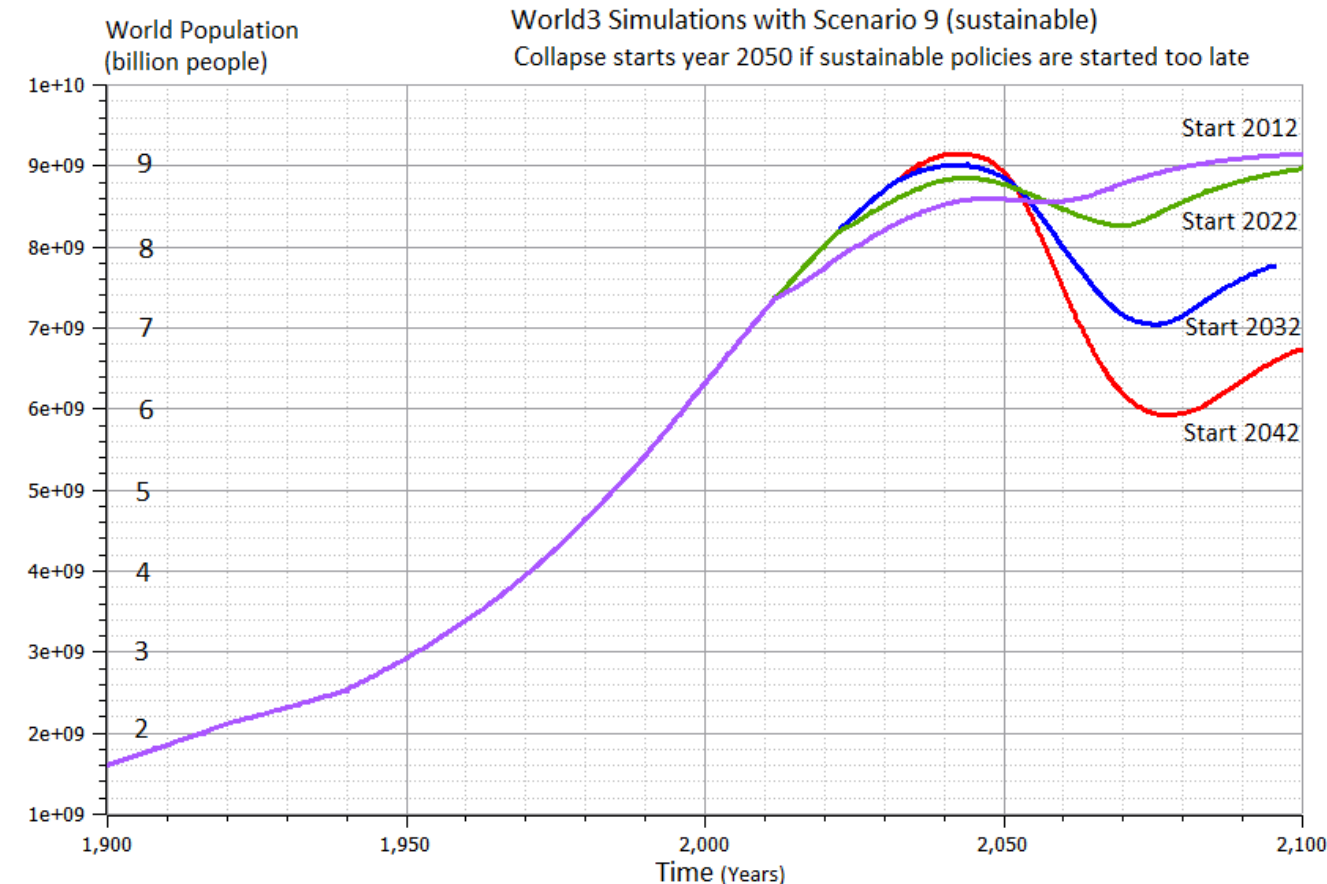
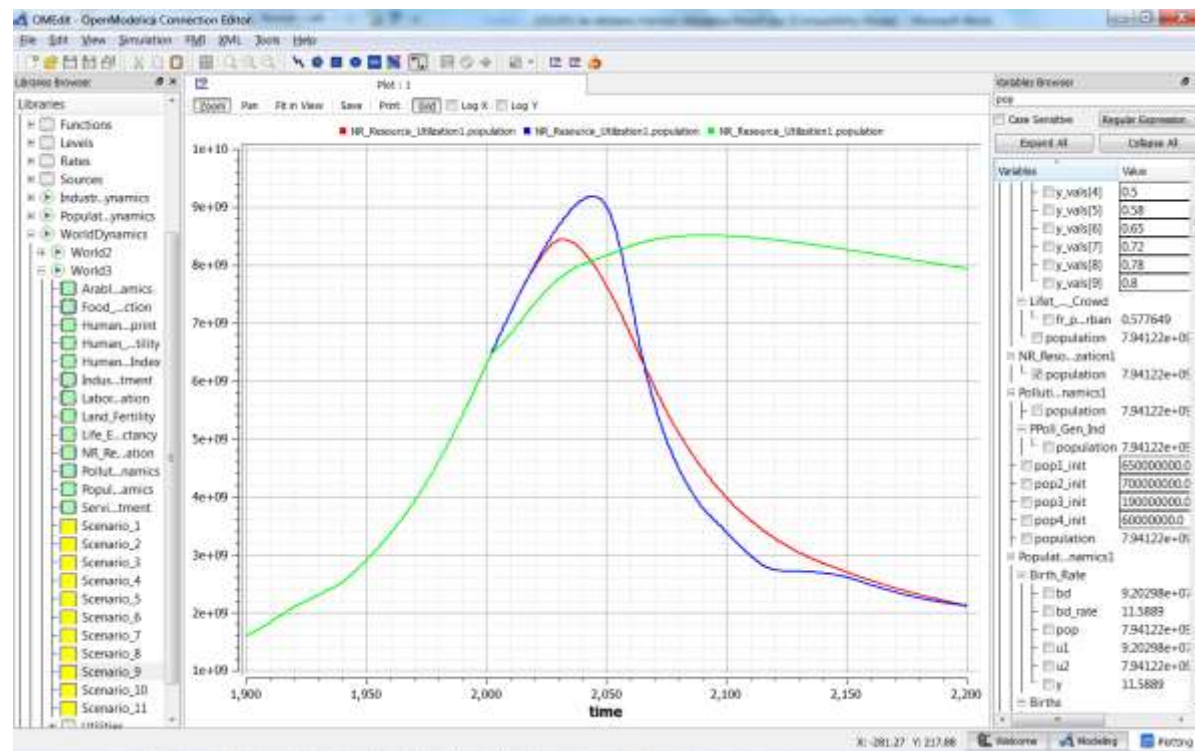
Carbon dioxide in the atmosphere and earth temperature



September 2023 shows 1.75 °C higher temperature than preindustrial.
climate.copernicus.eu

World3 Simulations with Different Start Years for Sustainable Policies

– World Society Collapse if starting too late



Left. System Dynamics World3 simulation with OpenModelica. World population. (ref Meadows et al)

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

- 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

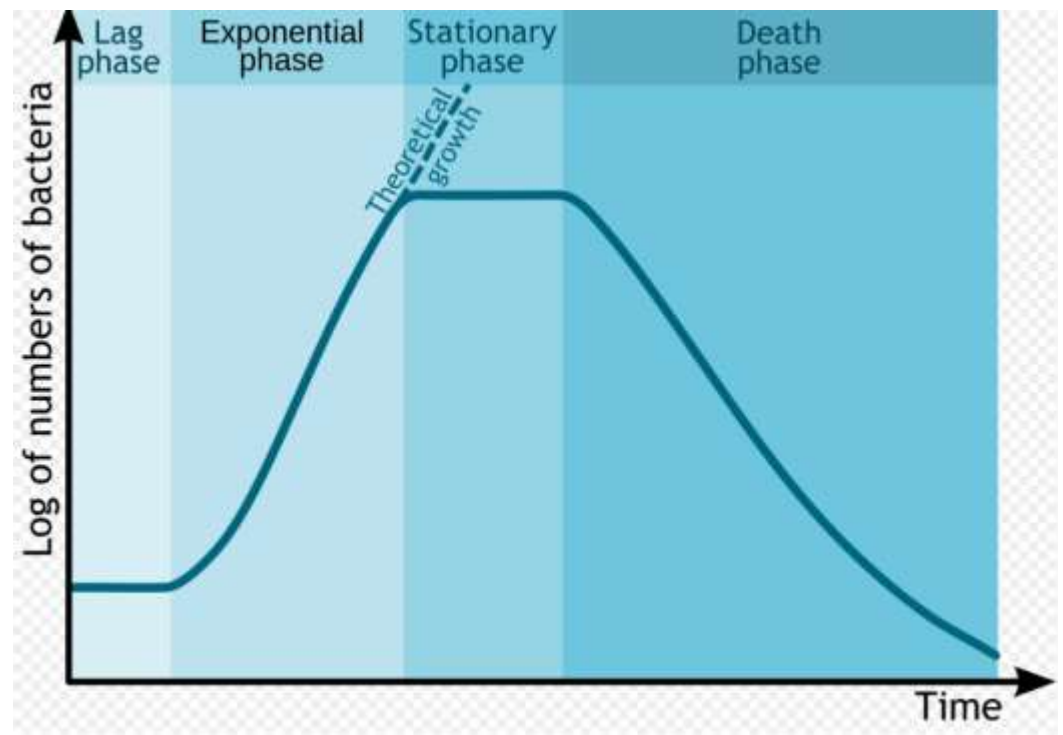
Are Humans More Intelligent than Bacteria?

Not yet evident!

Humans
on a
finite
Earth

vs

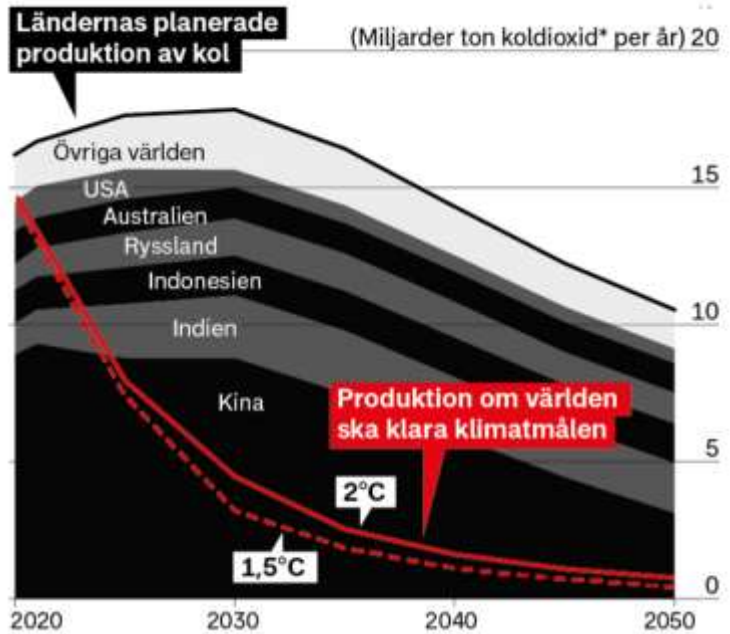
Bacteria
on a
finite
substrate



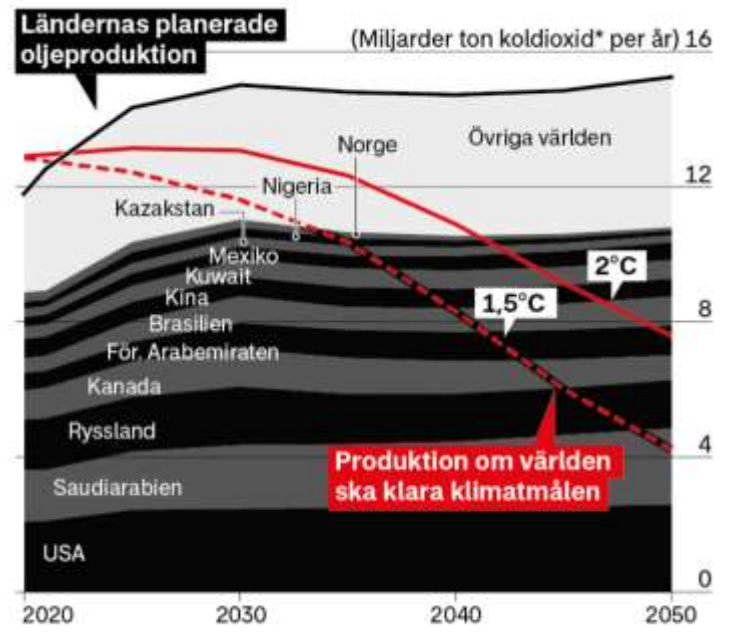
Bacterial growth curve /kinetic curve (Wikipedia)

ProductionGap.org

Coal



Oil



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

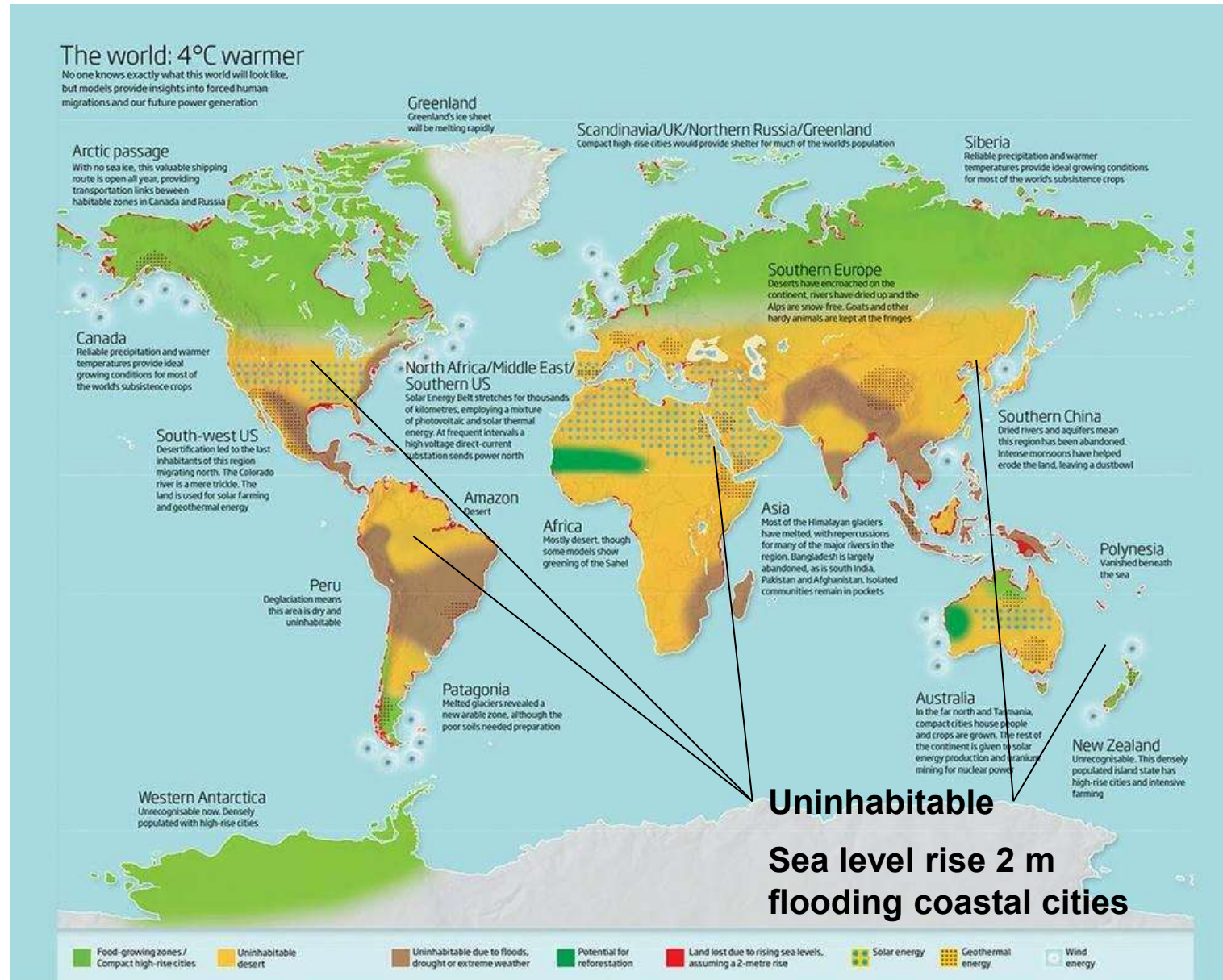
WITH A NEW AFTERWORD



MODELICA

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

Business-as-usual scenario, IPCC, 2009



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

Massive migration to northern Europe, Russia, and Canada

Example Emissions
CO₂e / person

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

References

New Scientist, 28 February 2009
IPCC, business as usual scenario
www.climate-lab-book.ac.uk
www.atmosfair.de

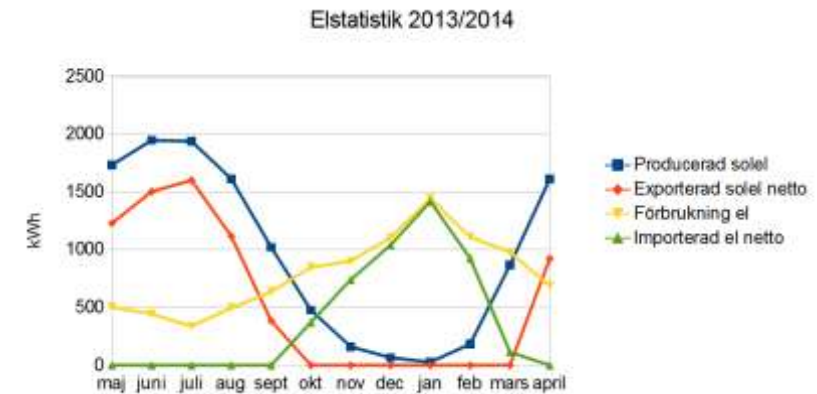
What Can You Do?

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

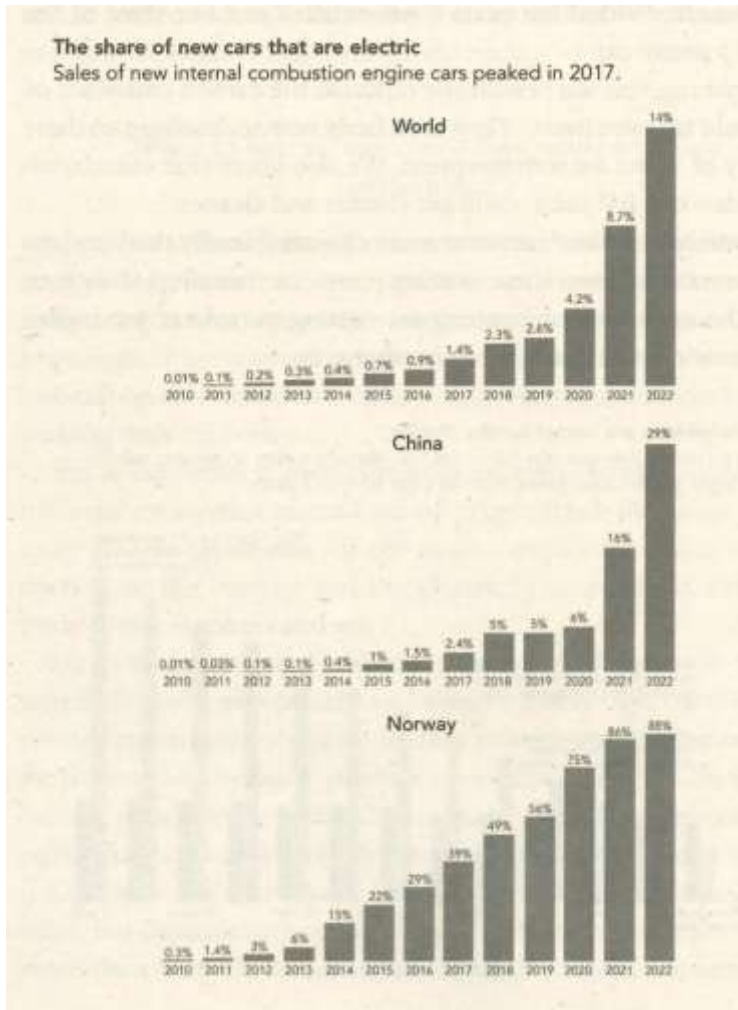


Expanded to 93 sqm, 12 kW, March 2013
House produced 11600 kwh, used 9500 kwh
Avoids 10 ton CO2 emission per year

Example Electric Cars

Can be charged by electricity from own solar panels

Share of new electric cars: world, China, Norway 2010-2022



**Small electric car Renault ZOE;
5 seat; Range:**

- **Realistic Swedish drive cycle:**
- **Summer: 2019 385 km**
- **Winter: 2019 290 km**



2020, Volvo XC40 recharge, range 400 km



2025 Volkswagen ID4 GTX, range up to 515 km

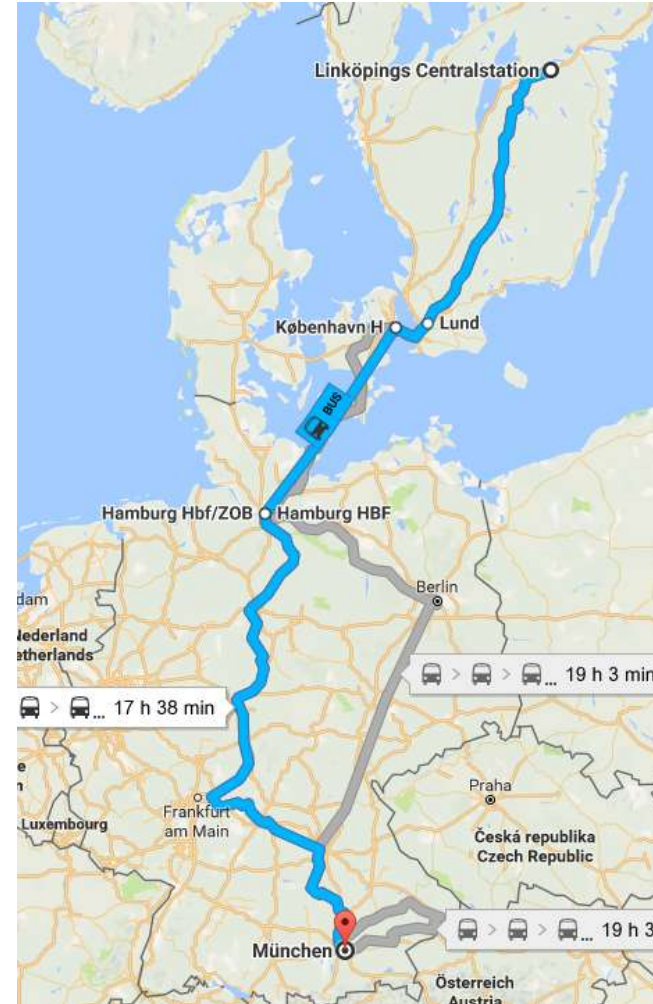
What Can You Do?

More Train or Bus Travel – Less Air Travel

- Air travel by Swedish Citizens – about the same emissions as all personal car traffic in Sweden, about 10 Mtons/year!
- By train from Linköping to Munich and back – saves almost 1 ton of CO2e emissions compared to flight
- Leave Linköping 07.00 in Munich 23.14

More Examples, PF travel:

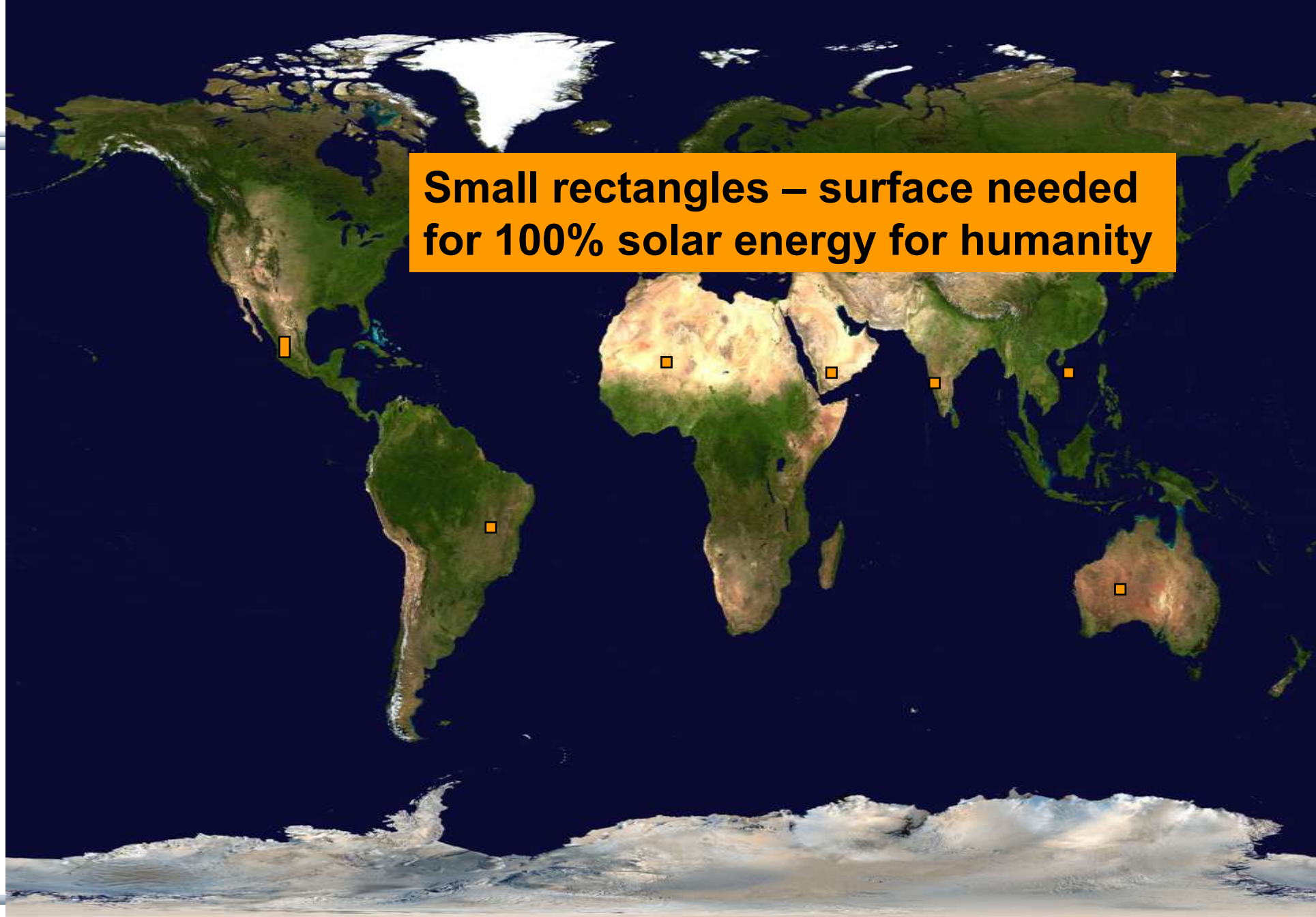
- Train Linköping-Paris, Dec 3-6, 2016, EU project meeting
- Train Linköping-Aachen, Oct 8-12, 2023 Modelica Conference



Train travel
Linköping -
Munich

Bus trip 2024
Sweden – Italy
- Sweden
4000 km, **56
kg** emissions

Compare:
Flight to Italy
almost
1 ton
emissions

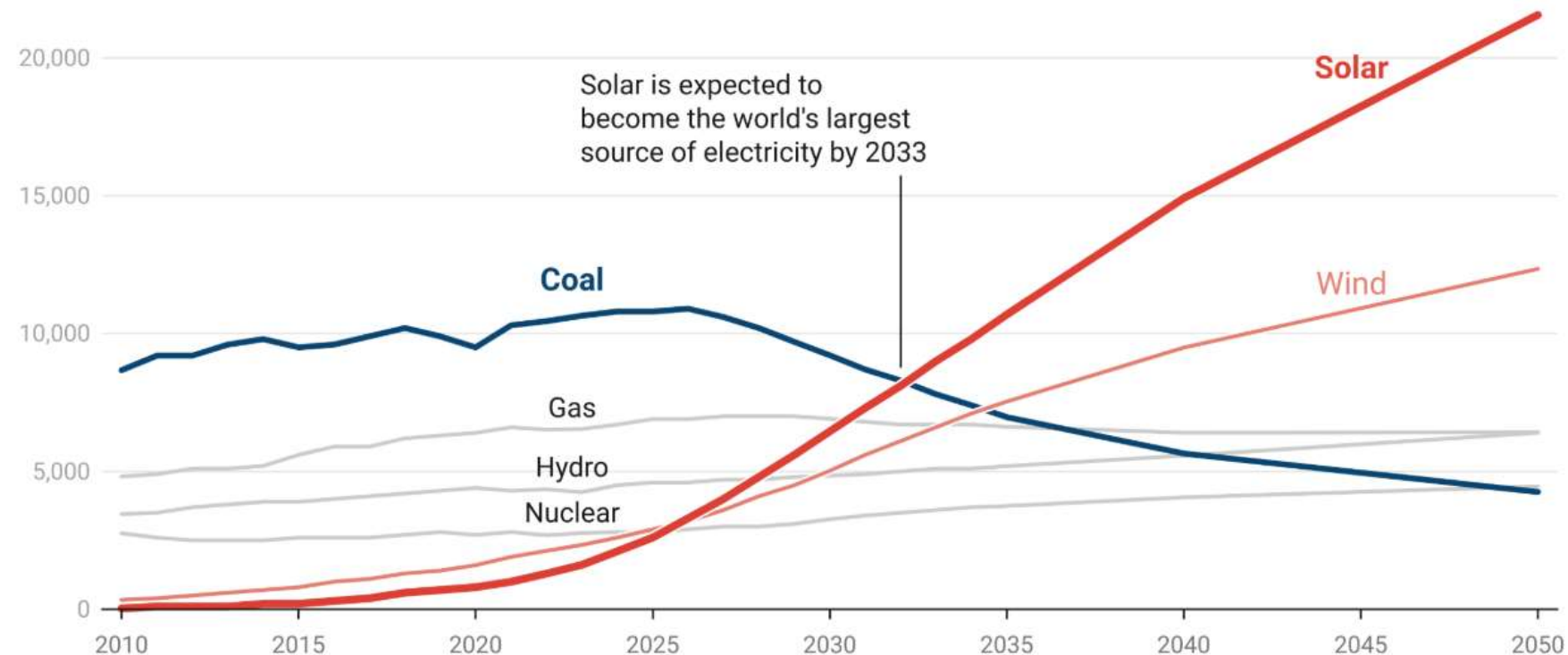


**Small rectangles – surface needed
for 100% solar energy for humanity**

Solar Energy (electricity) will increase 4 times to 2030 biggest electricity producer 2033, coal,oil,gas decrease

Solar generation is set to quadruple by 2030, sending coal power tumbling

Global electricity generation by source, TWh



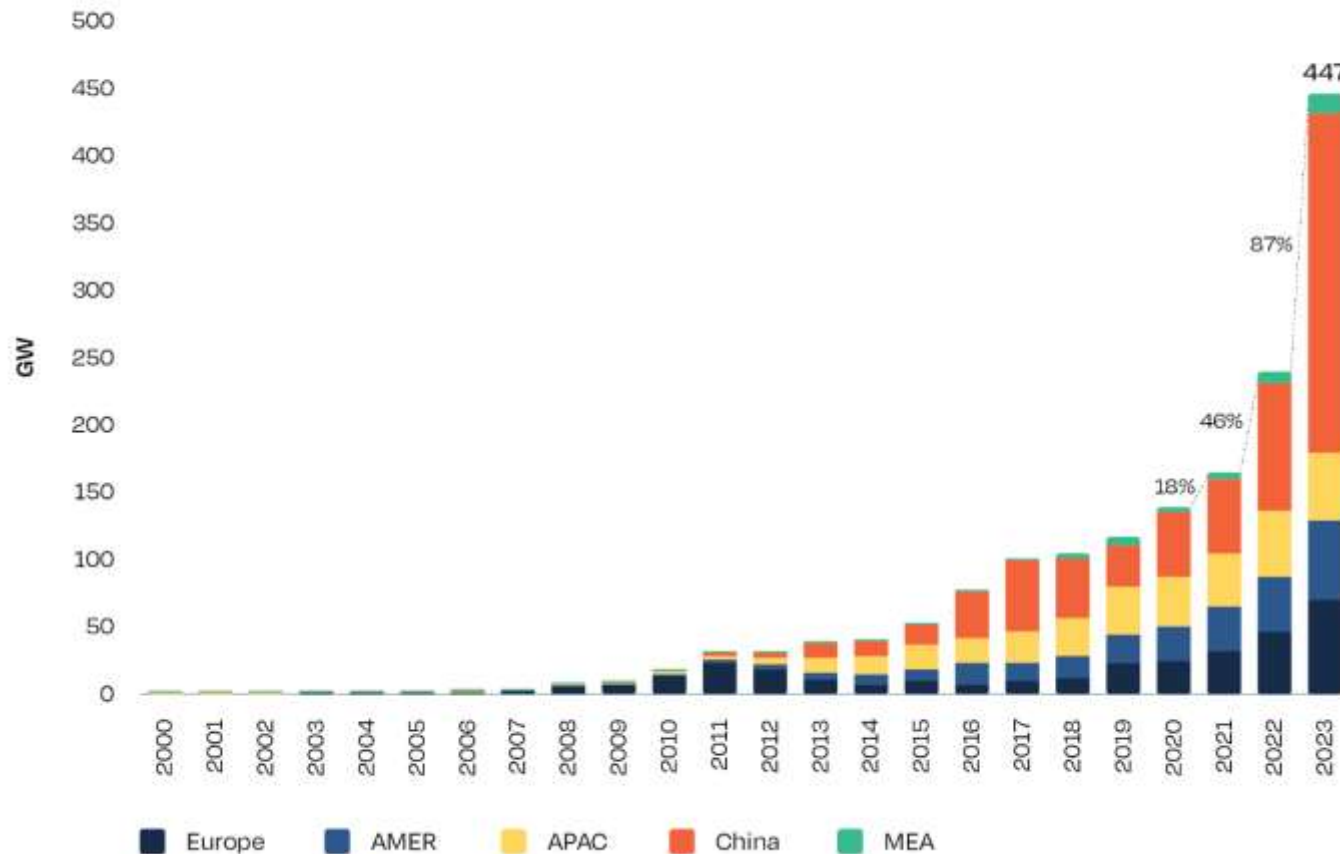
Source: World Energy Outlook 2024

CarbonBrief
CLEAR ON CLIMATE

Global electricity generation by source, TWh, 2010-2050. Source: World Energy Outlook 2024.

Solar, Installed Global Production from 2000 to 2023

Installed solar production per year (photo voltaics) 2000 to 2023



Renewable energy, Increased
50% year 2023, 75% is solar

**China is leading: Installed
solar 2023 same as the
whole world 2022.**

Increased wind power +66%

**Solar Boom! China installed
solar power corresponding
to 200 nuclear power
stations first half of 2025.**

Source: Global Market Outlook for Solar
Power 2024-2028. Solar Power Europe.
MEA: Middle East and Africa
APAC: Asia and Pacific (not China)
AMER: America

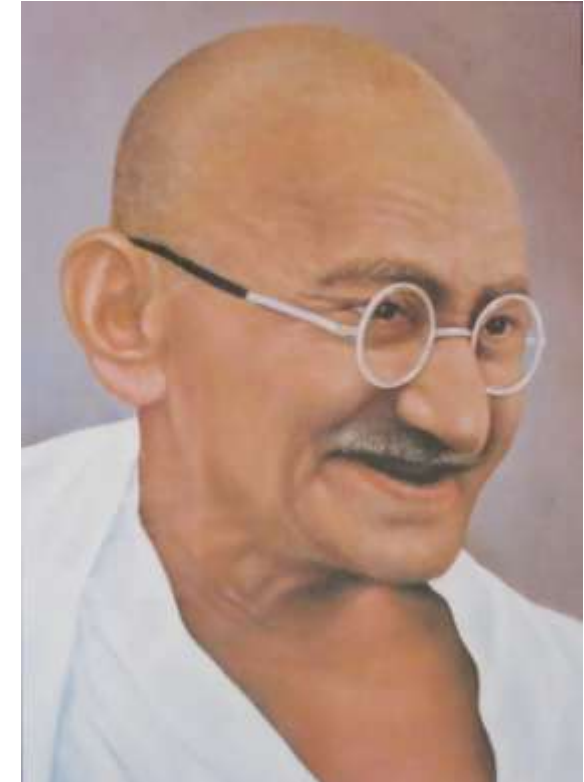
Sustainable Society Necessary for Human Survival

Almost Sustainable

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

Non-sustainable

- USA 17 ton CO₂, Sweden 7 ton CO₂/yr
- High meat consumption (1 kg beef uses ca 4000 L water for production)
- Hamburgers, unhealthy , includes beef
- Energy-consuming mechanized agriculture
- Transport dependent shopping 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
 - 3.6 released March 2023
- Modelica Association was established in 2000 in Linköping
 - Open, non-profit organization

Modelica 3.6 Language Standard Latest Release March 2023

- The Modelica language standard is in active development
- The Modelica 3.6 standard contains almost 130 (mostly small) updates and clarifications compared to the previous Modelica 3.5
- Aiming at support for larger and more complex system models
- Aiming at better portability of models



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

Language Specification

Version 3.6

March 9, 2023

Modelica Association

Abstract

This document defines the Modelica¹ language, version 3.6, which is developed by the Modelica Association, a non-profit organization with seat in Linköping, Sweden. Modelica is a freely available, object-oriented language for modeling of large, complex, and heterogeneous systems. It is suited for multi-domain modeling, for example, mechatronic models in robotics, automotive and aerospace applications involving mechanical, electrical, hydraulic control and state machine subsystems, process oriented applications and generation and distribution of electric power. Models in Modelica are mathematically described by differential, algebraic and discrete equations. No particular variable needs to be solved for manually. A Modelica tool will have enough information to decide that automatically. Modelica is designed such that available, specialized algorithms can be utilized to enable efficient handling of large models having more than one hundred thousand equations. Modelica is suited and used for hardware-in-the-loop simulations and for embedded control systems. More information is available at <https://modelica.org>

¹ Modelica is a registered trademark of the Modelica Association.

Modelica Conferences

- The 1st International Modelica conference October, 2000
- The 2nd International Modelica conference March 18-19, 2002
- The 3rd International Modelica conference November 5-6, 2003 in Linköping, Sweden
- The 4th International Modelica conference March 6-7, 2005 in Hamburg, Germany
- The 5th International Modelica conference September 4-5, 2006 in Vienna, Austria
- The 6th International Modelica conference March 3-4, 2008 in Bielefeld, Germany
- The 7th International Modelica conference Sept 21-22, 2009 in Como, Italy
- The 8th International Modelica conference March 20-22, 2011 in Dresden, Germany
- The 9th International Modelica conference Sept 3-5, 2012 in Munich, Germany
- The 10th International Modelica conference March 10-12, 2014 in Lund, Sweden
- The 11th International Modelica conference Sept 21-23, 2015 in Versailles, Paris
- The 12th International Modelica conference May 15-17, 2017 in Prague, Czech Rep
- The 13th International Modelica conference March 4-6, 2019, Regensburg, Germany
- The 14th International Modelica conference Sept 20-24, 2021, Linköping, Sweden
- The 15th International Modelica conference Oct 9-11, 2023, Aachen, Germany
- The 16th International Modelica conference Sept 8-10, 2025, Lucerne, Switzerland
- Also: Asian Modelica conferences 2016, 2017, 2018, 2020, 2022, 2024
- Also: US Modelica conference 2018, 2020, 2022, 2024

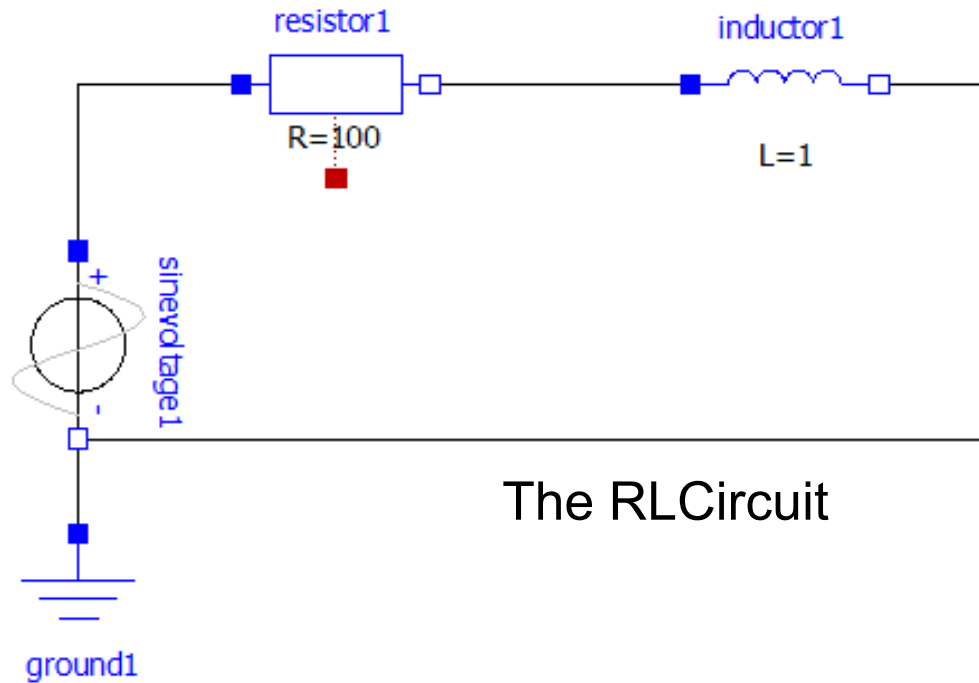
Exercises Part I

Hands-on graphical modeling

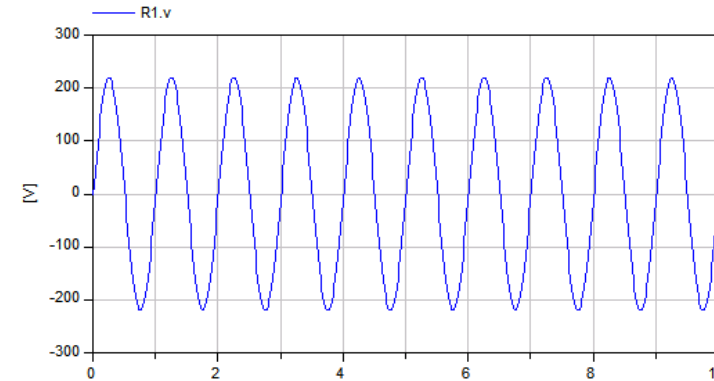
(15 minutes)

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**:

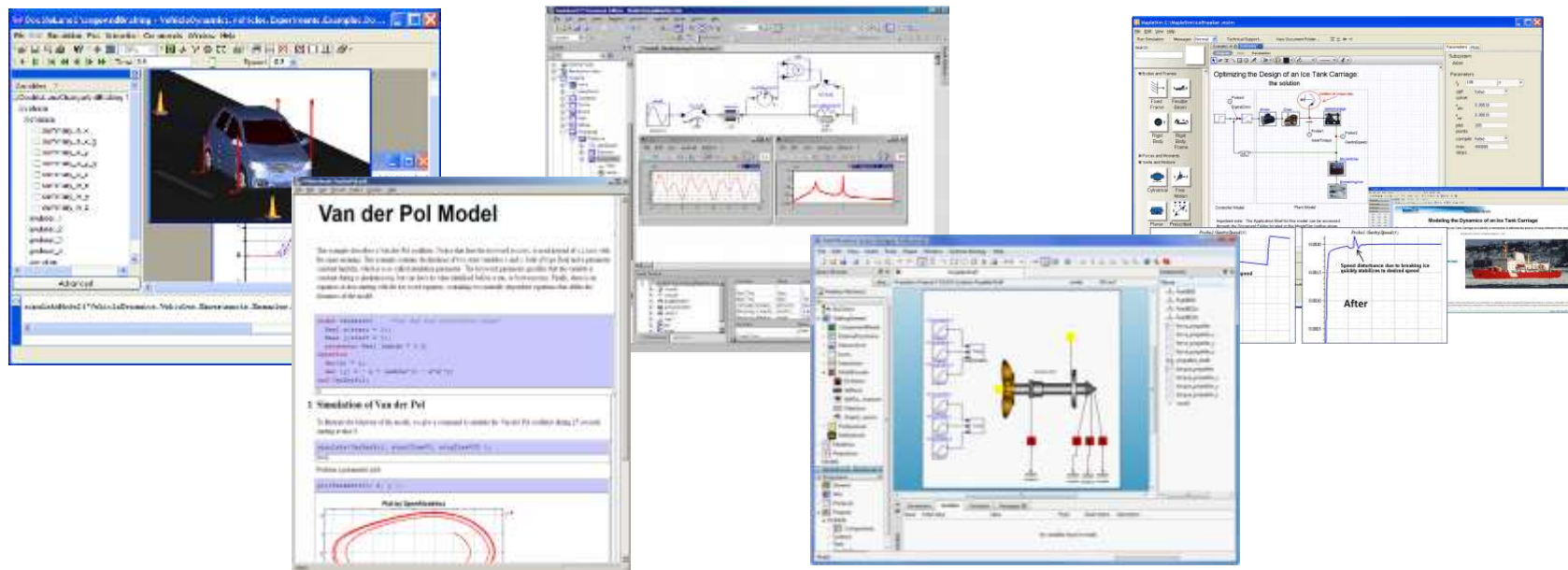
- Contains The standard Modelica library components
- The **Modelica files** contains the list of models you have created.

Exercises Part I – OMEdit Instructions (Part II)

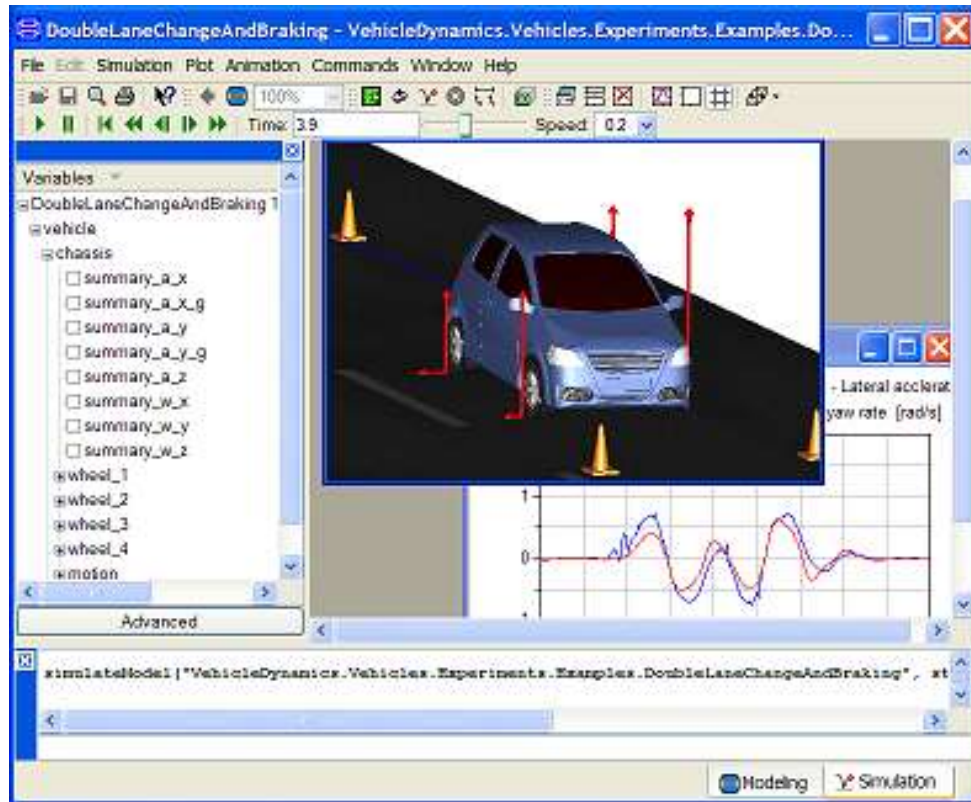
- For the RLCircuit model, **browse** the Modelica standard library and **add** the following component models:
 - Add `Ground`, `Inductor` and `Resistor` component models from `Modelica.Electrical.Analog.Basic` package.
 - Add `SineVoltage` component model from `Modelica.Electrical.Analog.Sources` package.
- 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

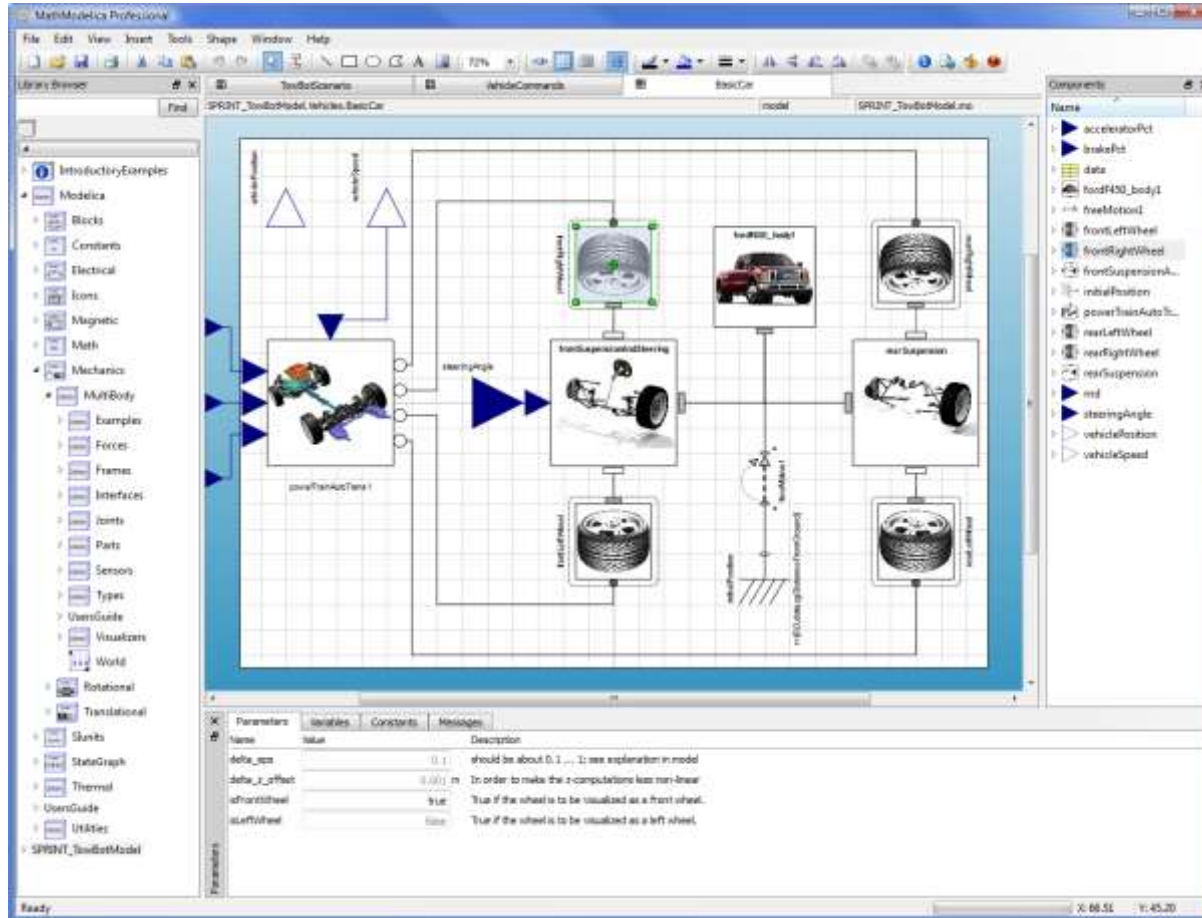


Dymola



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

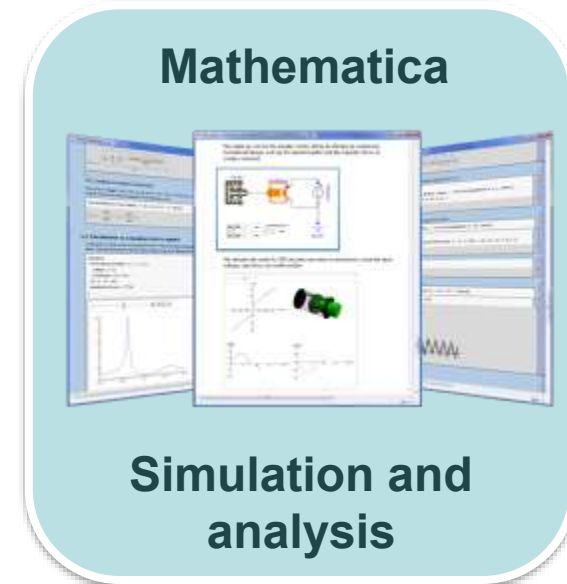
Wolfram System Modeler – Wolfram MathCore



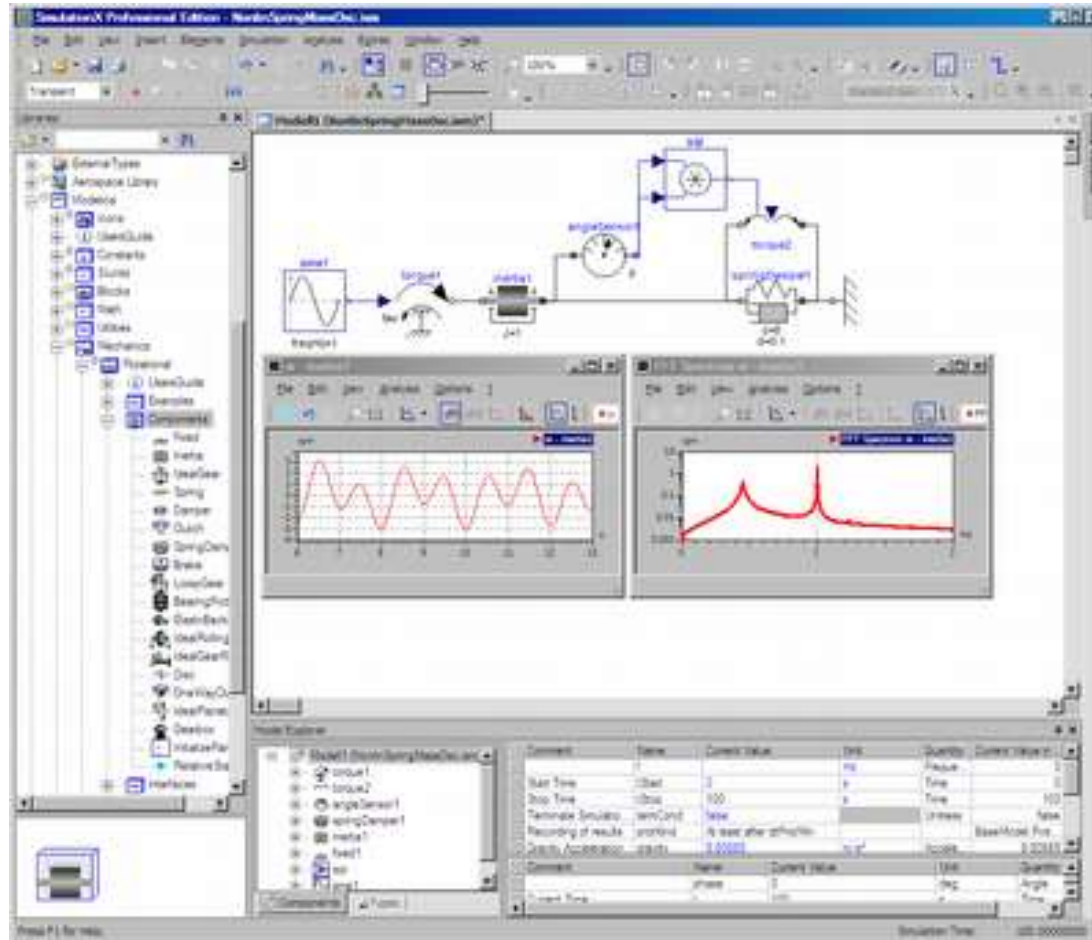
Courtesy
Wolfram
Research

Car model graphical view

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

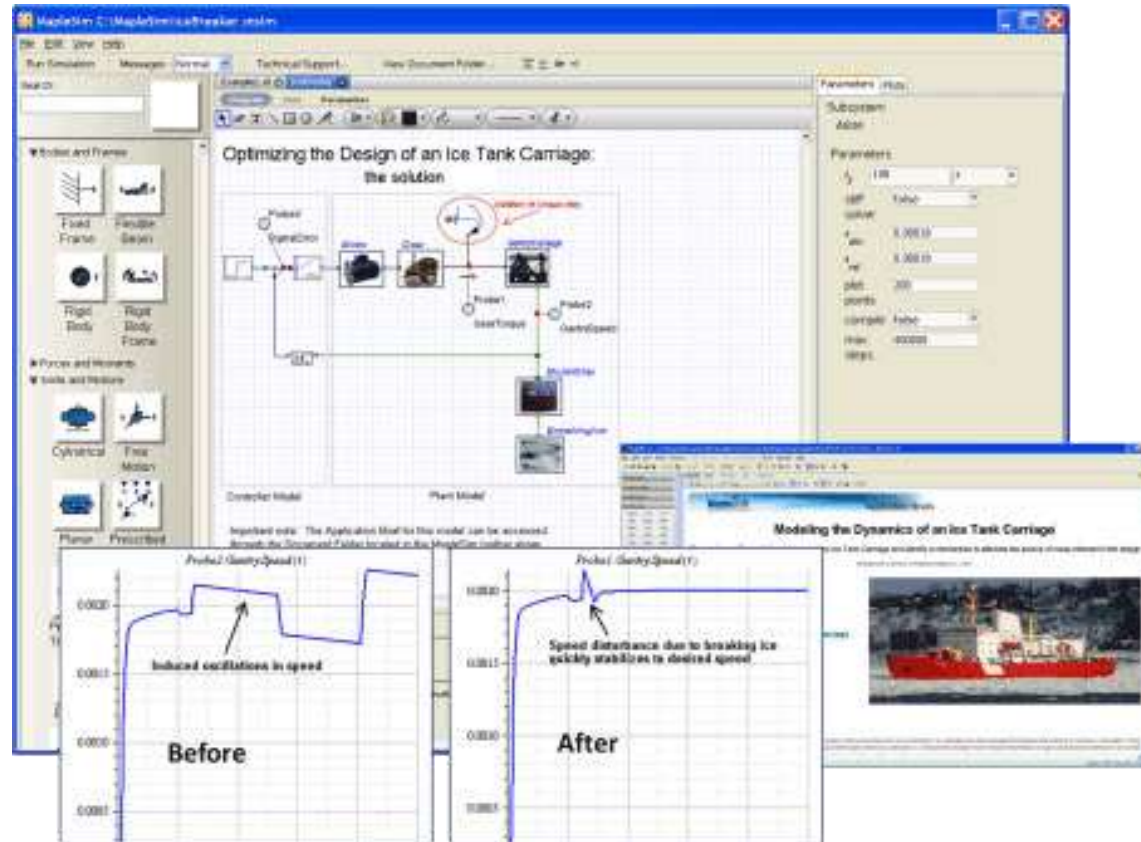


Simulation X



- ITI Gmbh (Part of ESI Group)
- Germany
- Mechatronic systems
- www.simulationx.com

MapleSim



- Maplesoft
- Canada
- Integrated with Maple
- 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

The OpenModelica Environment

www.OpenModelica.org




Home Download Users Developers Events Research

Introduction

OPENMODELICA is an open-source Modelica-based¹ 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.



Join the OpenModelicaInterest [mailing list](#) to get information about new releases.

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](#) Interactive [OMWebbook](#) with examples of Modelica textual modeling and [textbook companions](#) with application OpenModelica exercises. A [Jupyter notebook](#) Modelica mode, available in OpenModelica.

To get advice how to make existing Modelica libraries work in OpenModelica, see [Porting](#).

For systems engineering with requirement traceability and verification, see [ModelicaML](#).

OpenModelica provides [library coverage reports](#) of open-source Modelica libraries showing which libraries work well with OpenModelica and how the support improved over time.



Latest news and events 

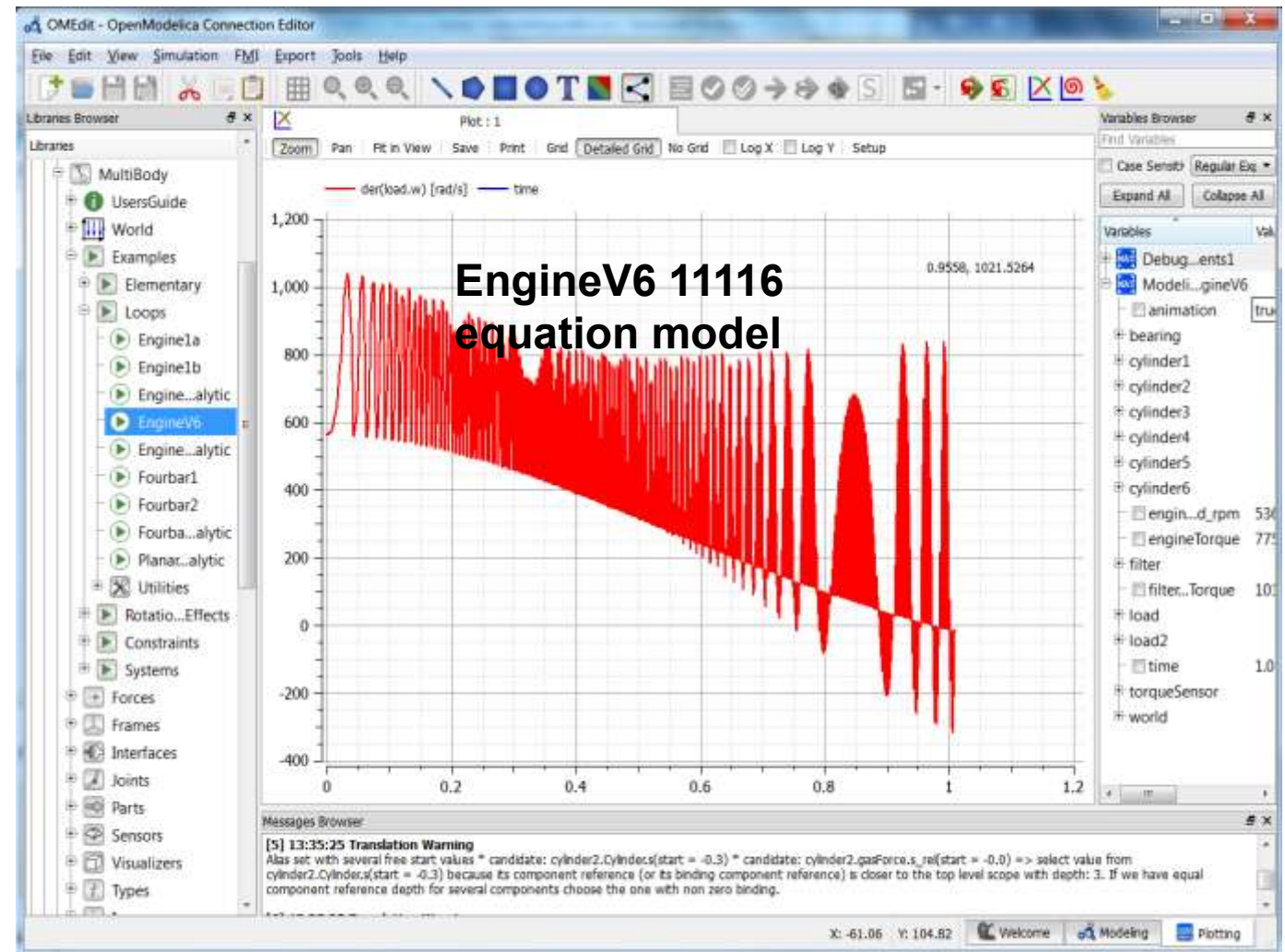
2025-09-04 [Openmodelica v1.25.3 released!](#)
2025-08-27 [Openmodelica v1.25.2 released!](#)
2025-06-15 [Openmodelica v1.25.1 released!](#)
2025-04-04 [Openmodelica v1.25.0 released!](#)
2025-03-19 [Openmodelica v1.25.0-dev.beta.1 released!](#)
2025-02-04 [Openmodelica v1.24.4 released!](#)
2025-01-07 [OpenModelica 2025](#)
2024-12-14 [Openmodelica v1.24.3 released!](#)
2024-11-15 [Openmodelica v1.24.2 released!](#)
2024-10-09 [Openmodelica v1.24.0 released!](#)
2024-09-16 [Openmodelica v1.24.0-dev.beta.0 released!](#)
2024-07-04 [Openmodelica v1.23.1 released!](#)
2024-06-06 [Openmodelica v1.23.0 released!](#)
2024-05-20 [Openmodelica v1.23.0-dev.beta.1 released!](#)
2024-03-12 [American Modelica Conference 2024!](#)
2024-03-11 [Openmodelica v1.22.3 released!](#)
2024-02-21 [Openmodelica v1.22.2 released!](#)
2024-02-05 [OpenModelica 2024](#)
2023-12-13 [Openmodelica v1.22.1 released!](#)
2023-11-08 [Openmodelica v1.22.0 released!](#)

1. This page references Modelica®, which is a registered trademark of [Modelica Association](#). ↗

OpenModelica – Free Open Source Tool

developed by the Open Source Modelica Consortium (OSMC)

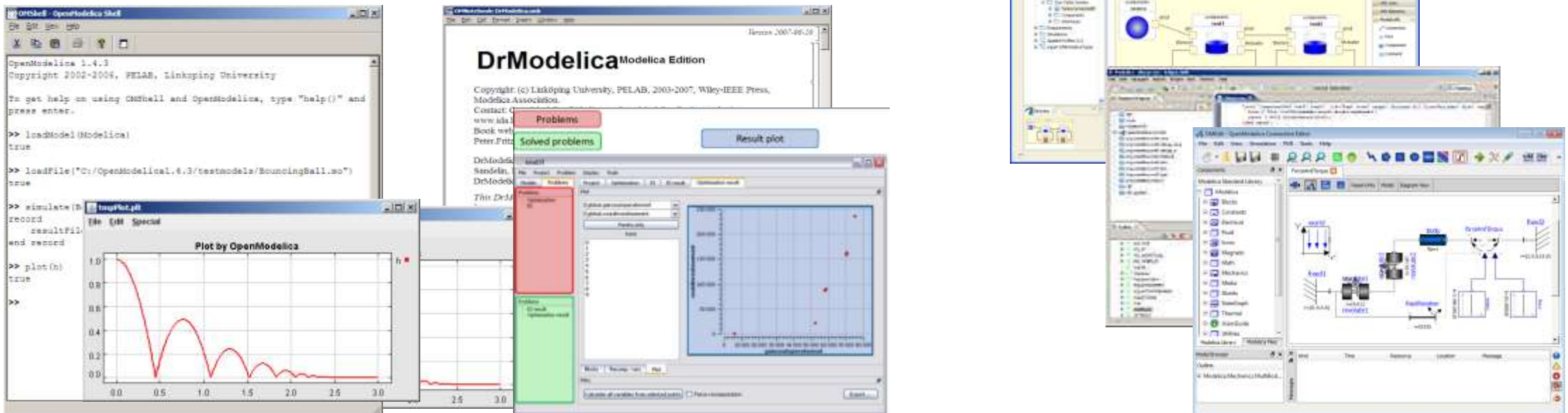
- Graphical editor
- Model compiler and simulator
- Debugger
- Performance analyzer
- Dynamic optimizer
- Symbolic modeling
- Parallelization
- Electronic Notebook and OMWebbook for teaching
- Spokentutorial for teaching



The OpenModelica Open Source Environment

www.openmodelica.org

- **Advanced Interactive Modelica compiler (OMC)**
 - Supports most of the Modelica Language
 - **Modelica, 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



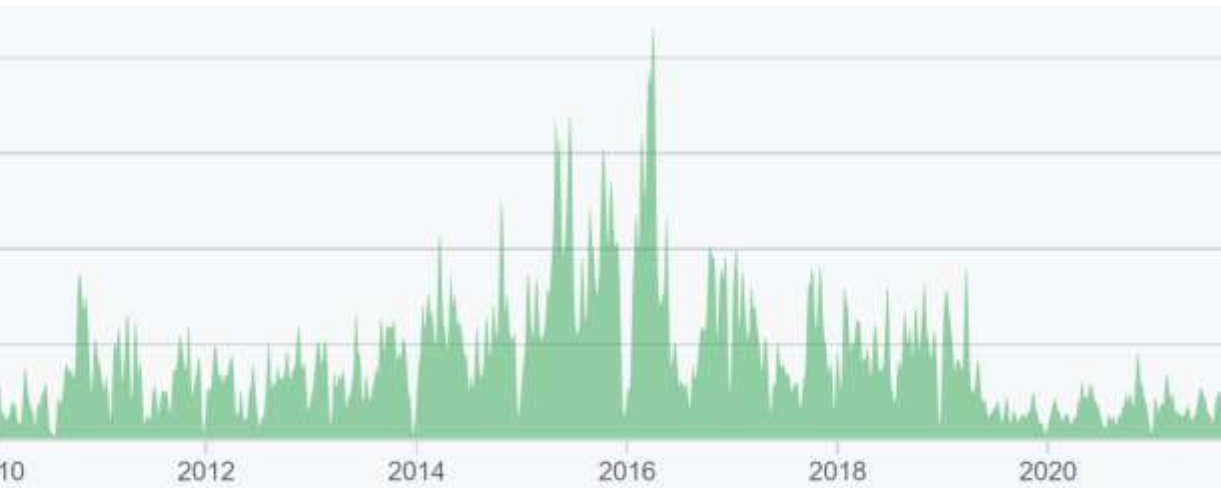
OSMC – International Consortium for Open Source Model-based Development Tools, 57 members 2025

Founded Dec 4, 2007

Open-source community services

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

Commits Statistics



Industrial members

- ABB AB, Sweden
- Bosch Rexroth AG, Germany
- Creative Connections, Prague
- DEMXS Ltd, Shanghai, China
- DHI, Aarhus, Denmark
- Dynamica s.r.l., Cremona, Italy
- EDF, Paris, France
- Equa Simulation AB, Sweden
- Festo, Esslingen, Germany
- Fraunhofer IWES, Bremerhaven
- Fraunhofer FCC, Gothenburg
- GSIMX Beijing Technology, China
- Hoerbiger Wien, Austria
- INRIA, Rennes, France

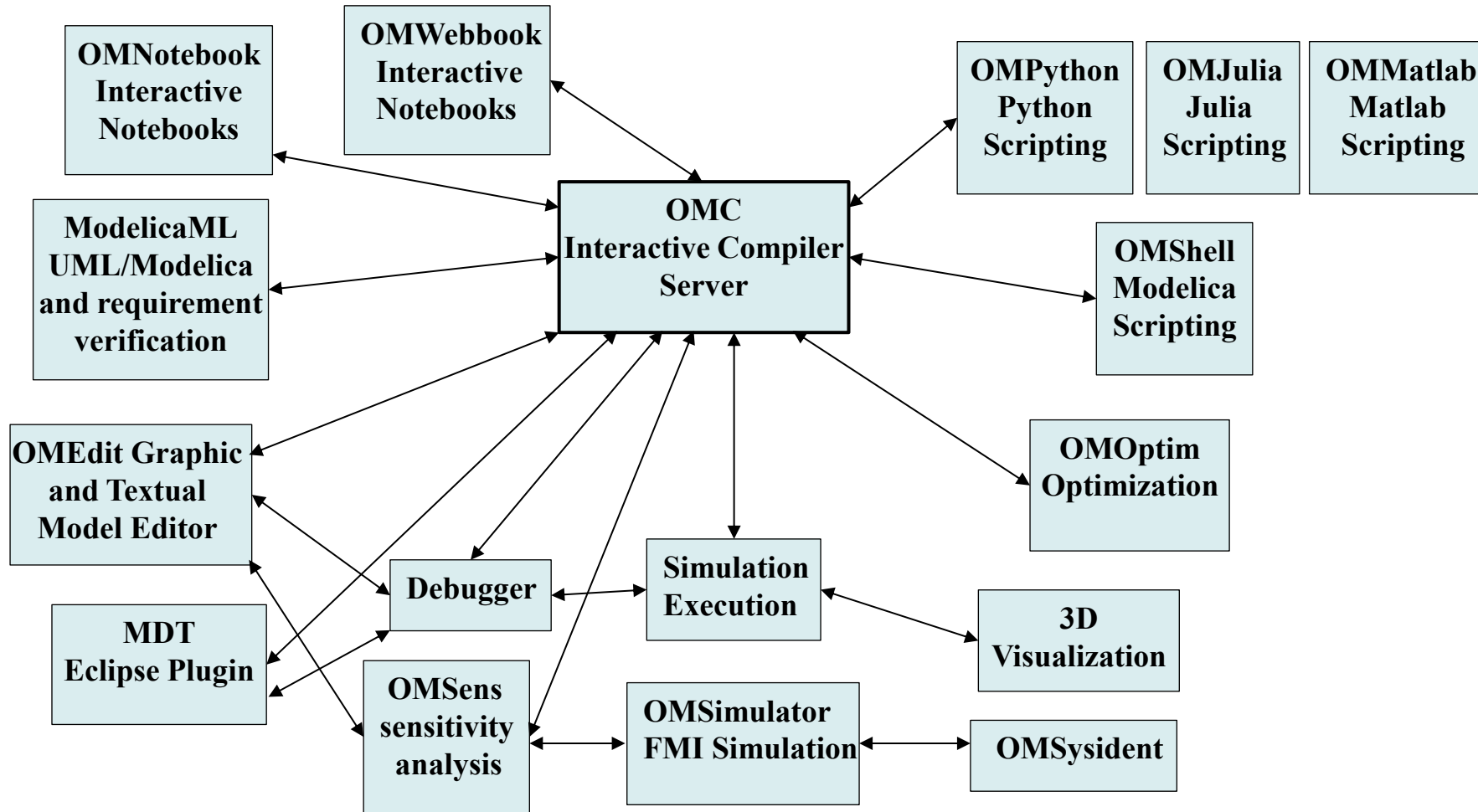
- Hefei Taize Turbomachinery Technology, Hefei, China
- ISID Dentsu, Tokyo, Japan
- Modelicon LLP, Bangalore, India
- Maplesoft, Canada
- Orthogonal Super Systems, Munich, Germany
- RTE France, Paris, France
- Saab AB, Linköping, Sweden
- Shanghai Duanyan Inf Tech China
- SIL3X, Paris, France
- Simtek, Nanjing, China
- SmartFluidPower, Italy
- Sozhou Tongyuan, China
- Swegon AB, Gothenburg, Sweden
- TLK Thermo, Germany
- VTI, Linköping, Sweden
- XRG Simulation, Germany

University members

- Augsburg University, Germany
- Australian Nation Univ., Australia
- Berkeley Lab, California, USA
- Hochschule Bielefeld, HSBI, Germany
- Huazhong University, NCD, China
- University of Bolivar, Colombia
- TU Braunschweig, Germany
- Univ of Buenos Aires, Argentina
- Univ Catalunya, Spain
- Chalmers Univ, Control, Sweden
- Chalmers Univ, Machine, Sweden
- TU Delft, The Netherlands
- TU Dresden, Germany
- Université Laval, Canada

- TU Hamburg/Harburg Germany
- KU Leuven, Leuven, Belgium
- IT Bombay, Mumbai, India
- Linneaus University, Sweden
- Linköping University, Sweden
- Univ of Maryland, Syst Eng USA
- Univ of Maryland, CEEE, USA
- Politecnico di Milano, Italy
- Mälardalen University, Sweden
- RPI, Troy, USA
- Univ Pisa, Italy
- Univ SouthEast Norway
- Vanderbilt Univ, USA

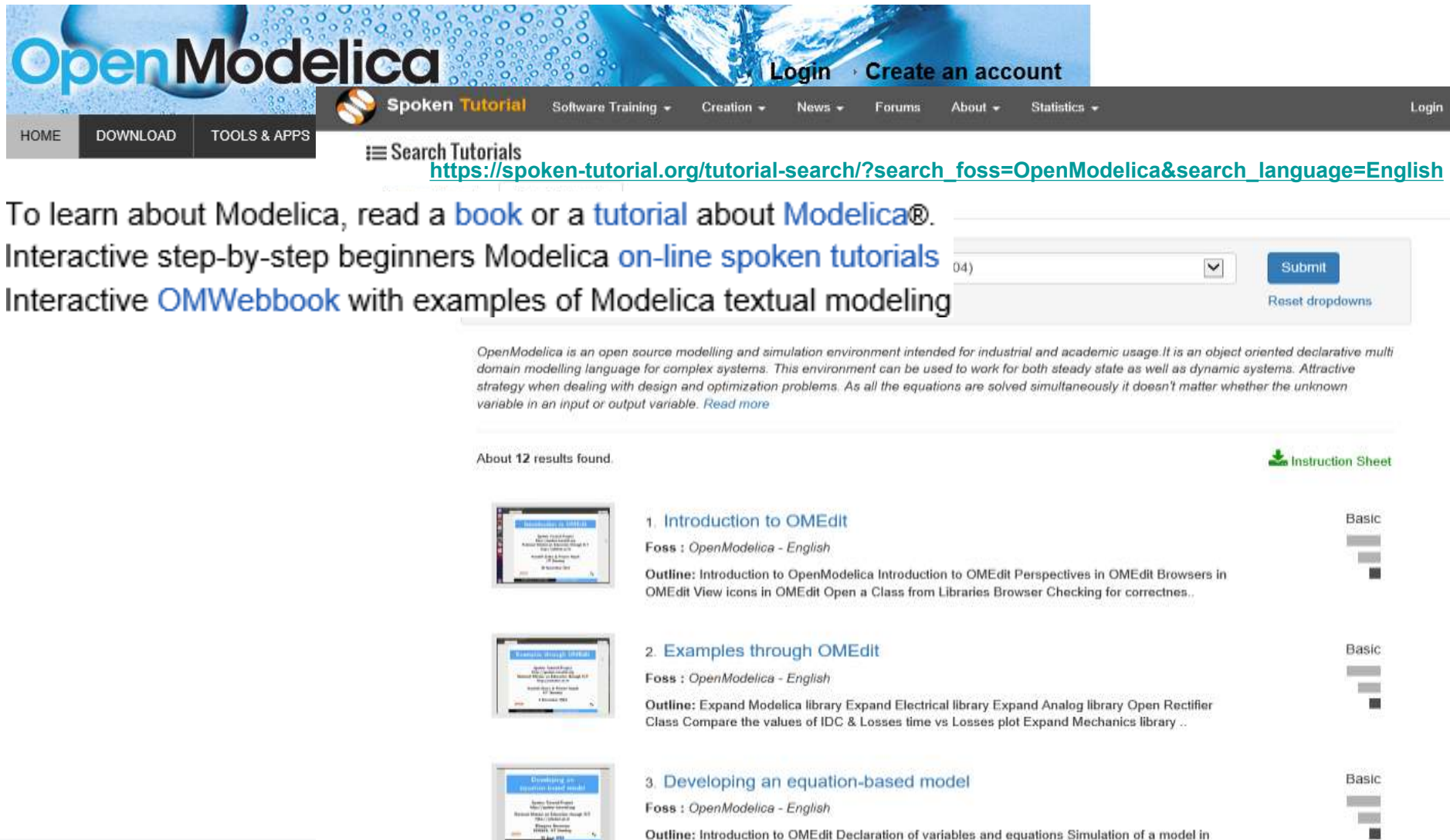
The OpenModelica Tool Architecture



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



The screenshot shows the OpenModelica website interface. At the top, there's a header with the OpenModelica logo and navigation links like 'Login', 'Create an account', 'Spoken Tutorial', 'Software Training', 'Creation', 'News', 'Forums', 'About', and 'Statistics'. Below the header, there's a search bar with the text 'Search Tutorials' and a URL: https://spoken-tutorial.org/tutorial-search/?search_foss=OpenModelica&search_language=English. The main content area features a search bar with a dropdown menu showing '04)' and a 'Submit' button. Below the search bar, there's a description of OpenModelica: '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'. Below this, it says 'About 12 results found.' and lists three results:

- 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

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

OMNotebook: Kalman.onb

File Edit Cell Format Insert Window Help

1 Kalman Filter

Often we don't have access to the internal states of a system and can only measure the output. We have to reconstruct the state of the system based on these measurements. This is normally done. The idea with an observer is that we feedback the difference of the measured output with the estimation. If the estimation is correct then the difference should be zero.

Another difficulty is that the measured quantities often contain disturbance, i.e. noise.

$$\begin{cases} \dot{\hat{x}} = A\hat{x} + Bu + e \\ \hat{y} = C\hat{x} + v \end{cases}$$

Here e denotes a disturbance in the input signal and v is a measurement error. The quality can be evaluated by the difference

$$K(y(t) - C\hat{x}(t) - Du(t))$$

By using this quantity as feedback we obtain the observer

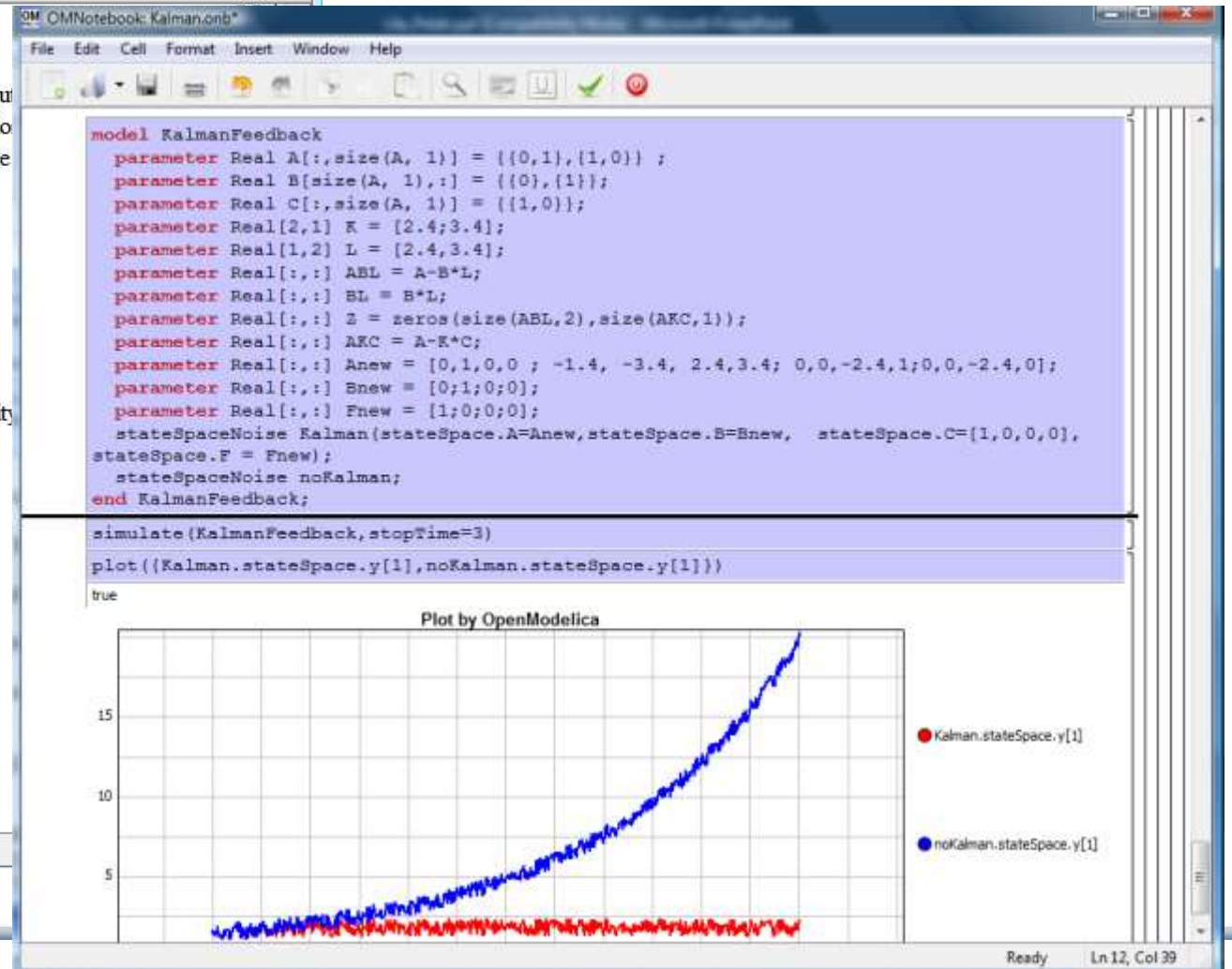
$$\dot{\hat{x}} = A\hat{x}(t) + Bu(t) + K(y(t) - C\hat{x}(t) - Du(t))$$

Now form the error as

$$\tilde{x} = x - \hat{x}$$

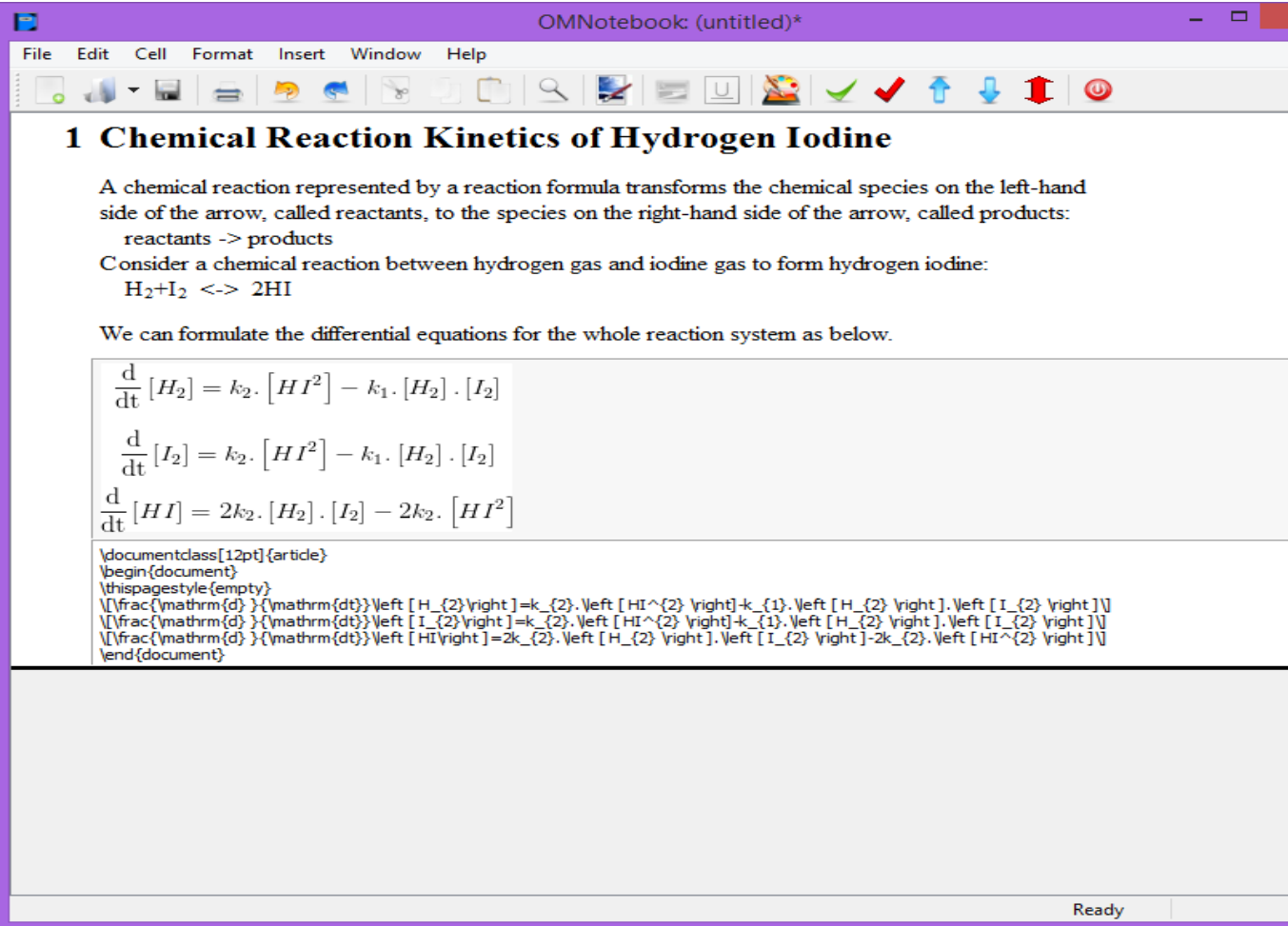
The differential error is

Ready



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.

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

Below the equations, the LaTeX source code is visible:

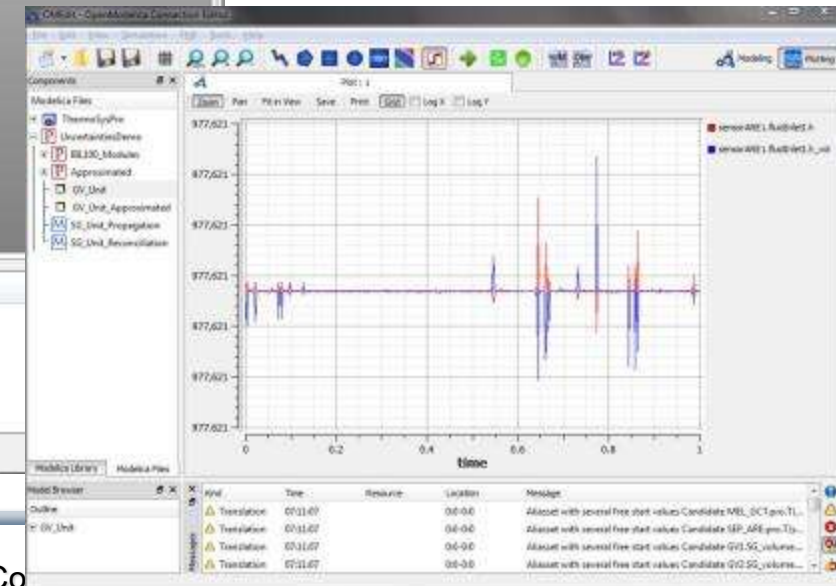
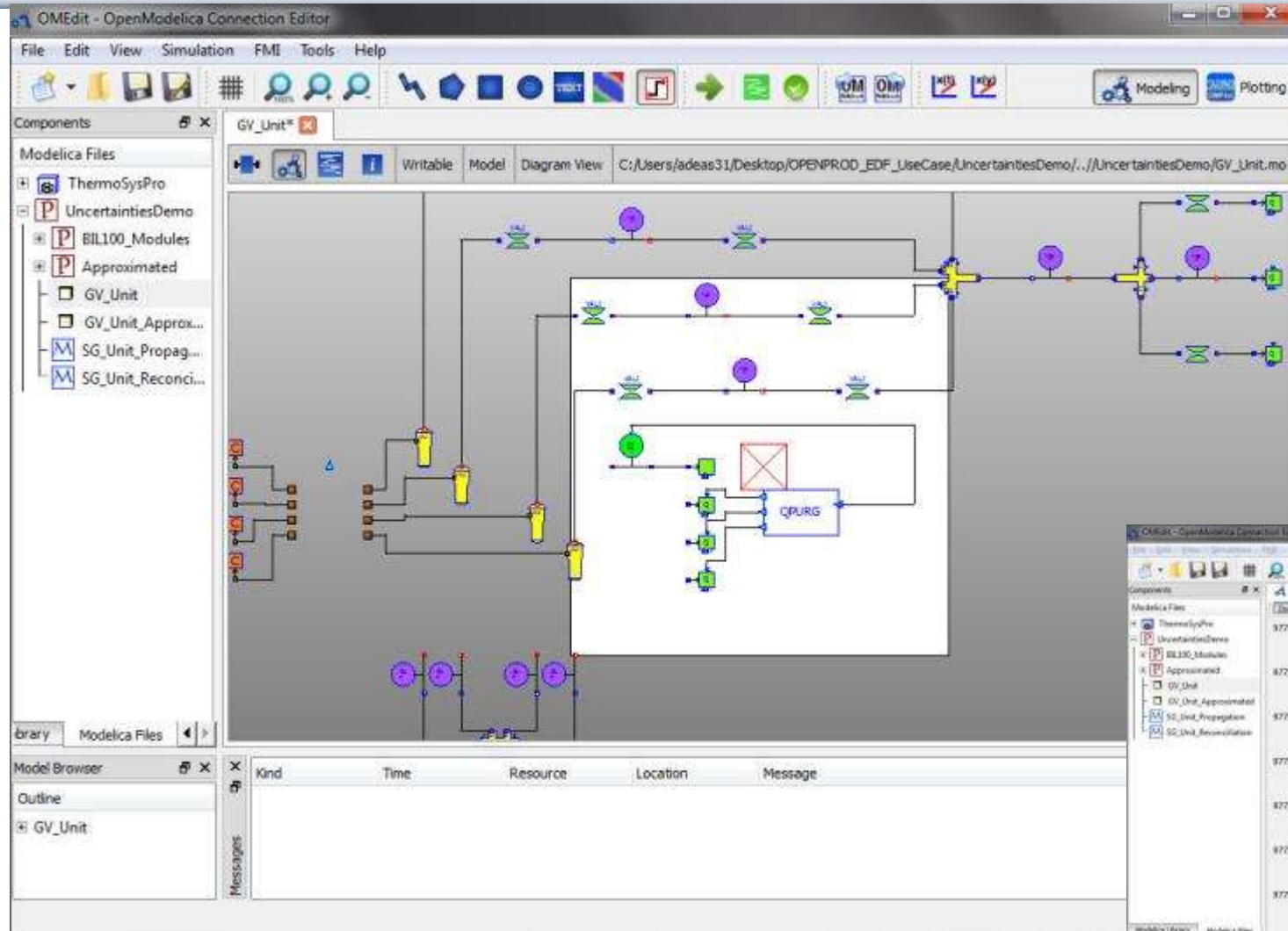
```
\documentclass[12pt]{article}
\begin{document}
\thispagestyle{empty}
\[\frac{\mathrm{d}}{\mathrm{d}t} \mathrm{[H_2]} = k_2 \cdot \mathrm{[HI^2]} - k_1 \cdot \mathrm{[H_2]} \cdot \mathrm{[I_2]}\]
\[\frac{\mathrm{d}}{\mathrm{d}t} \mathrm{[I_2]} = k_2 \cdot \mathrm{[HI^2]} - k_1 \cdot \mathrm{[H_2]}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

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

Contents in
OMWebbook
Generated from
OMNotebook

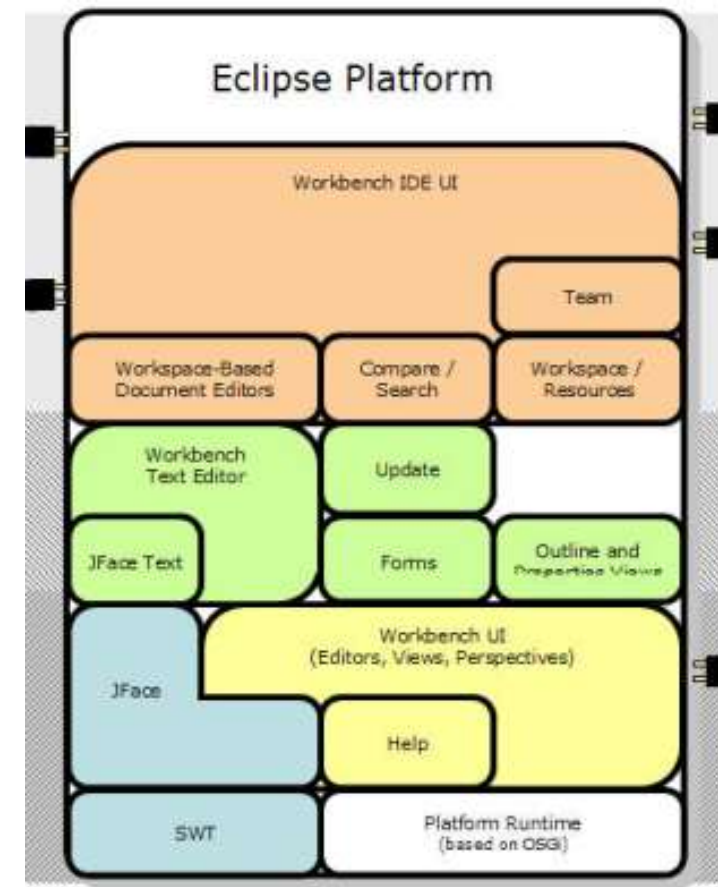
Latex instructions
can be hidden by
double clicking the
Cell in tree view

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 menu bar includes File, Edit, Navigate, Search, Project, Run, Field Assist, Window, and Help. The toolbar contains icons for file operations, running, and debugging. The left sidebar shows the 'Modelica Projects' tree with a list of files including rml2sig, runtime, scripts, test_codegen, tools, VC7, Absyn.mo, Algorithm.mo, BuiltIn.mo, Ceval.mo, ClassInf.mo, ClassLoader.mo, Codegen.mo, Connect.mo, Constants.mo, and Convert.mo. The main editor window shows the 'Absyn.mo' file with a code snippet. A tooltip is visible over the 'getCrefFromExp' function, providing its signature and description: 'function getCrefFromExp "function: getCrefFromExp Returns a flattened list of the component references in an expression"'. The bottom-left pane shows the 'Outline' view for the 'Absyn' package, listing various algorithm items like 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 113 errors, 0 warnings, and 0 infos. A large blue arrow points from the text 'Identifier Info on Hovering' to the tooltip. Another large blue arrow points from the text 'Code Outline for easy navigation within Modelica files' to the Outline view.

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
    l1 = getCrefFromExp(e1);
    l2 =
      function getCrefFromExp "function: getCrefFromExp
        Returns a flattened list of the
        component references in an expression"
      input Exp inExp;
      then
        output list<ComponentRef> outComponentRefList;
      algorithm
        outComponentRefList:=matchcontinue inExp
        local
          ComponentRef cr;
          l1 =
          l2 =
          res = listAppend(l1, l2);
    then
```

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
ComponentRef cr;
l1 =
l2 =
res = listAppend(l1, l2);
then

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

Identifier Info on Hovering

Code Outline for easy navigation within Modelica files

Ctrl Contrib (Bottom)

Modelica and MetaModelica Support in Visual Studio

Language server, Syntax highlighting and editing

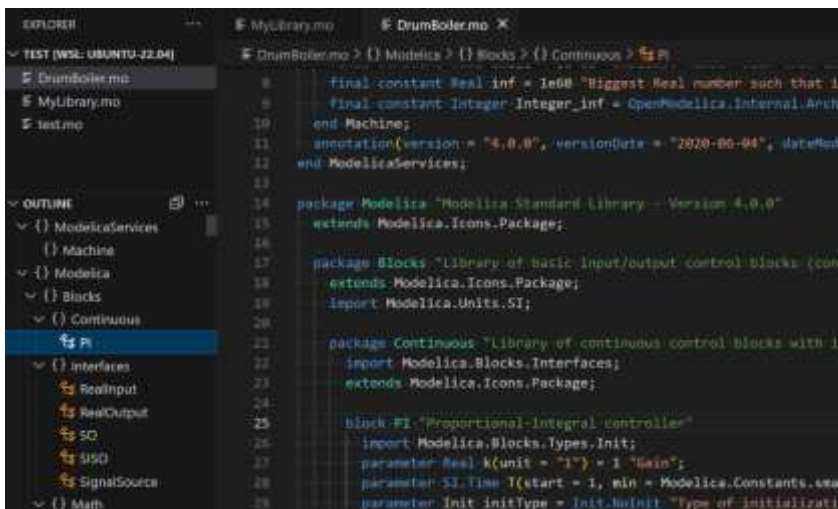
Modelica model syntax highlighting

```
2 model LotkaVolterra "Lotka-Volterra equations"
3   Real x(start = 1) "Prey";
4   Real y(start = 1) "Predator";
5   parameter Real lambda = 0.5;
6   input Real u;
7   equation
8     der(x) = x - x * y - lambda * u;
9     der(y) = -y + x * y - u;
10  end LotkaVolterra;
```

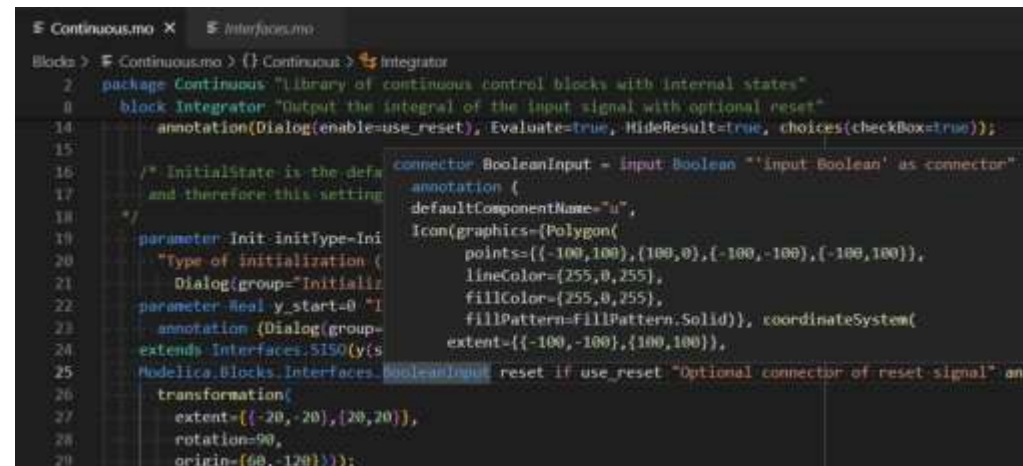
Modelica function syntax highlighting

```
fu
mo function
  Modelica function definition
  Real y(start = 1) "Predator";
  parameter Real lambda = 0.5;
  input Real u;
  equation
    der(x) = x - x * y - lambda * u;
    der(y) = -y + x * y - u;
  end LotkaVolterra;
```

Provide outline of Modelica files



Goto Modelica declaration



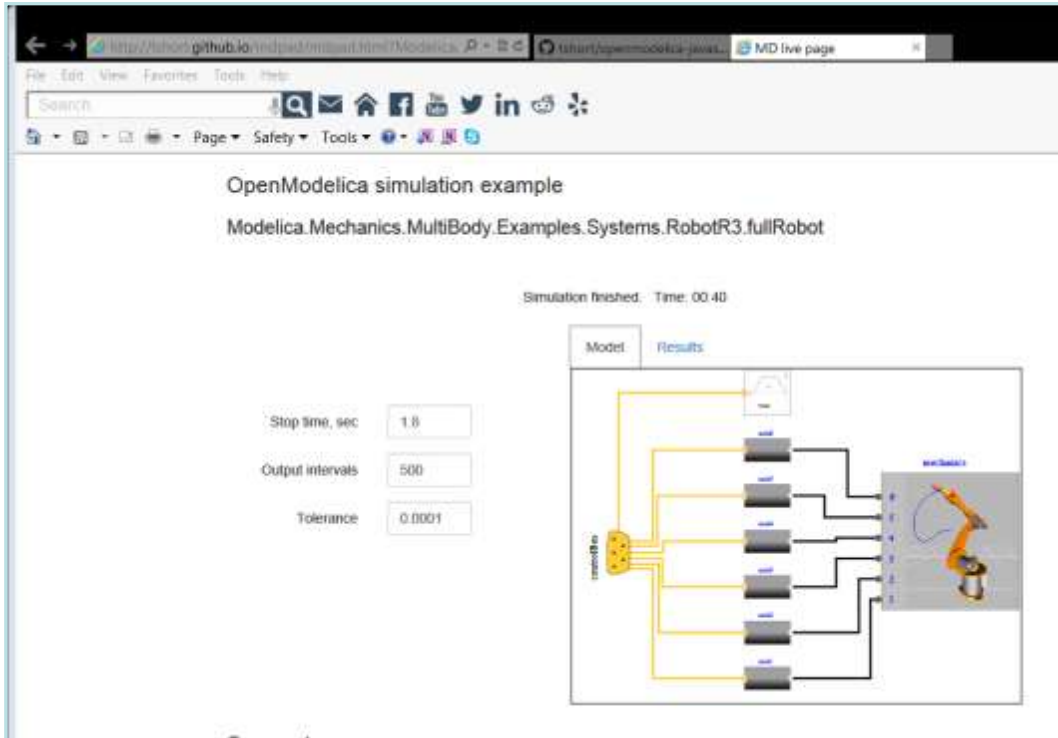
VSCode extensions for Modelica (<https://github.com/OpenModelica/modelica-language-server>)

and MetaModelica (<https://github.com/OpenModelica/metamodelica-language-server>).

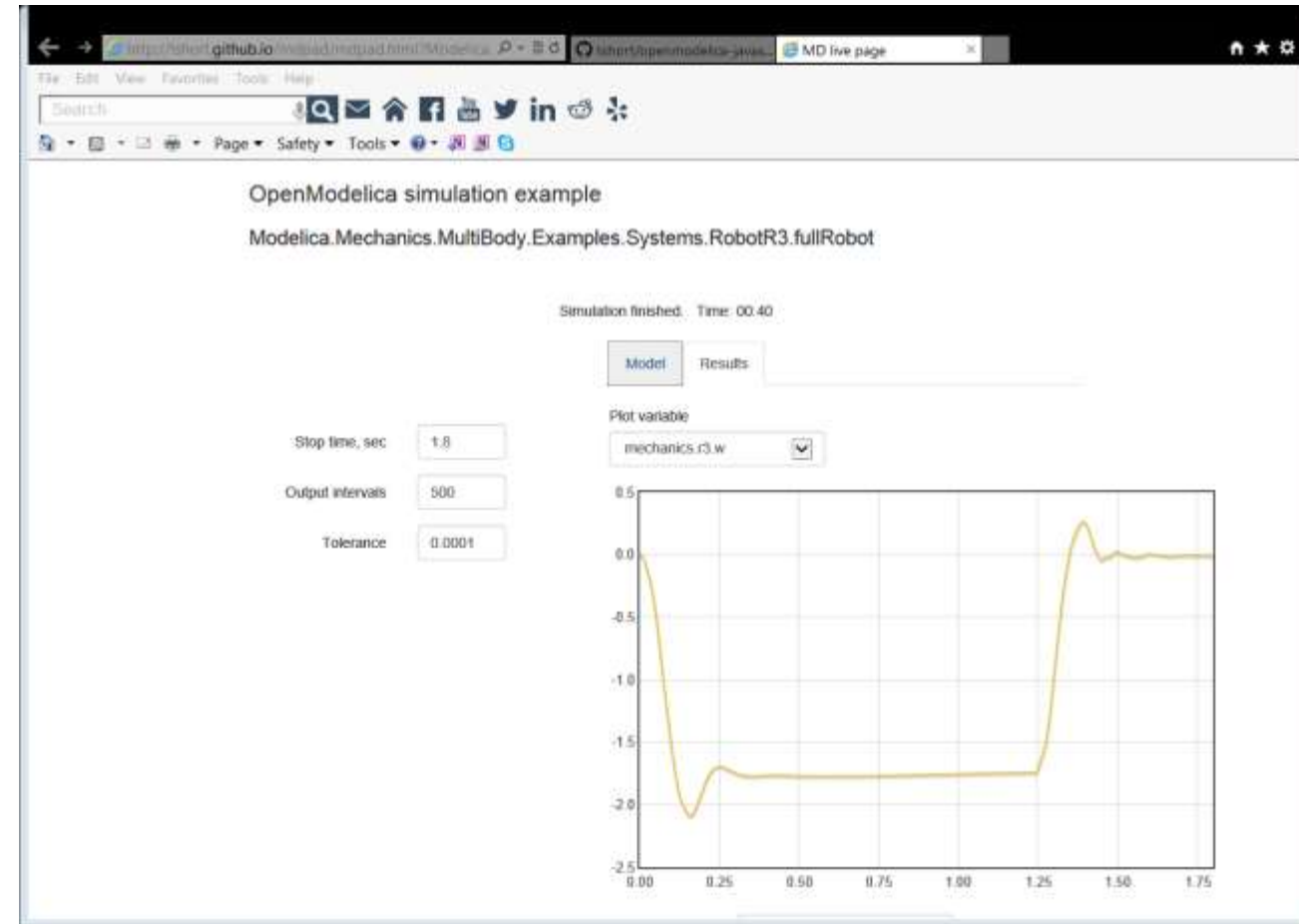
Syntax highlighting <https://marketplace.visualstudio.com/items?itemName=AnHeuermann.metamodelica>

OpenModelica Simulation in Web Browser Client

MultiBody RobotR3.FullRobot

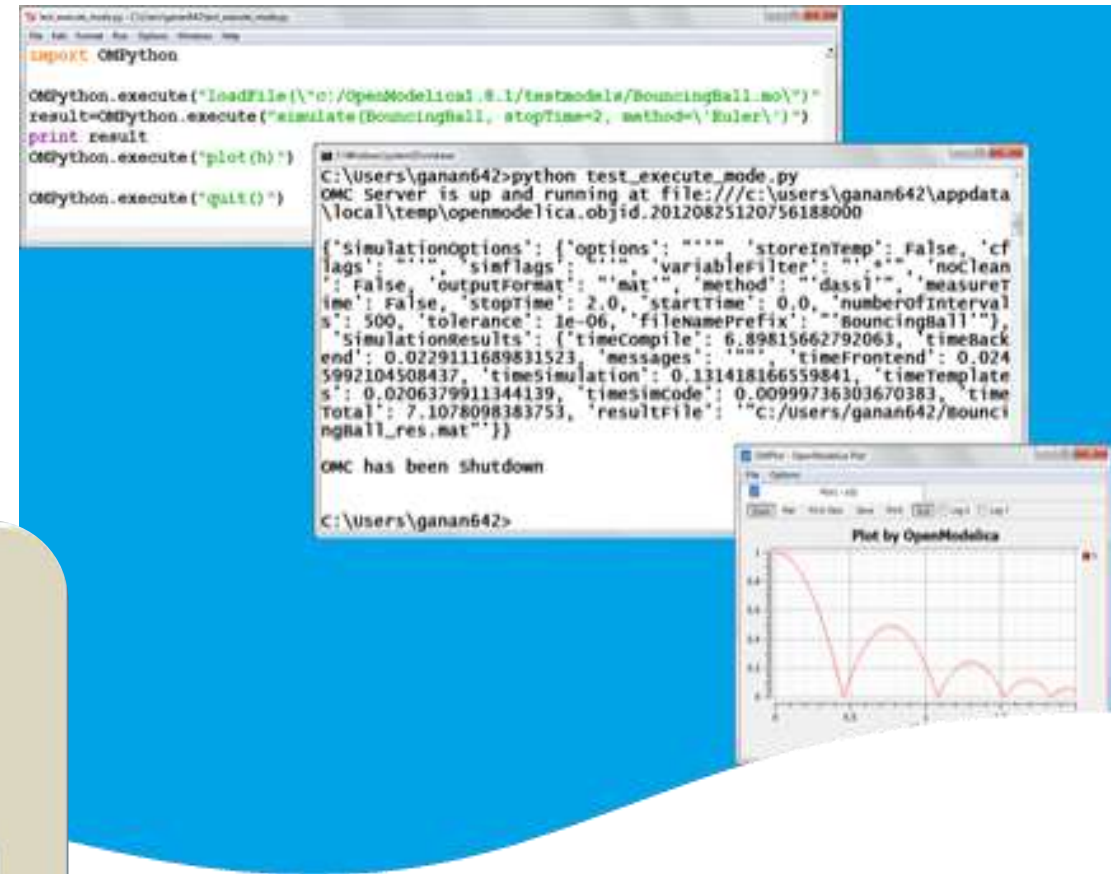
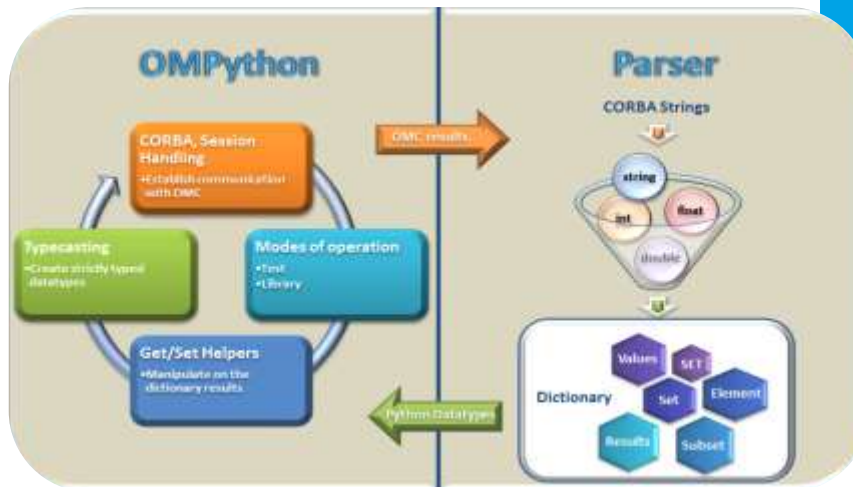


OpenModelica compiles to efficient JavaScript code which is executed in the web browser



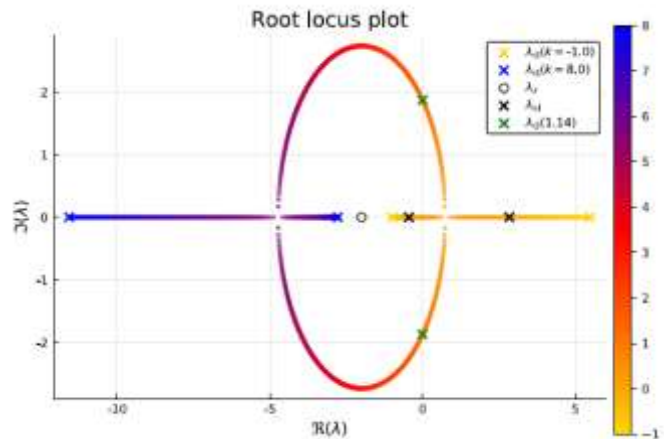
OMPython – Python Scripting with OpenModelica

- 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"

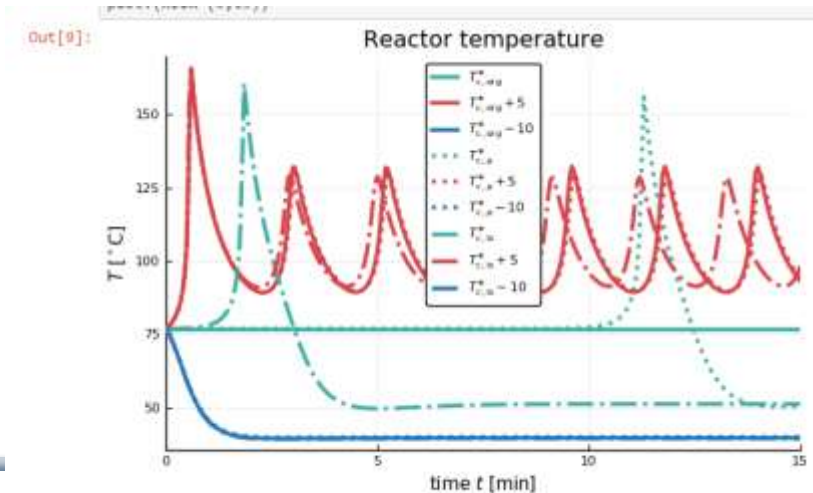
Bernt Lie*, Arunkumar Palanisamy**, Peter Fritzson**

*University of South-Eastern Norway, Norway

**University of Linköping, Sweden

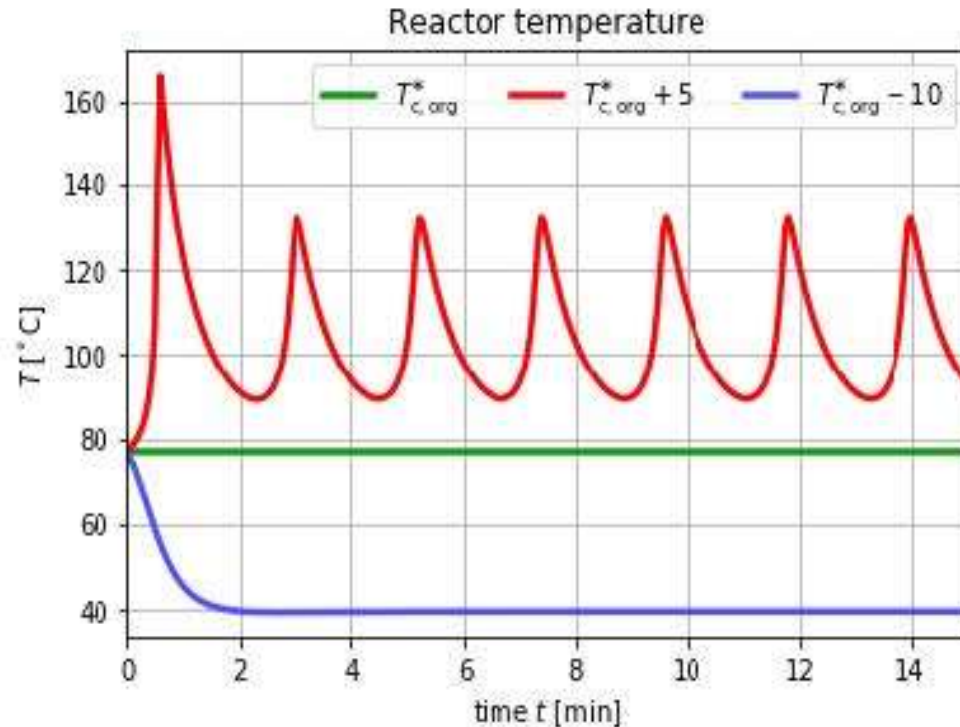
Introducing packages

```
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
```



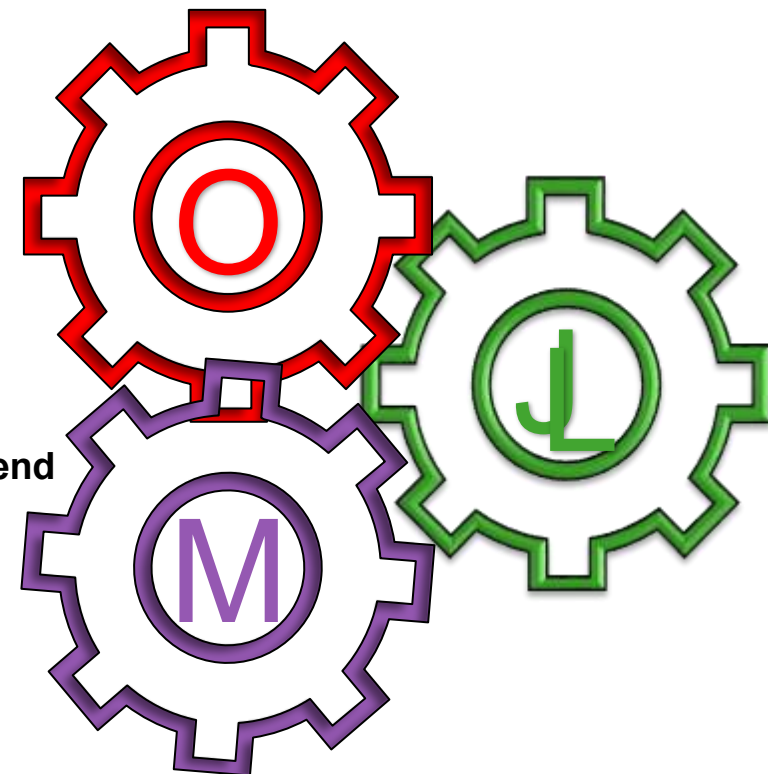
OMMatlab – Matlab Scripting with OpenModelica

- 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

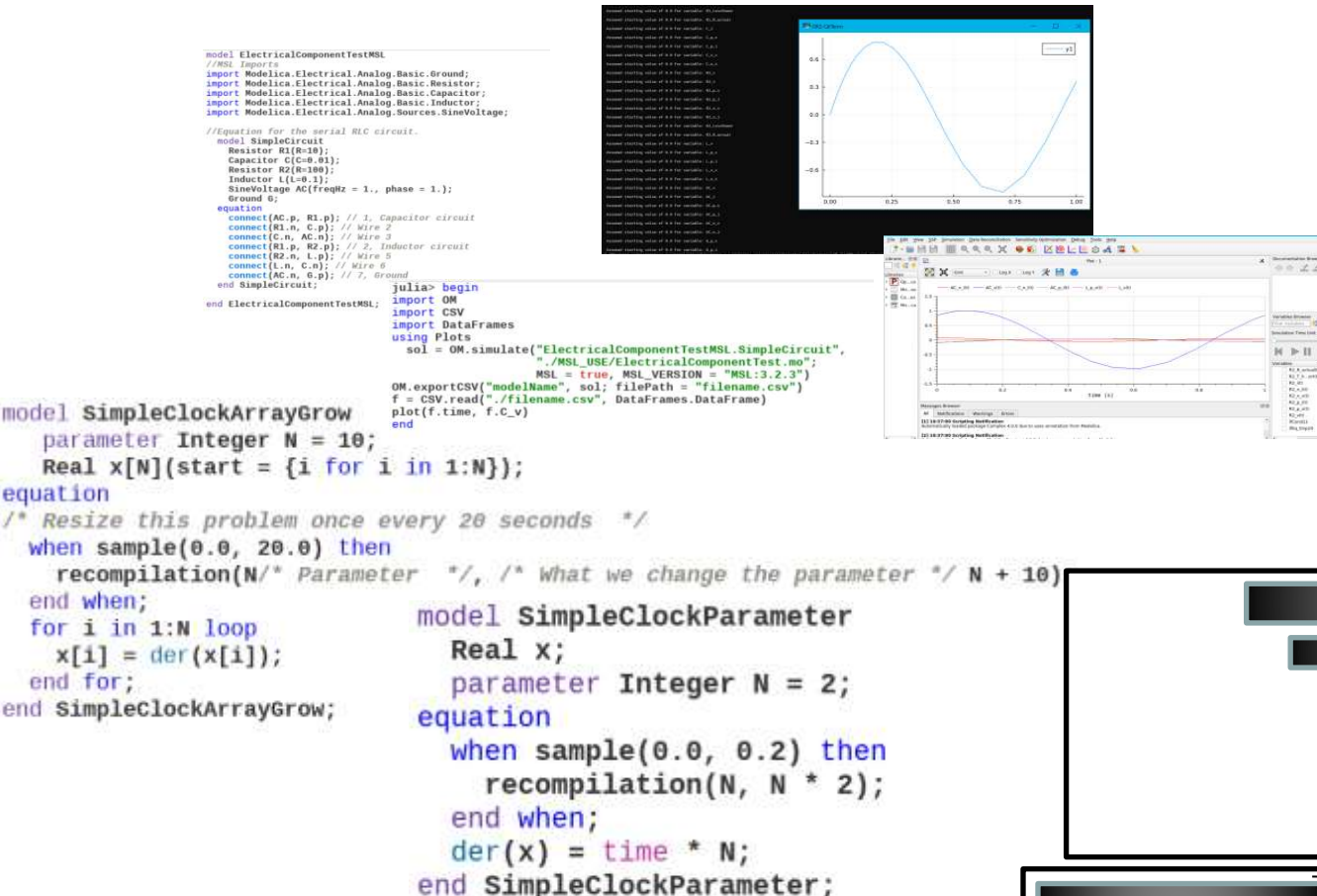


Experimental OpenModelica Environment in Julia (OMJL)

- **OpenModelica.jl**: A Modular and Extensible Modelica compiler framework in Julia (OMJL). PhD thesis work. Defense planned Dec 2025.
- **Translated the high-performance frontend.**
- **Able to execute and translate Modelica/MetaModelica functions**
- **Able to simulate discrete-hybrid systems + regular continuous systems**
- **Experimental backends developed**
 - Targeting DifferentialEquations.jl (low-level code generator) and ModelingToolkit.jl (MTK)
 - Causalization sorting, matching...
 - Integrated various software packages, DAG representation of the hybrid DAE
 - Integration with Sundials
 - Integrated Plots.jl for interactive plotting and animation
- Alpha is released, and a Beta Release is in development
 - **MSL support (New 2022/2023)**
 - **OpenModelica C-runtime support (New 2024)**
 - **Updated for Julia 1.10 (New 2024)**
 - **Migration to 1.11 (In progress)**
 - **Optimization of both runtime and compiler structures in the frontend and in the backend**
 - **MsC Thesis using OpenModelica.jl to propose a new language done at TU-Dresden Autumn 2023**
- Supporting Experimental Modelica Features:
 - **Language extensions for variable-structure system support (2022)**
 - **Dynamic Overconstrained Connectors (2022)**
 - **Theta Operator (New 2023)**



Features of OMJL



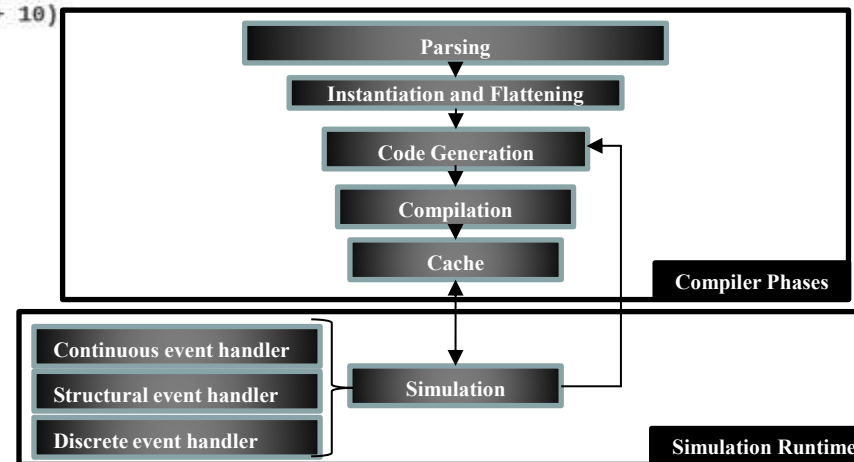
Static Reconfiguration via separate flattening

```

struct FLAT_MODEL <: FlatModel
  name::String
  variables::Vector{Variable}
  equations::Vector{Equation}
  initialEquations::Vector{Equation}
  algorithms::Vector{Algorithm}
  initialAlgorithms::Vector{Algorithm}
  #= VSS Modelica extension =#
  structuralSubmodels::List{FlatModel}
  scodeProgram::Option{SCode.CLASS}
  #= Dynamically Overconstrained connectors =#
  DOCC_equations::List{Equation}
  #= Contains the set of unresolved connect equations =#
  unresolvedConnectEquations::List{Equation}
  active_DOCC_Equations::Vector{Bool}
  #= End VSS Modelica extension =#
  comment::Option{SCode.Comment}
end

```

- Possible to formulate recursive models that expand during simulation



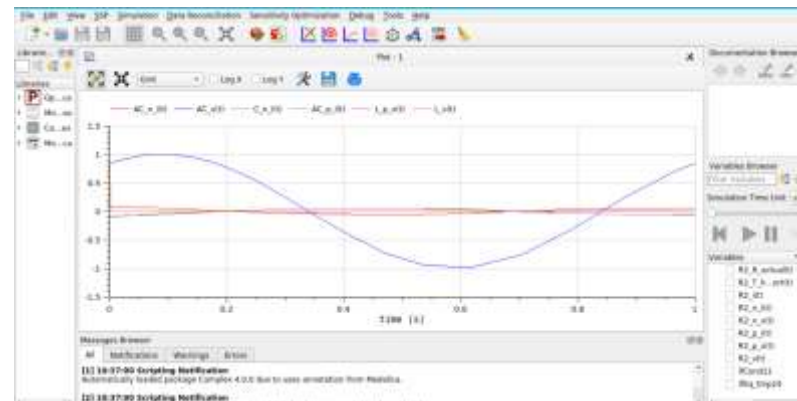
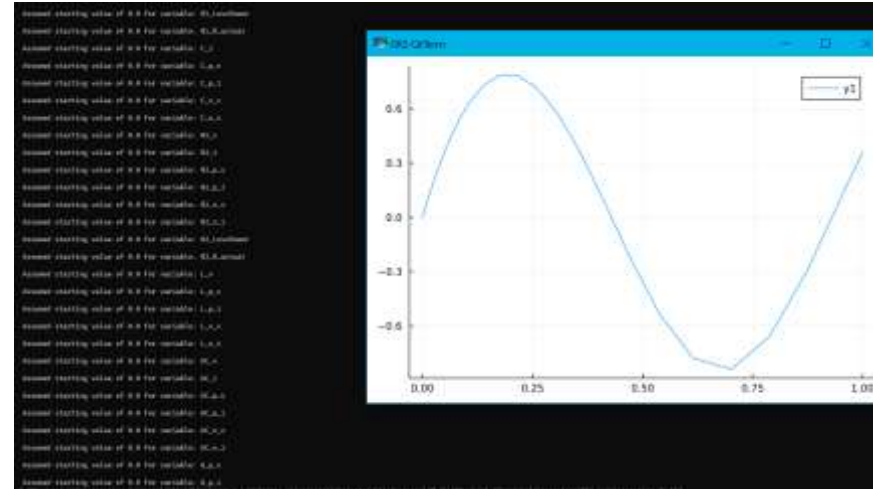
- Frontend was modified s.t it can flatten models in separation
- Note requires the structural mode keyword
- Possible to formulate model with varying index and compile AOT.

Practical Examples using the Standard Library with OMJL

```
model ElectricalComponentTestMSL
//MSL Imports
import Modelica.Electrical.Analog.Basic.Ground;
import Modelica.Electrical.Analog.Basic.Resistor;
import Modelica.Electrical.Analog.Basic.Capacitor;
import Modelica.Electrical.Analog.Basic.Inductor;
import Modelica.Electrical.Analog.Sources.SineVoltage;

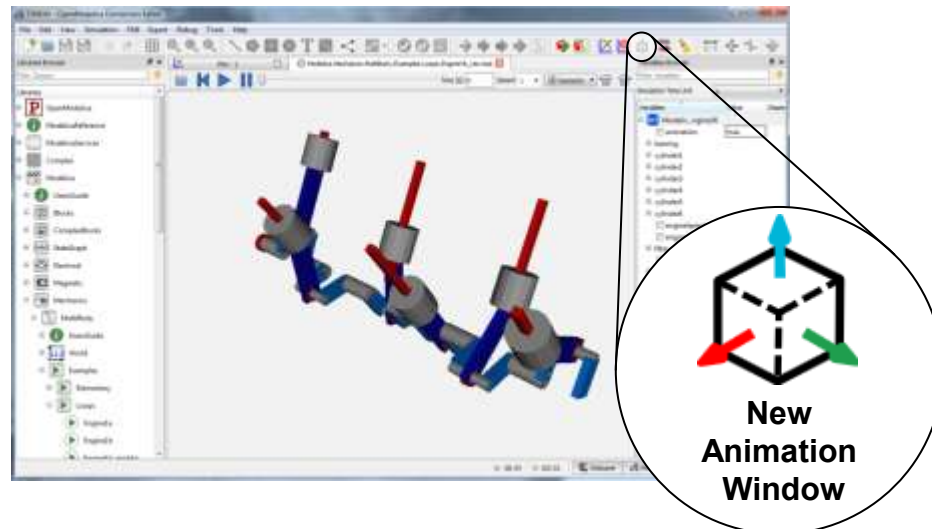
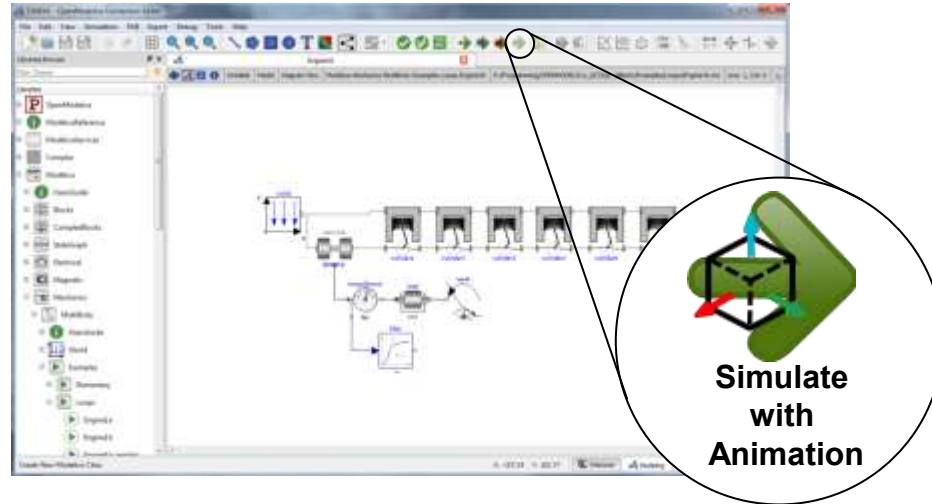
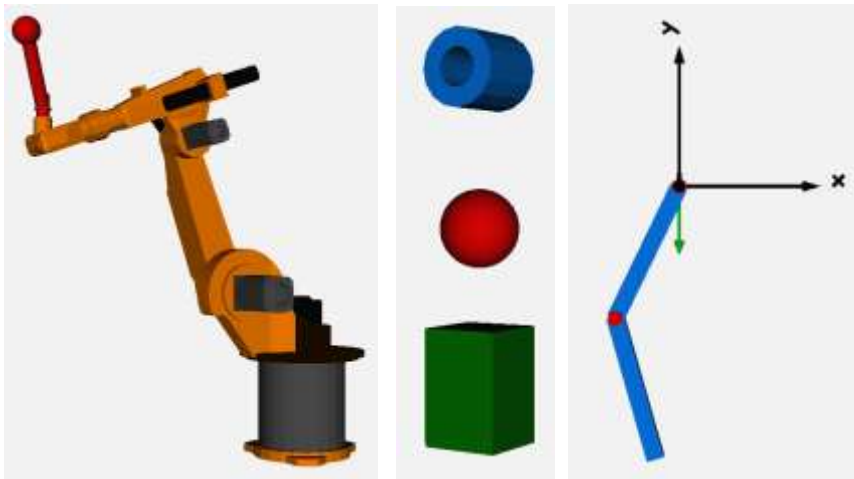
//Equation for the serial RLC circuit.
model SimpleCircuit
  Resistor R1(R=10);
  Capacitor C(C=0.01);
  Resistor R2(R=100);
  Inductor L(L=0.1);
  SineVoltage AC(freqHz = 1., phase = 1.);
  Ground G;
equation
  connect(AC.p, R1.p); // 1, Capacitor circuit
  connect(R1.n, C.p); // Wire 2
  connect(C.n, AC.n); // Wire 3
  connect(R1.p, R2.p); // 2, Inductor circuit
  connect(R2.n, L.p); // Wire 5
  connect(L.n, C.n); // Wire 6
  connect(AC.n, G.p); // 7, Ground
end SimpleCircuit;
end ElectricalComponentTestMSL;
```

```
julia> begin
import OM
import CSV
import DataFrames
using Plots
sol = OM.simulate("ElectricalComponentTestMSL.SimpleCircuit",
  "/MSL_USE/ElectricalComponentTest.mo";
  MSL = true, MSL_VERSION = "MSL:3.2.3")
OM.exportCSV("modelName", sol; filePath = "filename.csv")
f = CSV.read("./filename.csv", DataFrames.DataFrame)
plot(f.time, f.C_v)
end
```

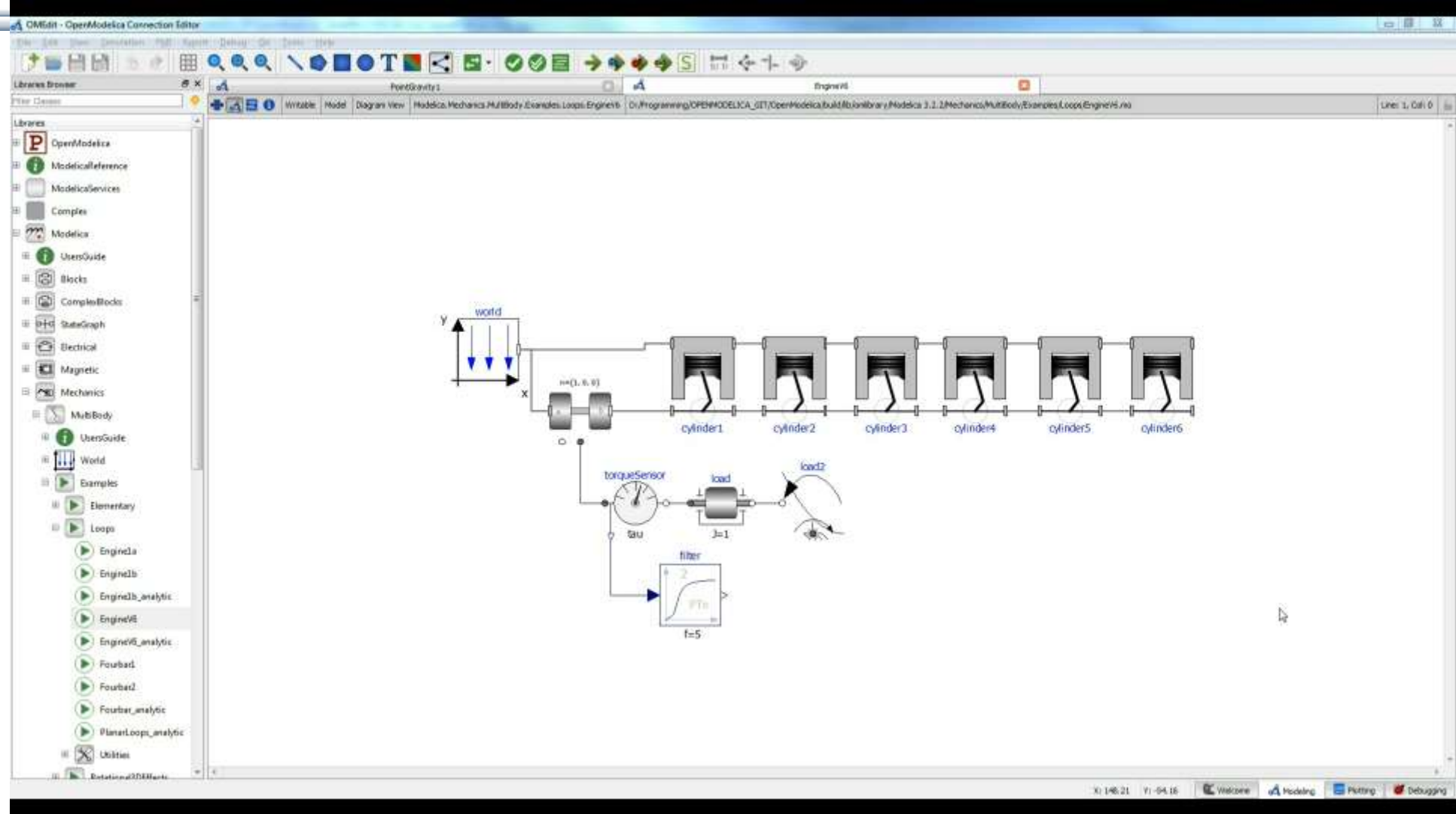


OMEdit 3D Visualization of Multi-Body Systems

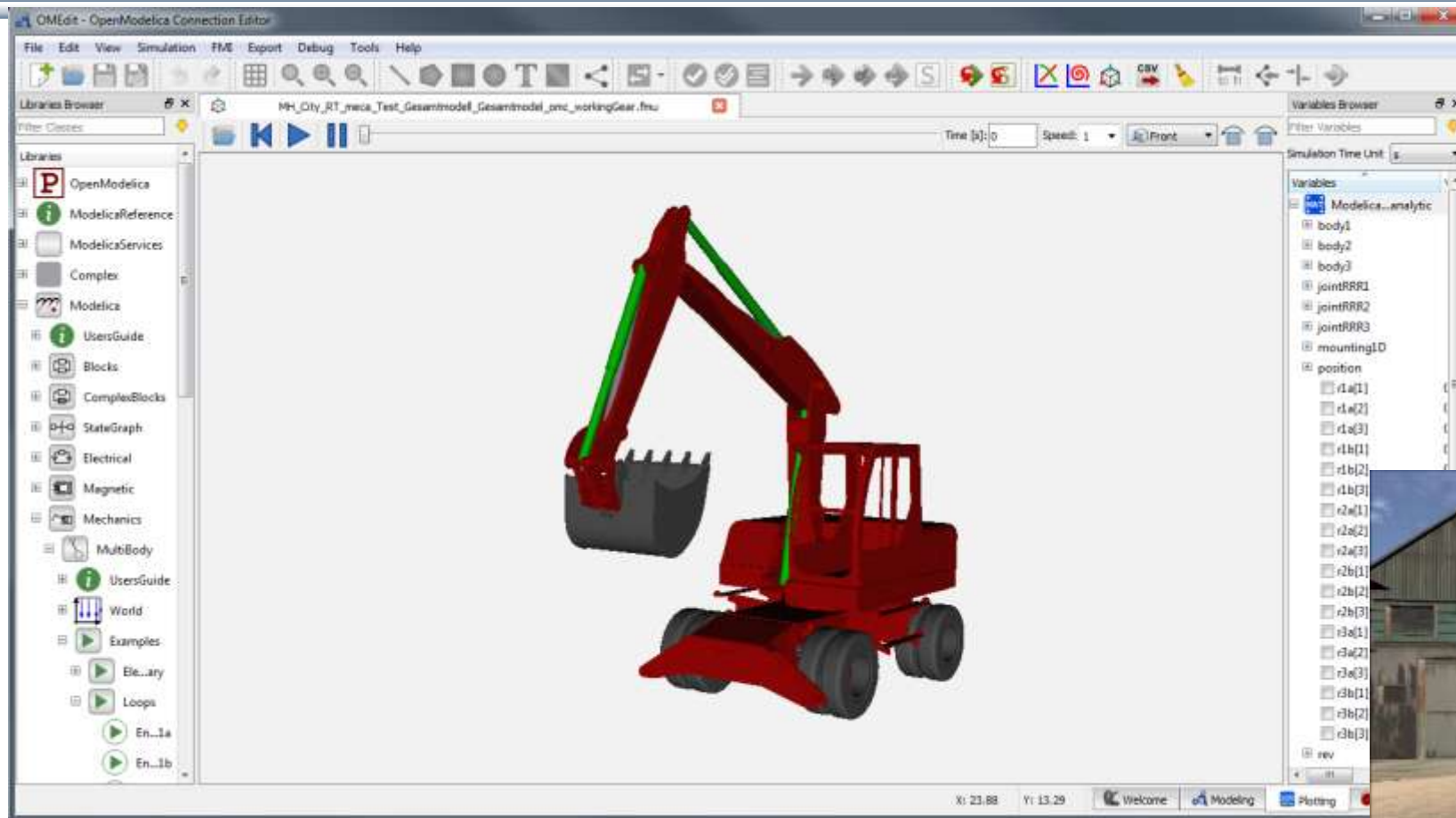
- Built-in feature of OMEdit to animate MSL-Multi-Body shapes
- Visualization of simulation results
- Animation of geometric primitives and CAD-Files



OpenModelica 3D Animation Demo (V6Engine and Excavator)



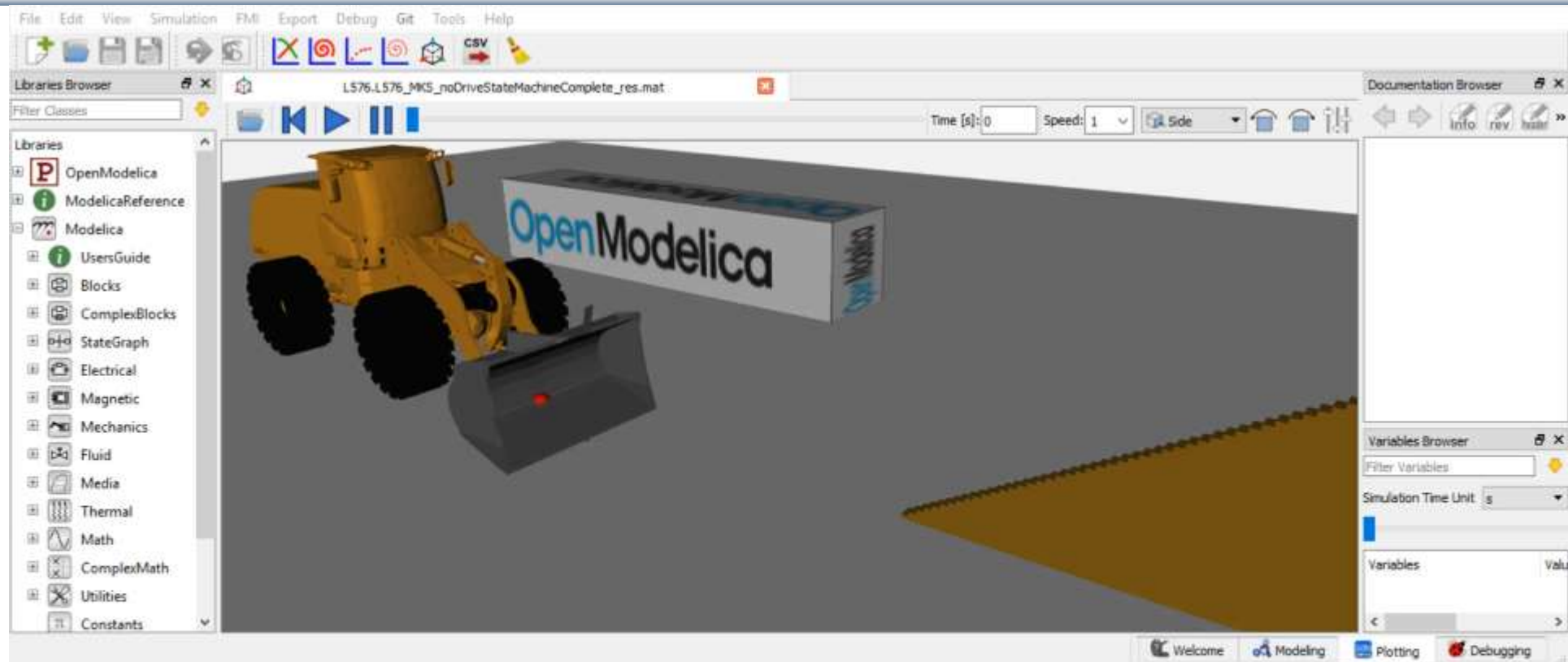
OpenModelica 3D Animation – Excavator



Connection with Unity

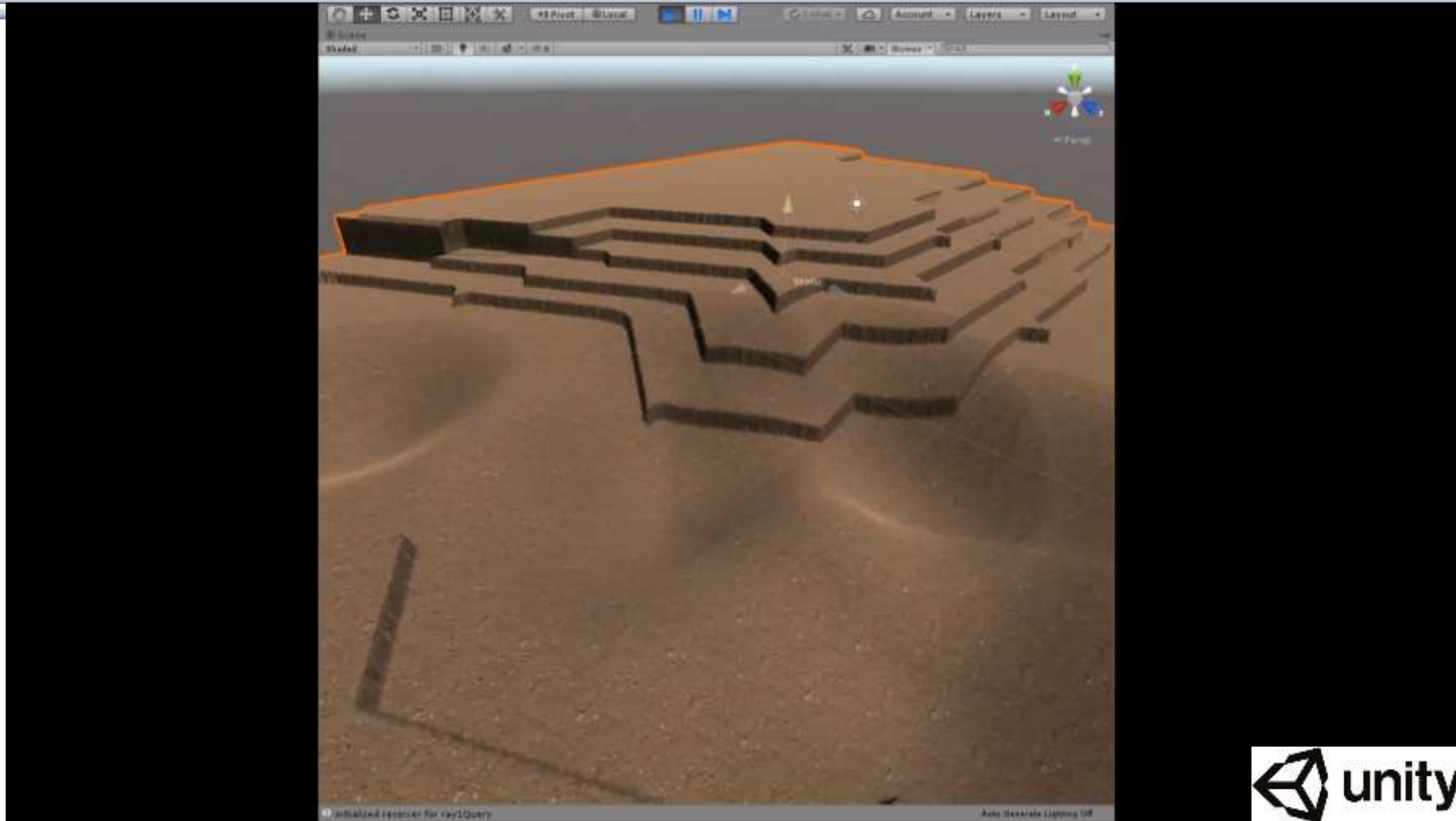
Courtesy of Volker Waurich - TU Dresden

OpenModelica 3D Animation – WheelLoader



Courtesy of Volker Waurich - TU Dresden

OpenModelica 3D Animation – BouncingBall

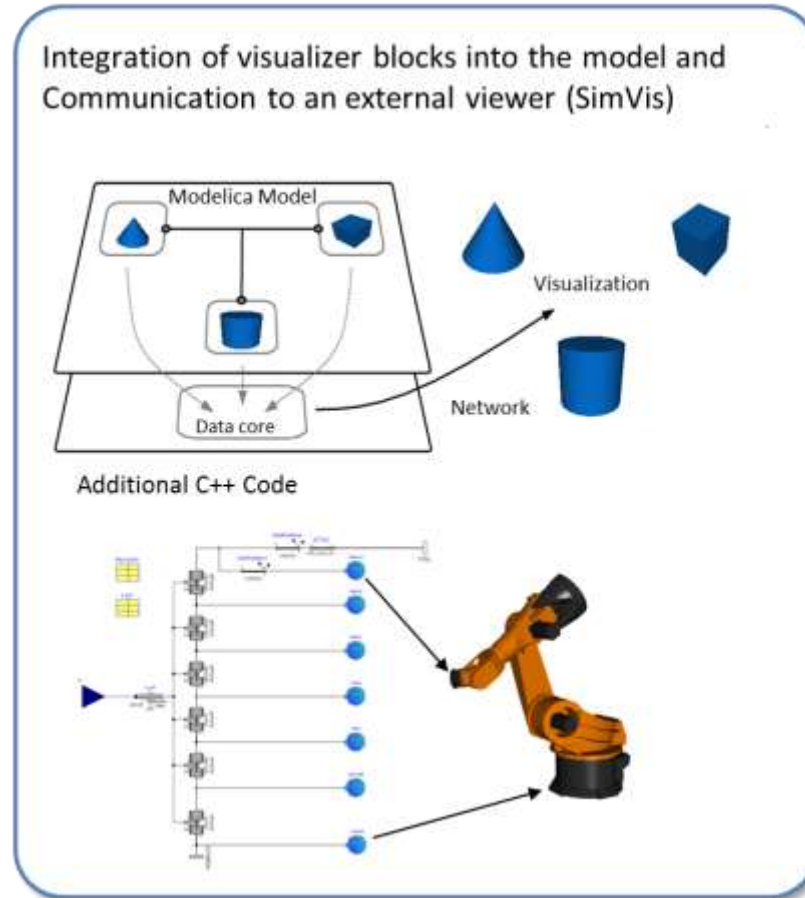


Collision detection in Unity

Courtesy of Volker Waurich - TU Dresden

Visualization using Third-Party Libraries: DLR Visualization Library

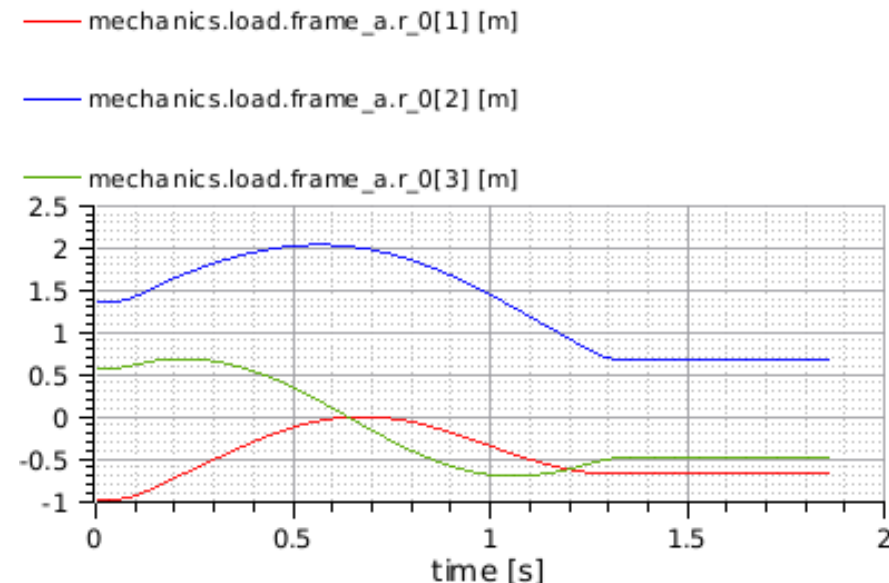
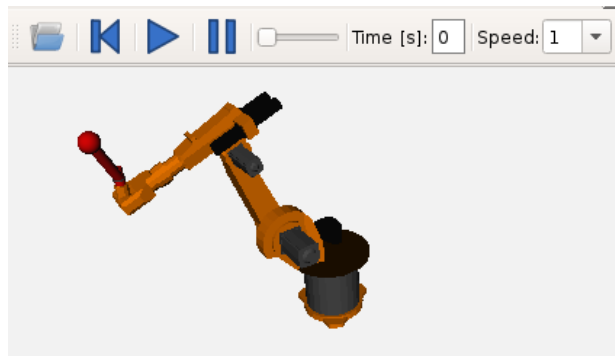
- Advanced, model-integrated and vendor-unspecific visualization tool for Modelica models
- Offline, online and real-time animation
- Video-export function
- Commercial library, feature reduced free Community Edition exists



Courtesy of Dr. Tobias Bellmann (DLR)

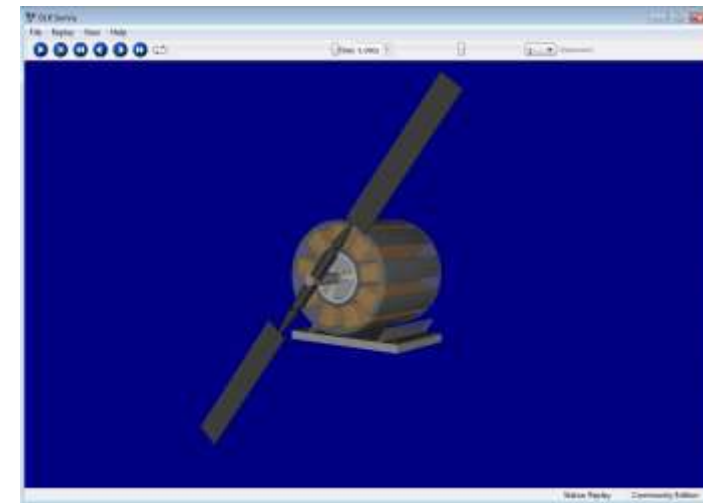
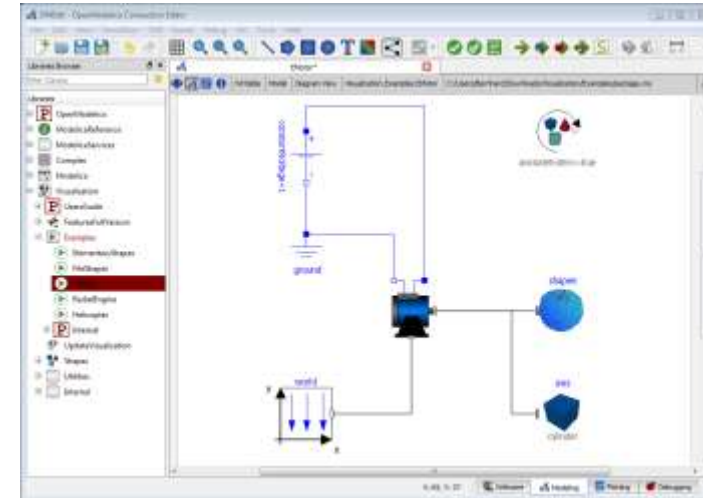
Exercise 1.2: Use 3D Visualization for Robot model

- 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)

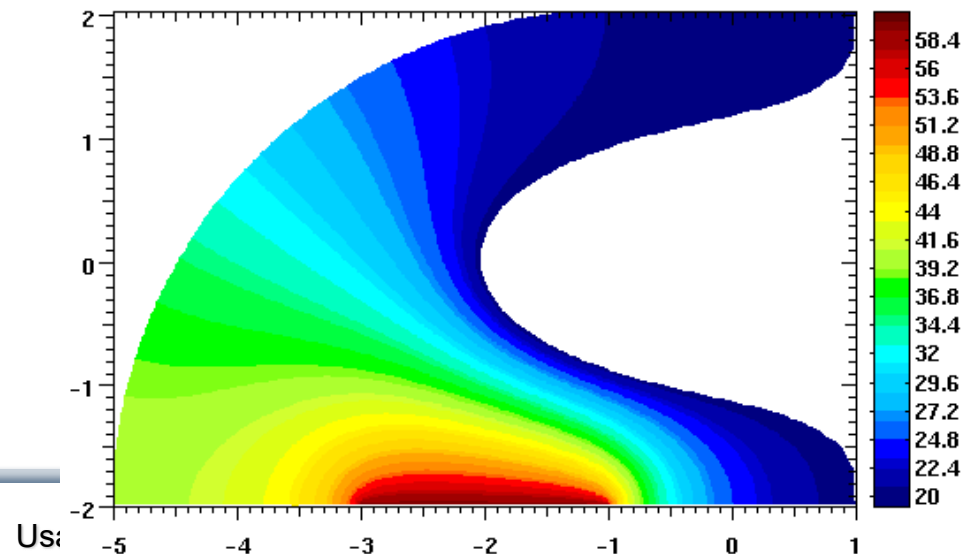
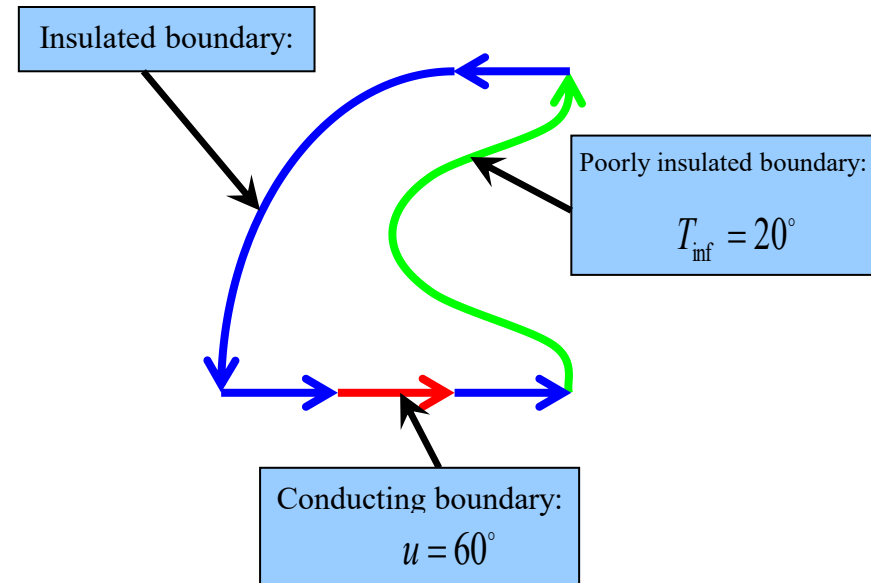
- Unpack VisualizationCommunityEdition.zip
- Open the library in OMEdit
- Simulate the EMotor example
- The DLR SimVis visualization app should start automatically
- Export the animation (File→Export Replay as Video)



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

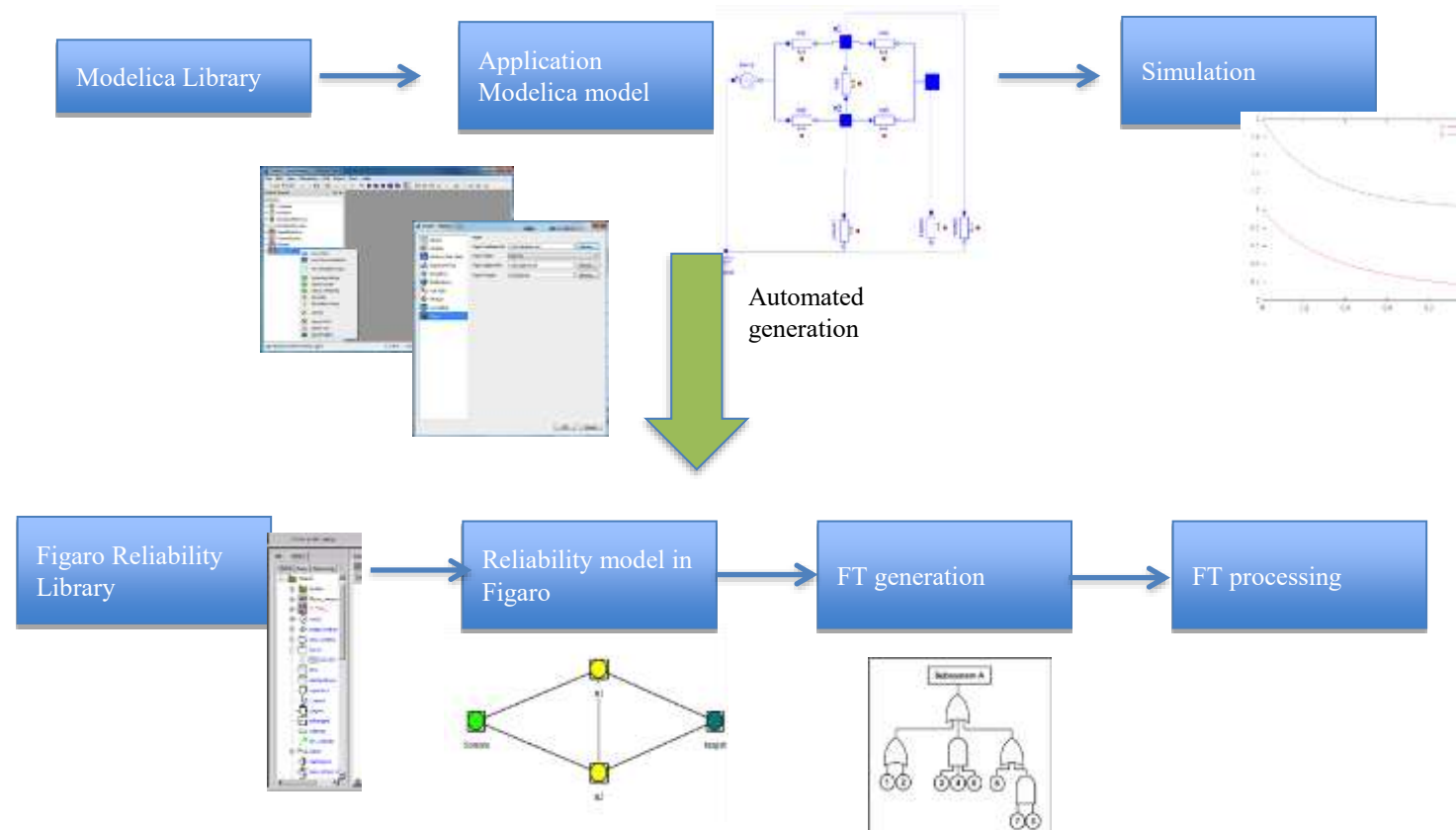
```
model PDEModel
  HeatNeumann h_iso;
  Dirichlet h_heated(g=50);
  HeatRobin h_glass(h_heat=30000);
  HeatTransfer ht;
  Rectangle2D dom;
equation
  dom.eq=ht;
  dom.left.bc=h_glass;
  dom.top.bc=h_iso;
  dom.right.bc=h_iso;
  dom.bottom.bc=h_heated;
end PDEModel;
```

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



Failure Mode and Effects Analysis (FMEA) in OpenModelica

- 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.

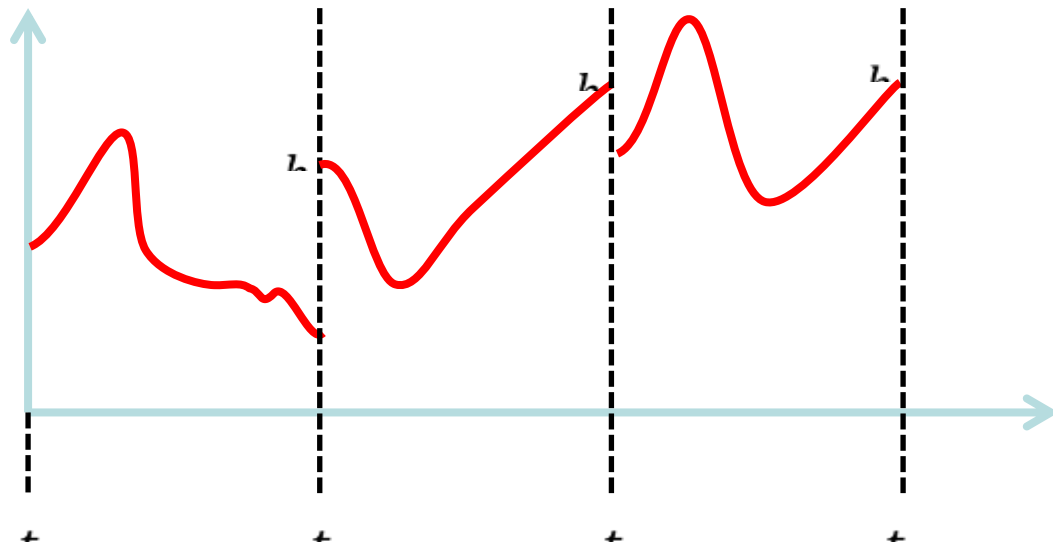


Multiple-Shooting and Collocation

Dynamic Trajectory Optimization

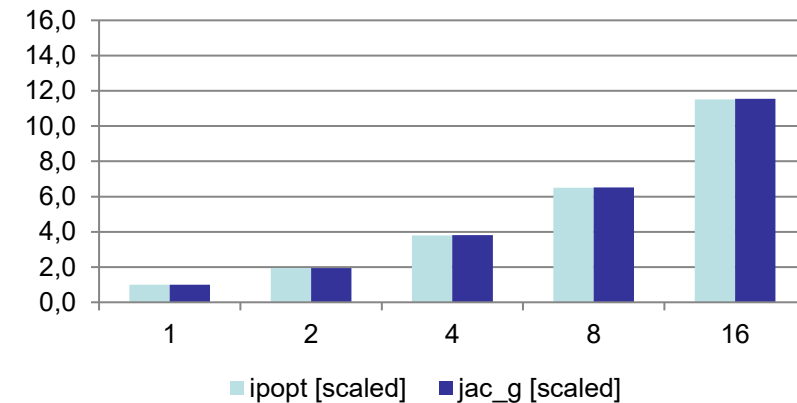
- Minimize a goal function subject to model equation constraints, useful e.g. for NMPC
- Multiple Shooting/Collocation
 - Solve sub-problem in each sub-interval

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

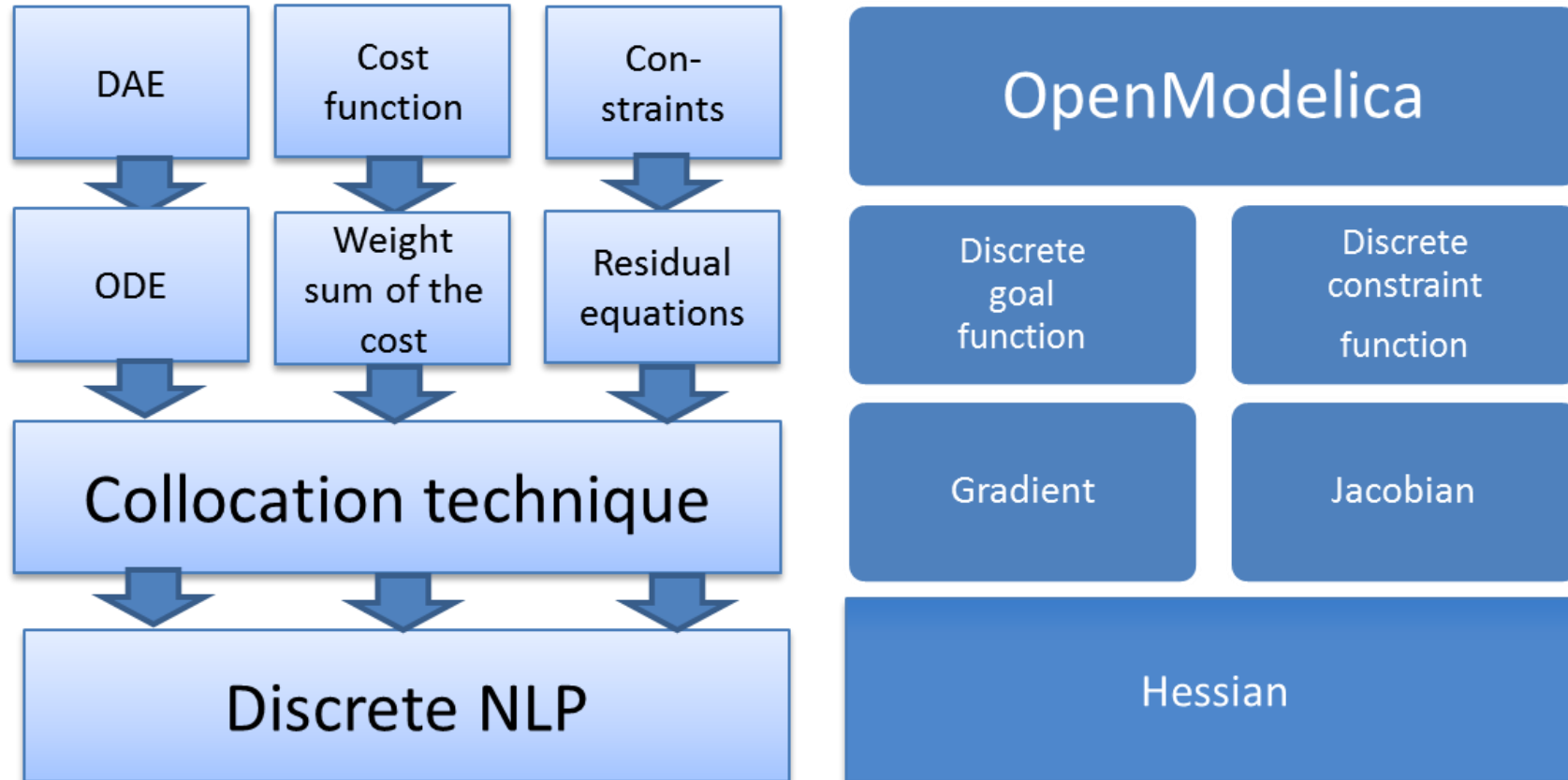


Example speedup, 16 cores:

MULTIPLE_COLLOCATION



OpenModelica Dynamic Optimization Collocation

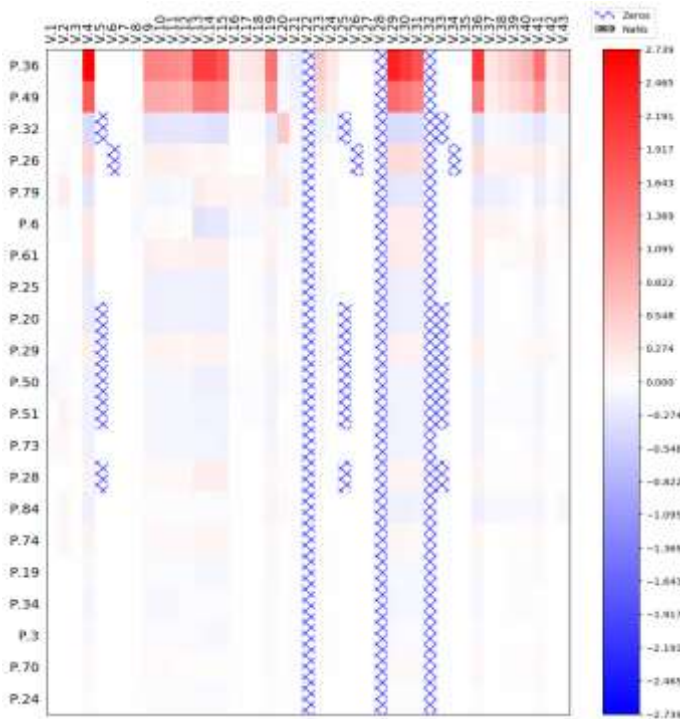


OMSens – Multi-Parameter Sensitivity Analysis

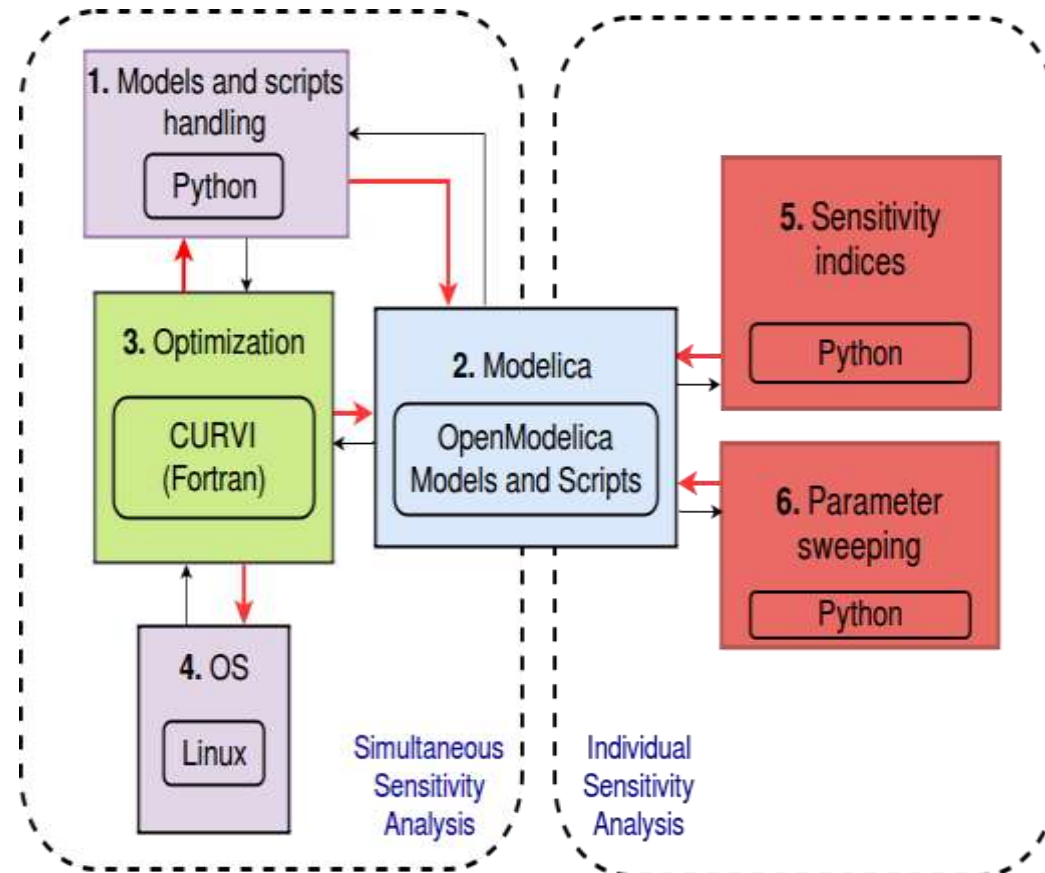
- Individual and simultaneous multi-parameter analysis
- Optimization-based simultaneous analysis
- Robust derivative free optimizer

For an exercise, see further in these slides

Heatmap visualization



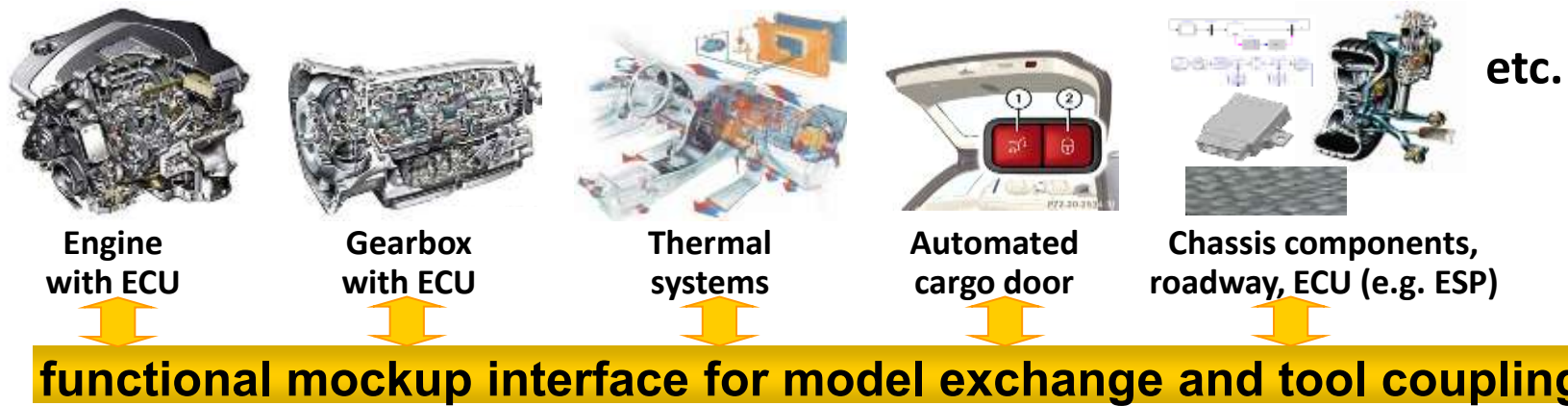
Tool architecture



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)
- **Version 1.0** FMI for Co-Simulation (released Oct 12, 2010)
- **Version 2.0** (released July 25 2014) **2.0.4** (released Dec 1, 2022)
- **Version 3.0** (released May 10 2022)
- **Version 3.01** (released July 10, 2023) **Version 3.02** (released Nov 27, 2024)
- FMI for Model Exchange and Co-Simulation
- ~ **180 tools** supporting it (<https://www.fmi-standard.org/tools>)

FMI 3.0 Main Improvements

- Event Mode for Co-Simulation
- Intermediate Variable Update
- Clocks
- New Types
- Array Variables
- Terminals and Icons
- FMI for Scheduled Execution
- Preparation for Layered Standards

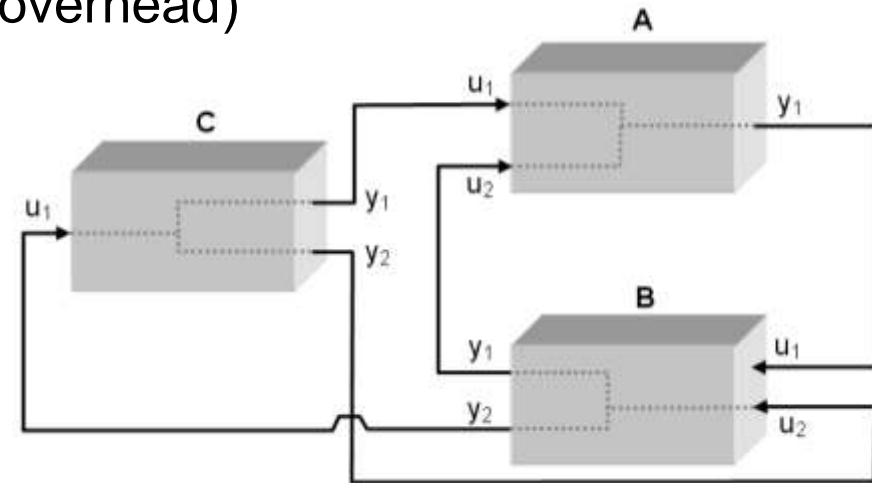
Performance

Accuracy

New Application

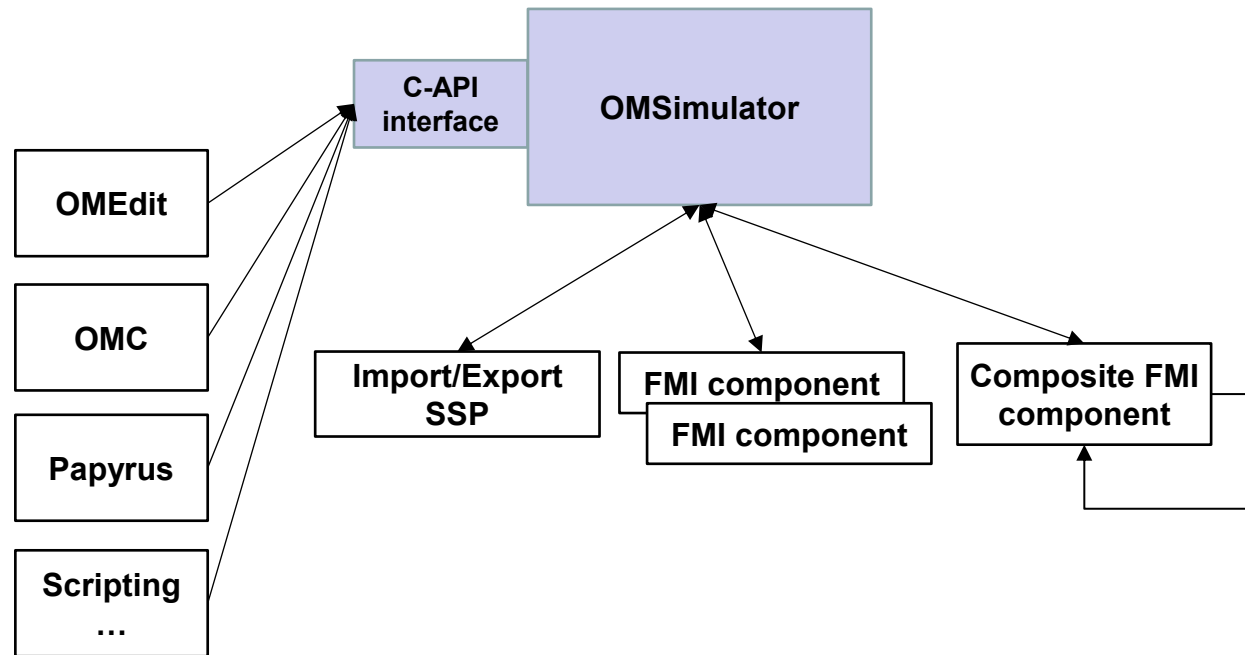
Functional Mockup Units

- 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 – FMI based Co-simulator/Simulator

Part of OpenModelica tool suite



Main Framework Aspects

Unified co-simulation/simulation tool

- FMI 2.0 (model exchange and co-simulation)
- System Structure and Parameterization (SSP)
- Real-time and offline simulation

Standalone open-source simulation tool with rich interfaces

- C/C++
- Scripting languages Python, Lua

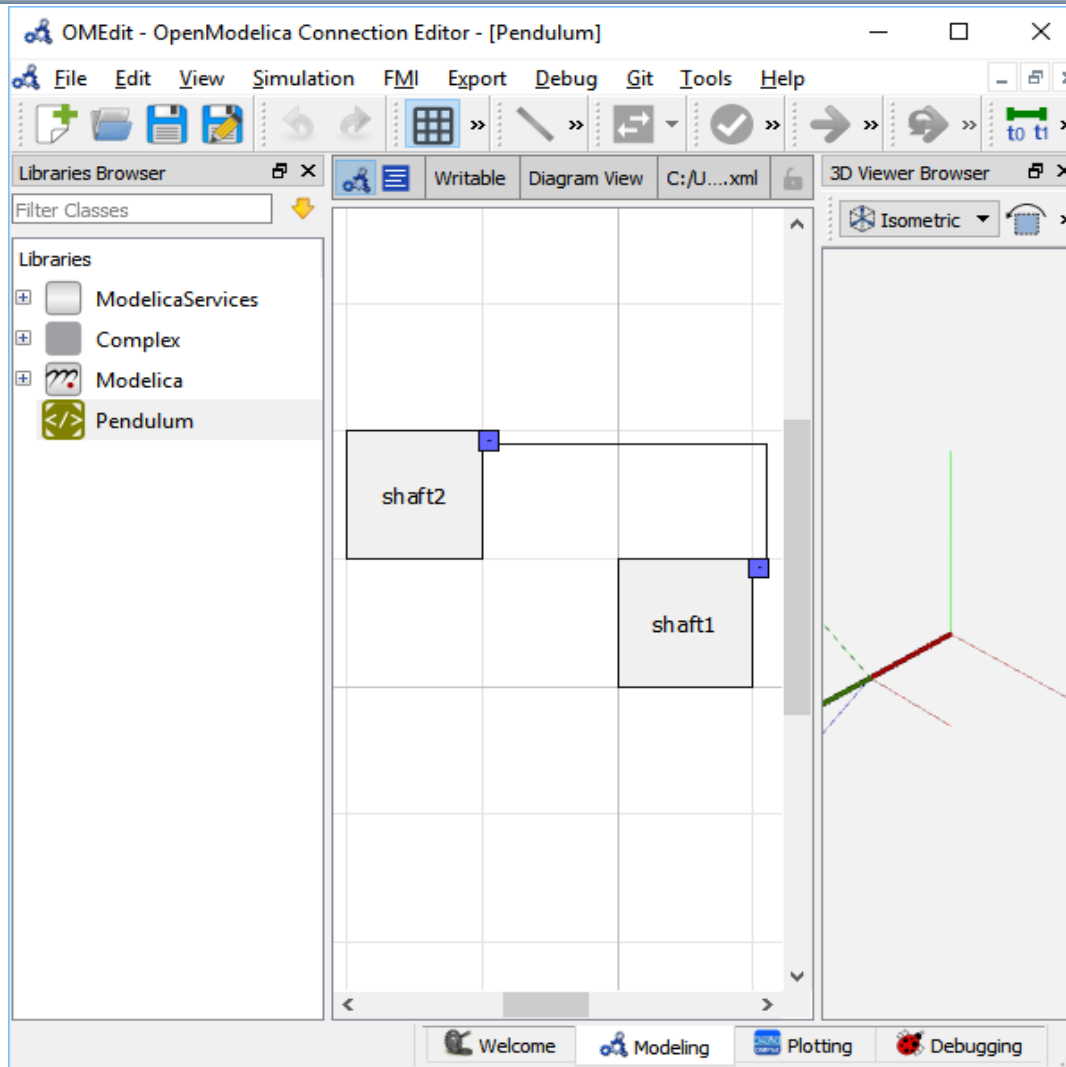
Co-simulation framework as a solid base for engineering tools

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

OMSimulator in OpenModelica 1.25.0

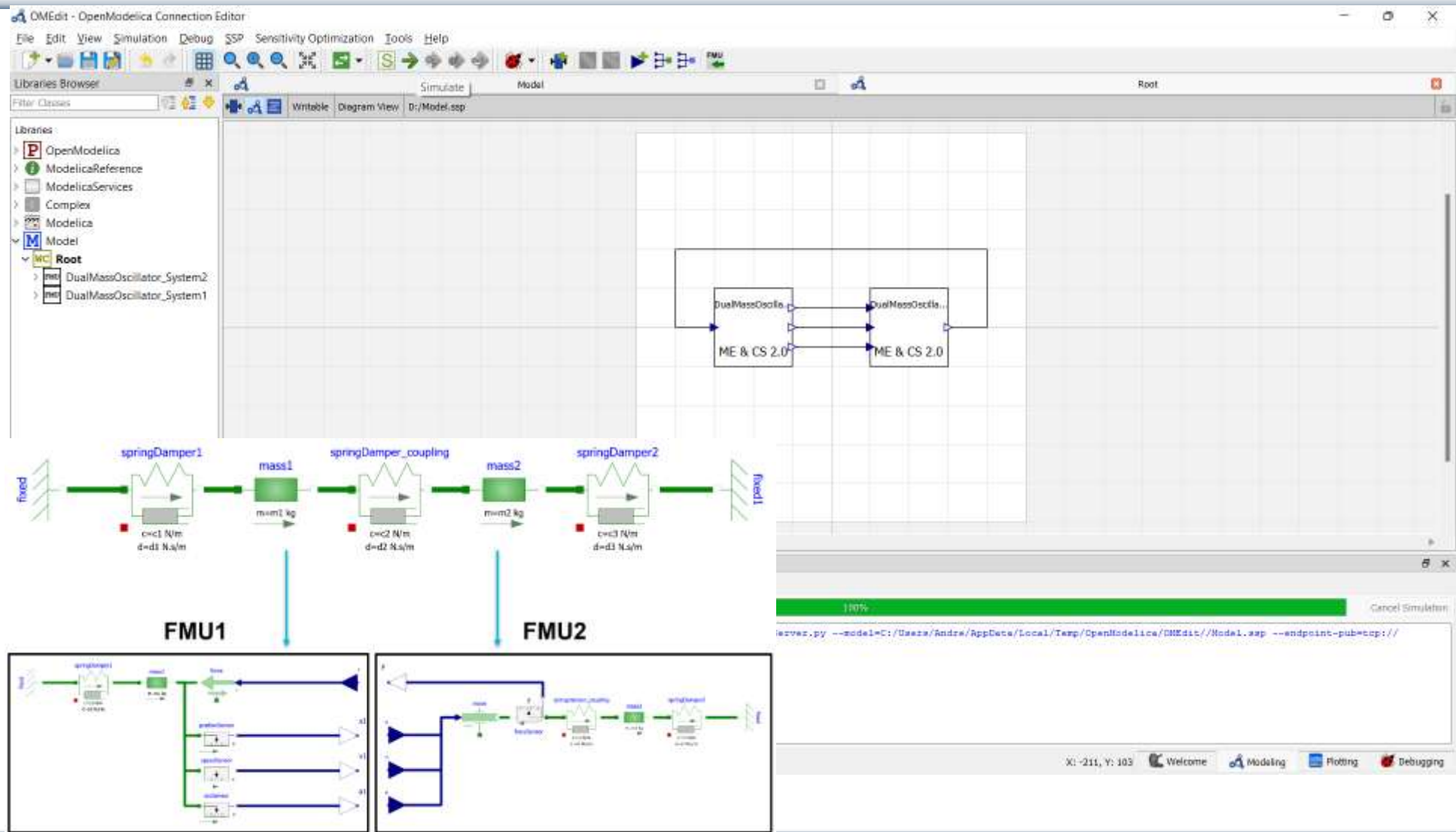
- Supports SSP & FMI standards
- Co-simulation to multiple tools
- Composite model editor
- External API interface and scripting (C, Python, Lua)

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 (in TLM mode)
- Composite model saved as SSP XML-file
- **Support for SSP** – System Structure and Parameterization standard

OMSimulator – GUI and SSP support



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

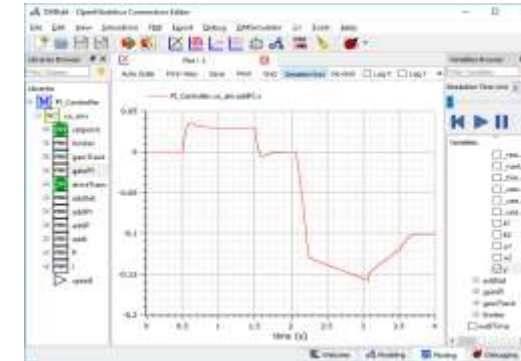
	bus2 inputs	bus1 outputs	ssd:Connection
1	<input checked="" type="checkbox"/> u1	y	<ssd:Con...t="sc2"
2	<input type="checkbox"/> u2		

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

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

OK Cancel

FMI Simulation
results
in OMEdit

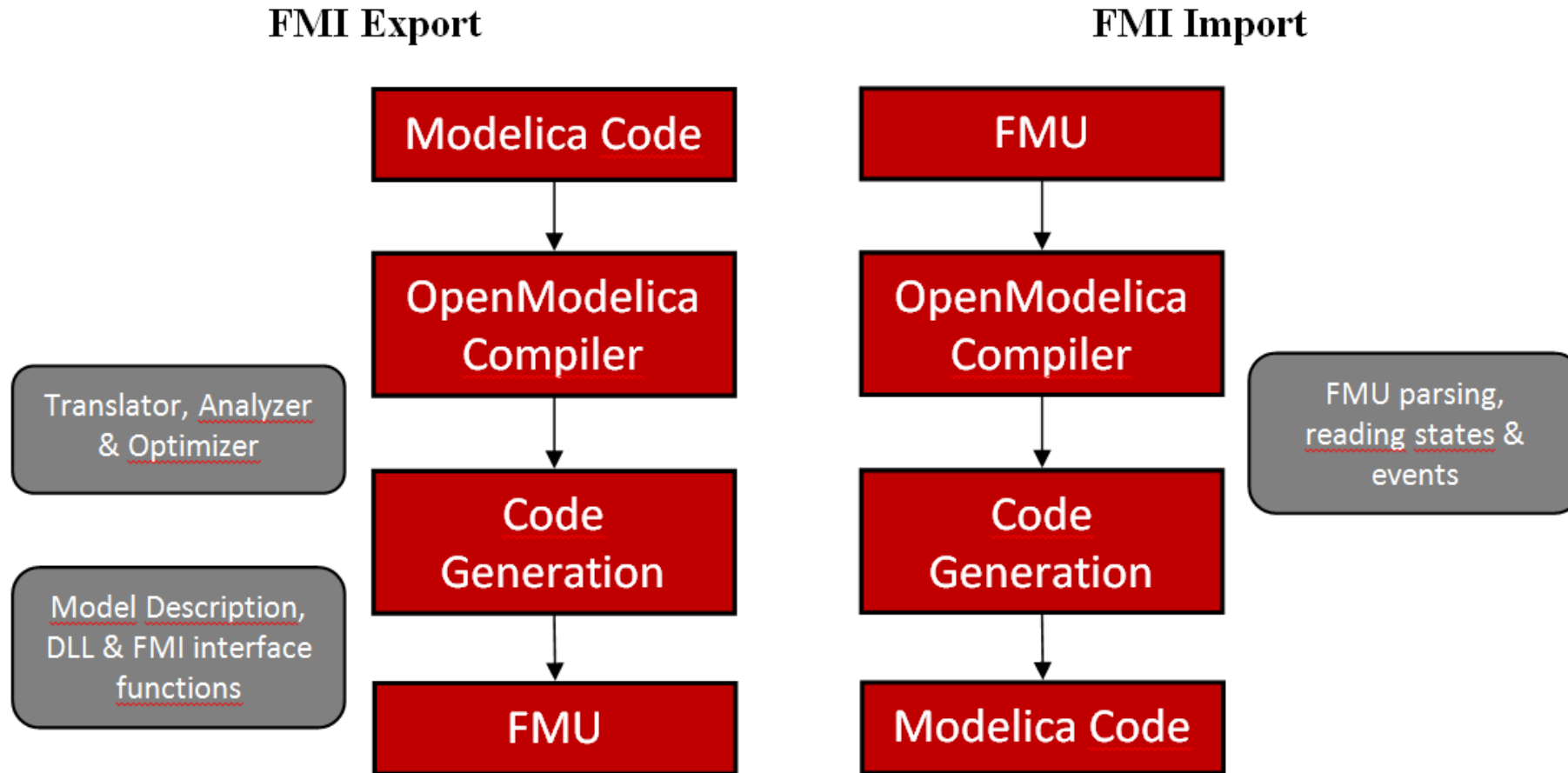


FMI Simulation Tool Comparison

	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

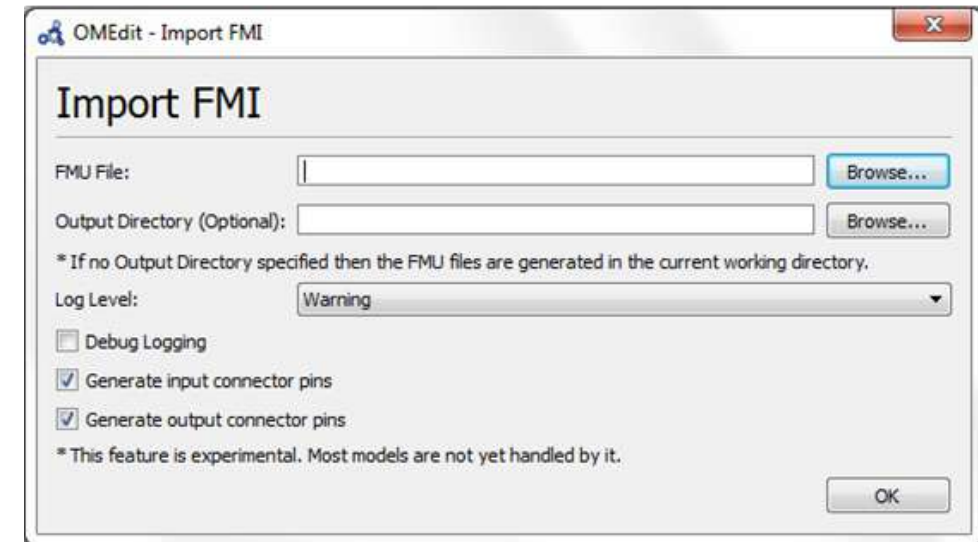
	Dymola	PySimulator	FMI Go!	FMI Composer
Commercial	Yes	No	No	Yes
Open-source	No	BSD	MIT	No
Lookup Table	Yes	Yes	Yes	Yes
Alg. Loops	Yes	Yes	Yes	Yes
Scripting	proprietary	Python	Go	No
GUI	Yes	Yes	No	Yes
SSP	No	No	Yes	Yes
platform	Linux/Win	Linux/Win	Linux/Win/macOS	Linux/Win/macOS

OpenModelica Functional Mockup Interface (FMI)



FMI in OpenModelica

- Model Exchange implemented (FMI 2.0)
- FMI 2.0 Co-simulation implemented
- Subset FMI 3.0 implemented for simulation using OpenModelica OMSimulator. Ongoing full implementation.
- 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)	68 kB flash memory 25 kB variables (RAM)	4090 B flash memory 151 B variables (RAM)

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

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

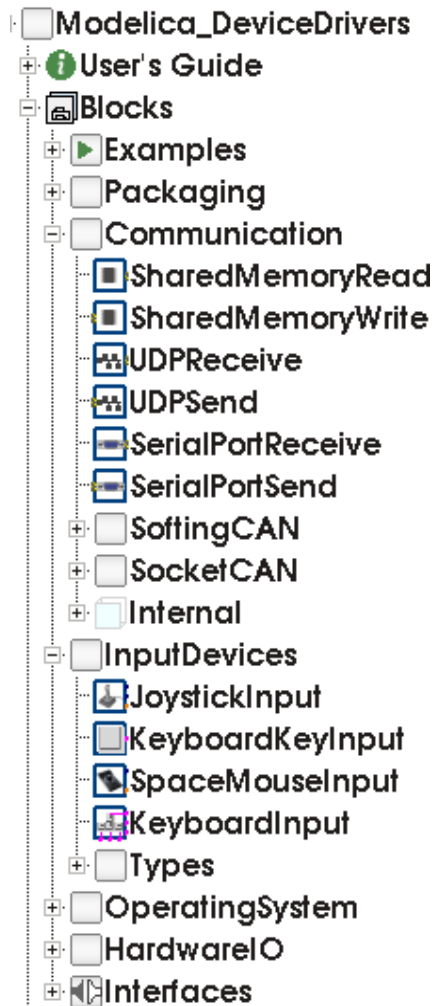
The ATmega16 we target has **1 kB SRAM available** (stack, heap, and global variables)

The Simple Code Generator

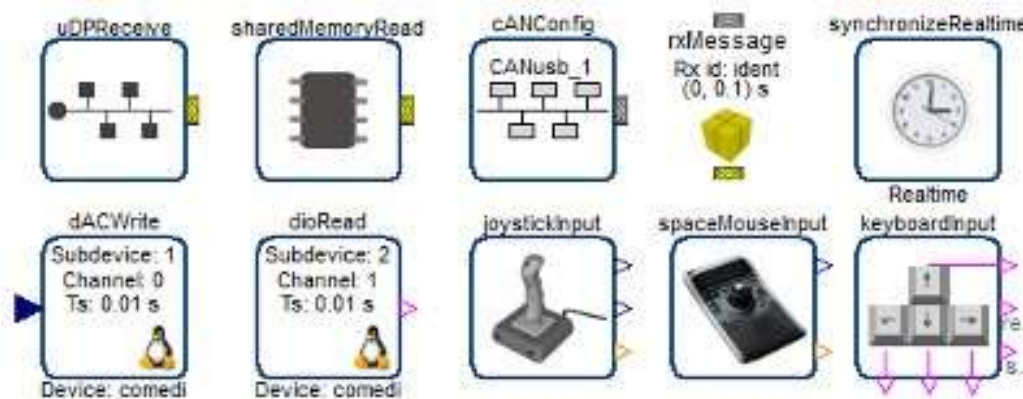
Supports only a limited Modelica subset

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

Communication & I/O Devices: MODELICA_DEVICEDRIVERS Library



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

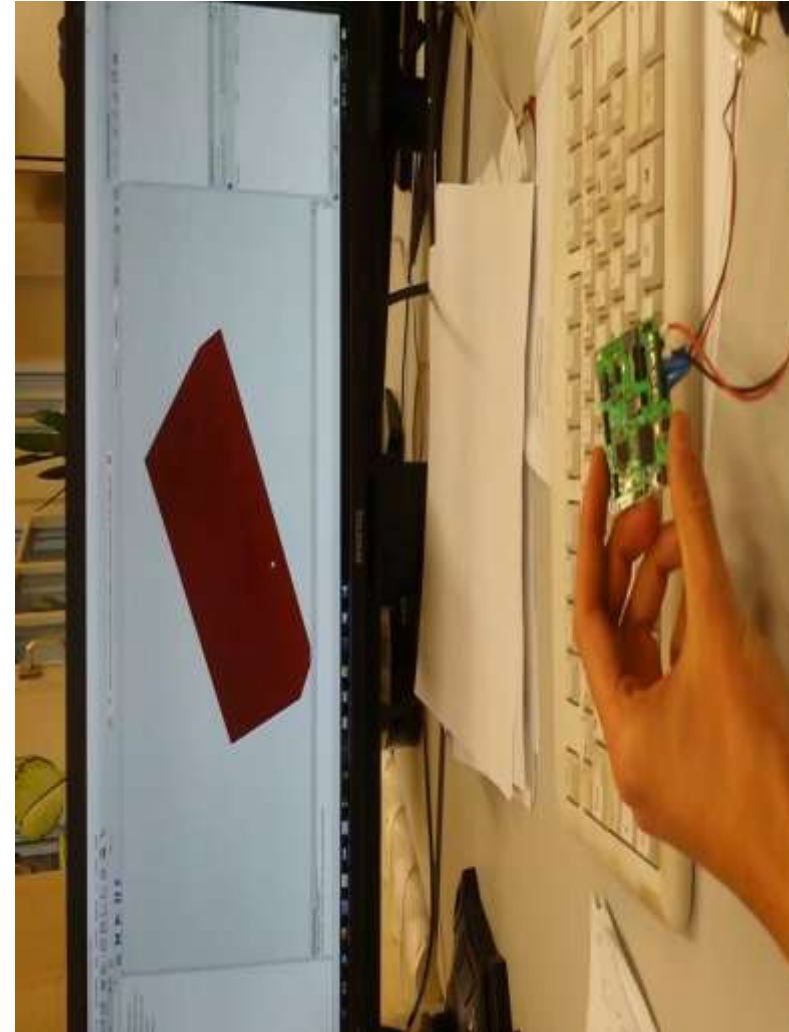


https://github.com/modelica/Modelica_DeviceDrivers/

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

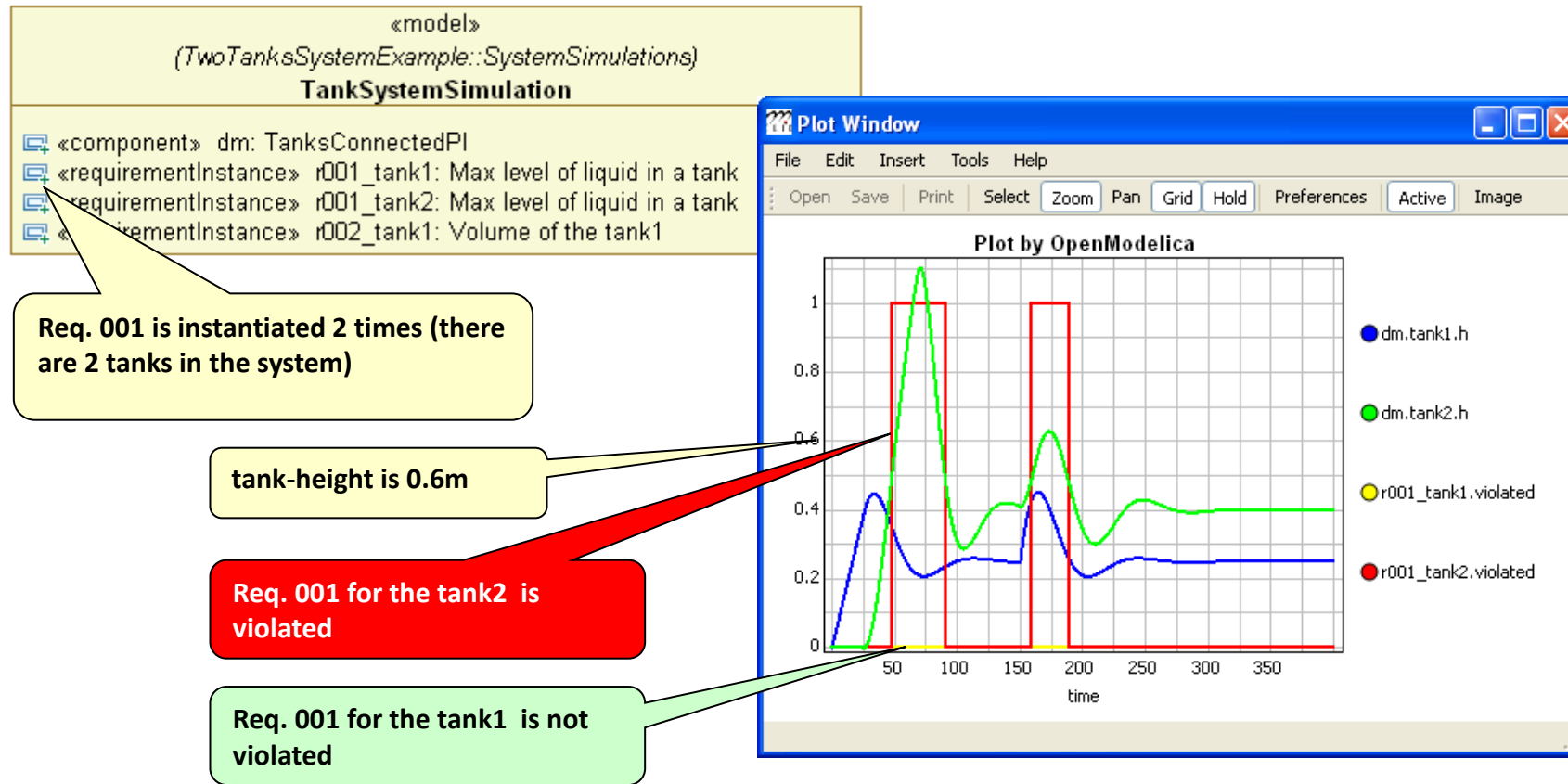


OpenModelica – ModelicaML UML Profile

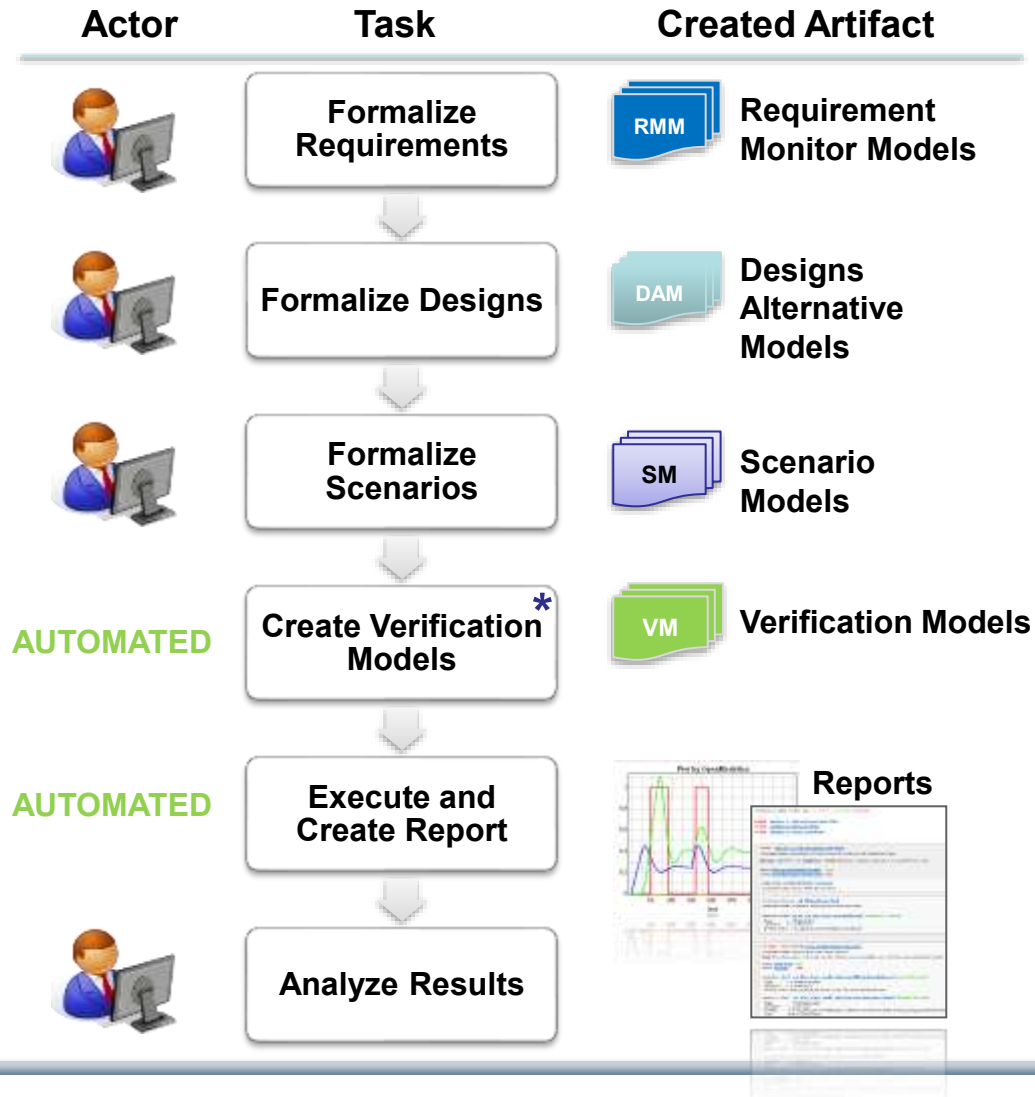
SysML/UML to Modelica OMG Standardization

- 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



vVDR Method – virtual Verification of Designs vs Requirements



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

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

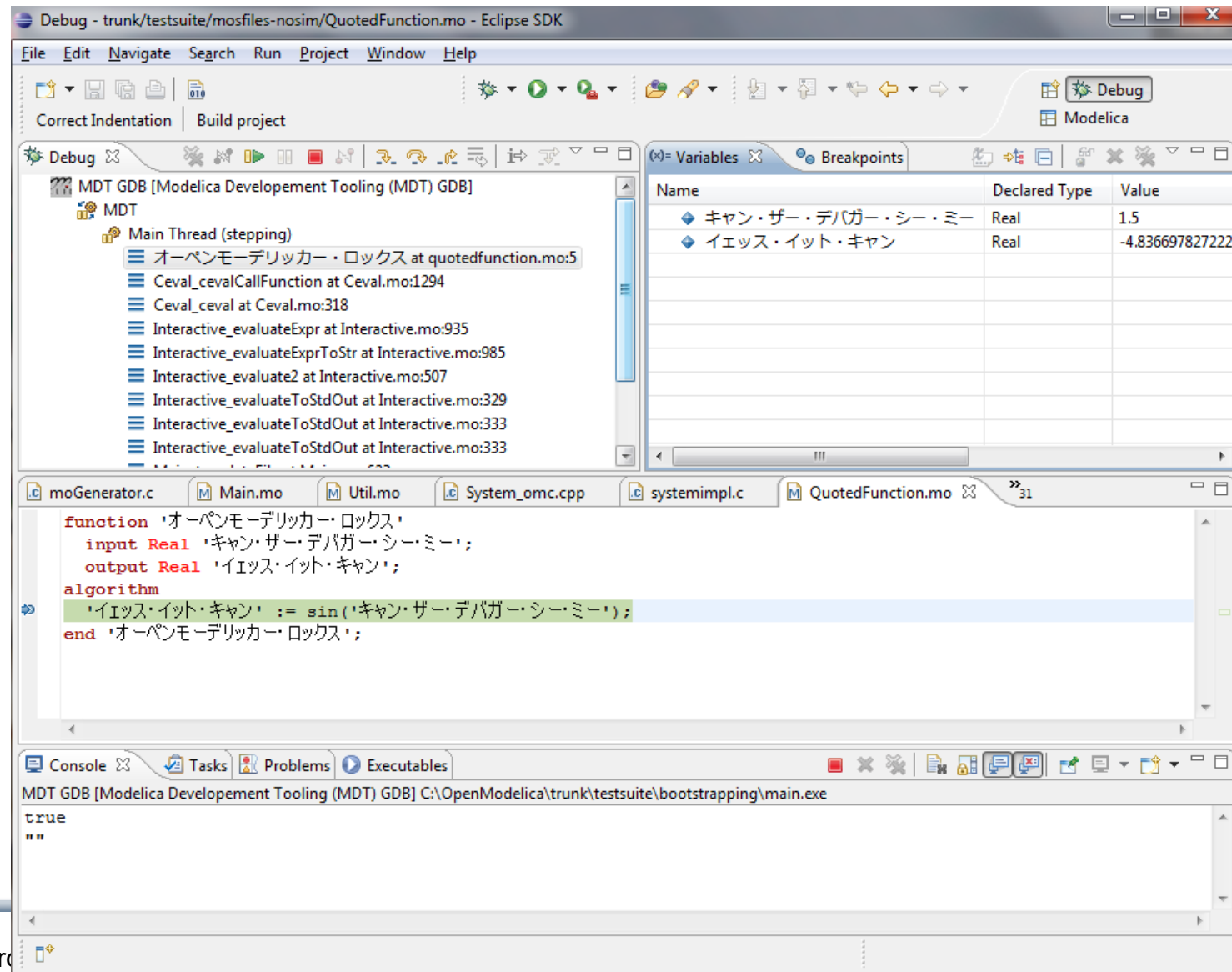
The screenshot displays the Eclipse IDE interface for the OpenModelica MDT Algorithmic Code Debugger. The main window is titled "Debug - HelloWorld/SimulationModel.mo - Eclipse SDK". The interface is divided into several panes:

- 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 takes a "Real inValue" and returns a "Real outValue" by calculating "inValue * 2". The current line of execution is highlighted at line 20: "outValue := inValue * 2;".
- Outline View:** Located in the bottom-right pane, it shows the project structure. The "getValueMultipliedByTwo" function is expanded, showing its inputs "inValue (Real - IN)" and "outValue (Real - OUT)". The "SimulationModel" is also expanded, showing its variables "x" and "y".
- Output View:** Located in the bottom-left pane, it displays 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



OpenModelica Equation Model Debugger

The screenshot displays the OMEdit - Transformational Debugger window. It features three main panels: Variables View, Equations View, and Source View.

Variables View: Shows a tree structure of variables. The 'Variables' section lists variables like 'frame', 'boxBody1', 'body', 'frame_a', 'R', and 'T'. The 'Defined In Equations' and 'Used In Equations' sections show the equations where these variables are defined and used.

Equations View: Shows a list of equations with their indices and types. The 'Equations Browser' lists equations from 819 to 829. The 'Defines' and 'Depends' sections show the variables defined and depended upon by each equation.

Source View: Shows the source code of the model. The code is color-coded and includes comments. The 'Source Browser' shows the file path: C:/OpenModelica/trunk/build/lin.../Mechanics/MultiBody/Joints.mo.

Showing equation transformations of a model:

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

(1) **substitution:**

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

(2) **simplify:**

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

(3) **expand derivative (symbolic diff):**

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

(4) **solve:**

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

Mapping run-time error to source model position

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

Activities OMEdit Tue 12:06 sv Martin Sjölund

OMEdit - Transformational Debugger

/tmp/OpenModelica_marsj/OMEdit/Modelica.Mechanics.MultiBody.Examples.Loops.EngineV6_info.xml

Variables

Variables Browser

Case Sensitive Regular Expression

Expand All Collapse All

Variables	Comment	Line	Location
phi	Exter...phi	6616	/usr/li...onal.mo
phi	Relat...ame_b	260	/usr/li...ints.mo
phi_offset	Relat...+ phi	242	/usr/li...ints.mo
Crank1	Absol...frame	11	/usr/li...mes.mo
body	Trans...frame	10	/usr/li...mes.mo
phi	Dumm...body	805	/usr/li...arts.mo
phi[1]	Dumm...body	805	/usr/li...arts.mo
phi[2]	Dumm...body	805	/usr/li...arts.mo
phi[3]	Dumm...body	805	/usr/li...arts.mo
phi_d	= der(phi)	809	/usr/li...arts.mo
phi_d[1]	= der(phi)	809	/usr/li...arts.mo
phi_d[2]	= der(phi)	809	/usr/li...arts.mo

Defined In Equations

Index	Type	Equation
587	initial	(nonlinear)
5016	regular	(nonlinear)

Used In Equations

Inc	Type	Equation
...	regular	(assignment) cylinder...cos(cylinder3.B2.phi)
...	regular	(assignment) cylinder...sin(cylinder3.B2.phi)
...	regular	(assignment) cylinder...sin(cylinder3.B2.phi)
...	regular	(assignment) cylinder...cos(cylinder3.B2.phi)
...	regular	(assignment) der(cyl...der3.Rod.body.w_a[1])
...	regular	(assignment) der(cyl...der3.Rod.body.w_a[1])
...	regular	(assignment) der(cyl...der3.Rod.body.w_a[1])
...	regular	(assignment) der(cyl...der3.Rod.body.w_a[1])

Variable Operations

Operations

Equations

Equations Browser

Inc	Type	Equation
...	regular	(assignment) cylind...ylinder3.Cylinders
...	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...linder.s else 1e-06
...	regular	(assignment) cylind...k2.frame_b.R.T[2,3]
...	regular	(linear,frame_b.r_0 = 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])

Defines

Variable

der(cylinder3.B2.R_rel.T[3,3])

Equation Operations

Operations

Equation

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:

Depends

Variable

cylinder3.B2.phi

cylinder3.Rod.body.w_a[1]

Source Browser

/usr/lib/omlibrary/Modelica 3.2.1/Mechanics/MultiBody/Jo

```
306 Connections.branch(frame_a.R,
307 frame_b.R);
308
309 assert(cardinality(frame_a) > 0,
310 "Connector frame_a of revolute
311 joint is not connected");
312
313 assert(cardinality(frame_b) > 0,
314 "Connector frame_b of revolute
315 joint is not connected");
316
317 angle = phi_offset + phi;
318 w = der(phi);
319 a = der(w);
320
321 // relationships between quantities
322 of frame_a and of frame_b
323 frame_b.r_0 = frame_a.r_0;
324
325 if rooted(frame_a.R) then
326   R_rel = Frames.planarRotation(e,
327 phi_offset + phi, w);
328   frame_b.R =
329 Frames.absoluteRotation(frame_a.R,
330 R_rel);
331   frame_a.f = -
332 Frames.resolve(R_rel, frame_b.f);
333   frame_a.t = -
334 Frames.resolve(R_rel, frame_b.t);
335 else
336   R_rel = Frames.planarRotation(-e,
337 phi_offset + phi, w);
338   frame_a.R =
339 Frames.absoluteRotation(frame_b.R,
340 R_rel);
341   frame_b.f = -
342 Frames.resolve(R_rel, frame_a.f);
343   frame_b.t = -
344 Frames.resolve(R_rel, frame_a.t);
345 end if;
346
347 // d'Alemberts principle
348 tau = -frame_b.t*e;
349
350 // Connection to internal
```

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

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

The Source Browser shows the following code snippet:

```
equation
y = u1/u2;
annotation {
  Documentation {info="Chml"
    <p>
      This block computes the output <b>y</b> (element-wise)
      by <b>dividing</b> the corresponding elements of
      the two inputs <b>u1</b> and <b>u2</b>.
    }
  </pre>
  y = u1 / u2;
  </pre>
```


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

The screenshot shows the OMEdit - Transformational Debugger interface. The main window displays the source code of a model. The 'Equations' browser on the left lists several equations, with equation 5 highlighted. An arrow points from equation 5 in the browser to the corresponding line in the source code.

Equations Browser:

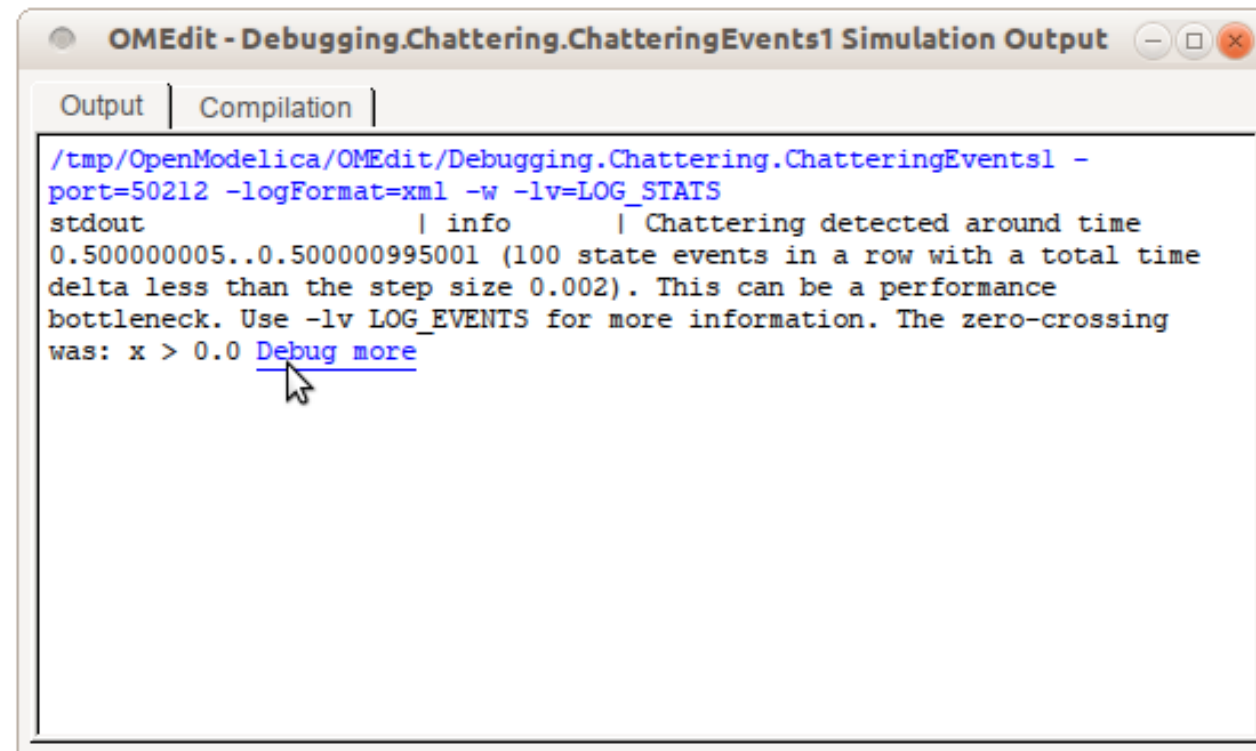
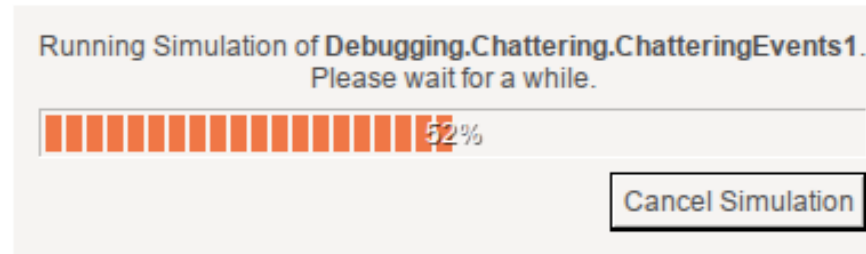
Inc	Type	Equation
1	initial	(assignment) $x = 1.0$
2	initial	(assignment) $x = 0$ else 1.0
3	initial	(assignment) $y = 2.0 * z$
4	initial	(assignment) $\text{der}(x) = y$
5	regular	(assignment) $x = 0$ else 1.0
6	regular	(assignment) $y = 2.0 * z$
7	regular	(assignment) $\text{der}(x) = y$

Source Code:

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

equation
 $z = \text{if } x > 0 \text{ then } -1 \text{ else } 1;$
 $y = 2 * z;$

Error Indication – Simulation Slows Down



Performance Profiling for Faster Simulation

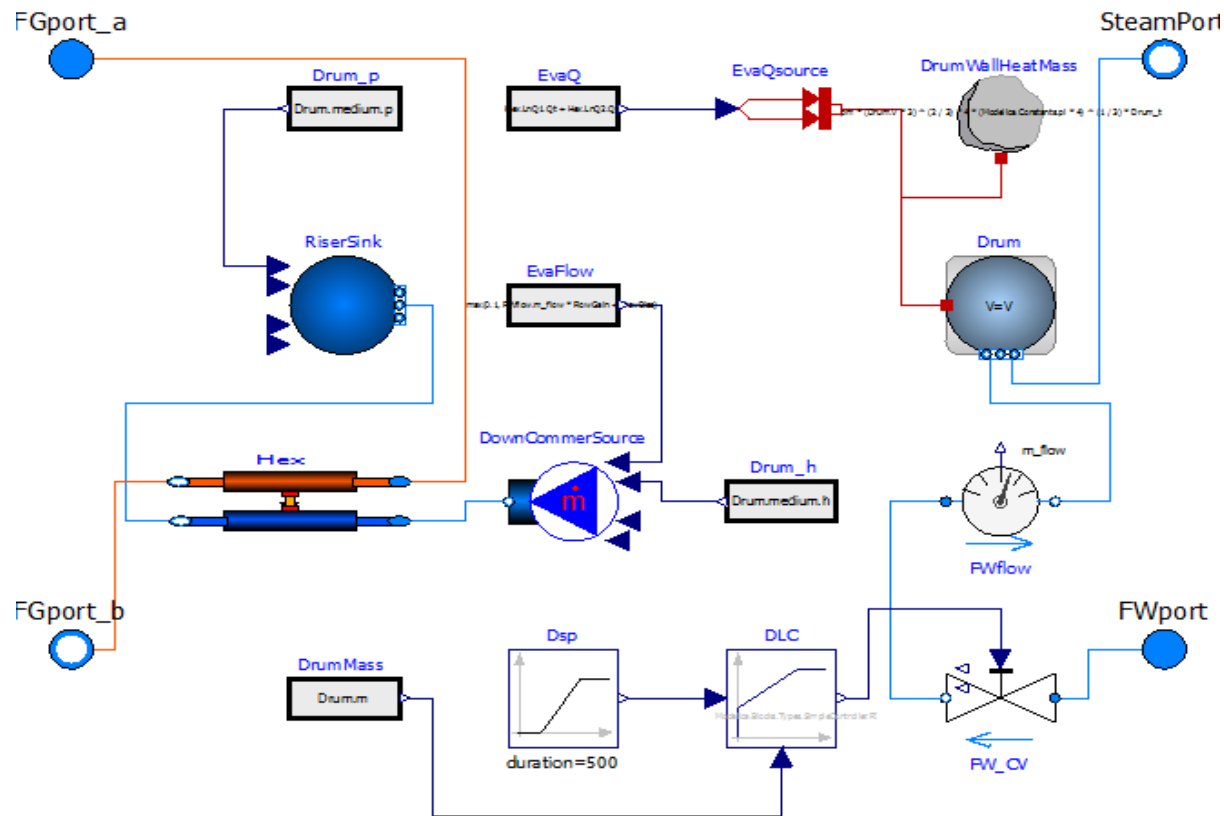
(Here: Profiling all equations in MSL 3.2.1 DoublePendulum)

- 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

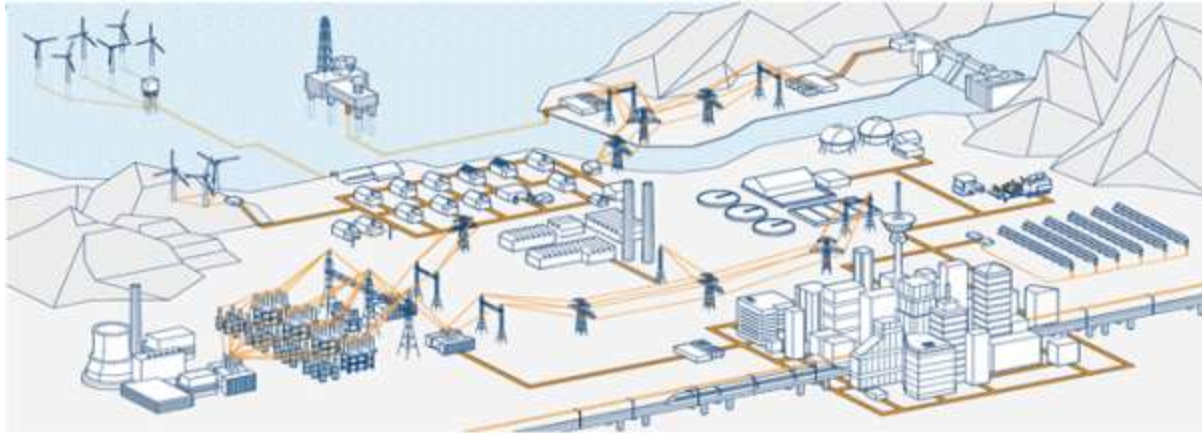


Conclusion from the evaluation:

“...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



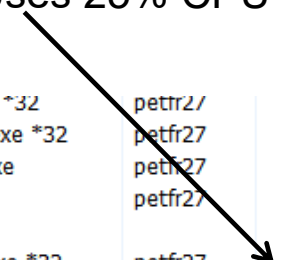
- ABB: *“ABB uses several compatible Modelica tools, including OpenModelica, depending on specific application needs.”*
- ABB: *“OpenModelica provides outstanding debugging features that help to save a lot of time during model development.”*

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

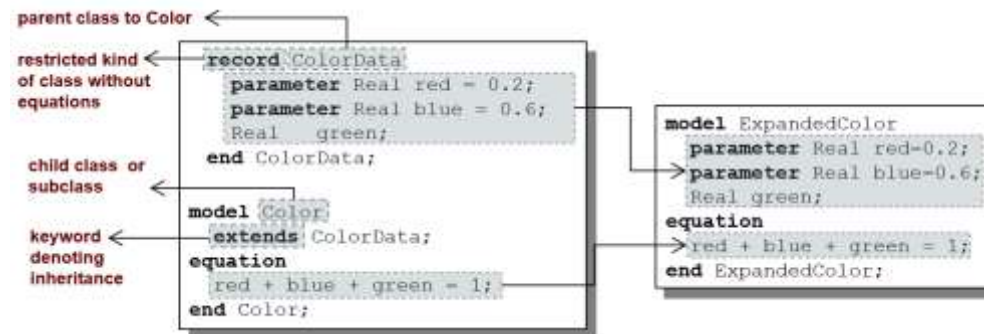


acrotray.exe *32	pettr2/	00	976 K	Acro Iray
AdobeARM.exe *32	petfr27	00	1,136 K	Adobe Reader an...
Bootcamp.exe	petfr27	00	1,448 K	Boot Camp Manager
conhost.exe	petfr27	00	1,300 K	Console Window ...
csrss.exe		00	3,000 K	
DCSHelper.exe *32	petfr27	00	660 K	DataCardMonitor
Debugging.Chattering....	petfr27	25	1,436 K	Debugging.Chatte...
dllhost.exe	petfr27	00	2,224 K	COM Surrogate

- 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

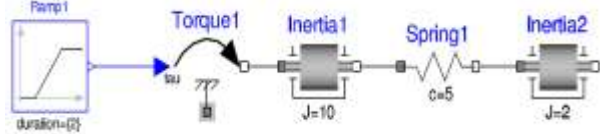
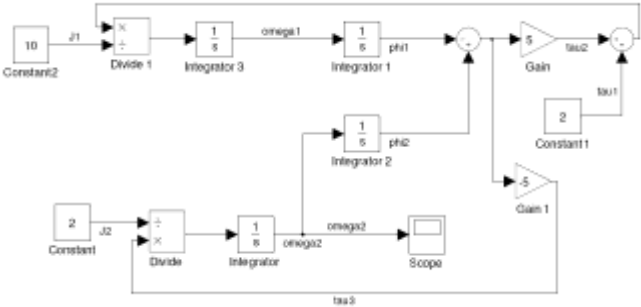


Typed
Declarative
Equation-based
Textual Language

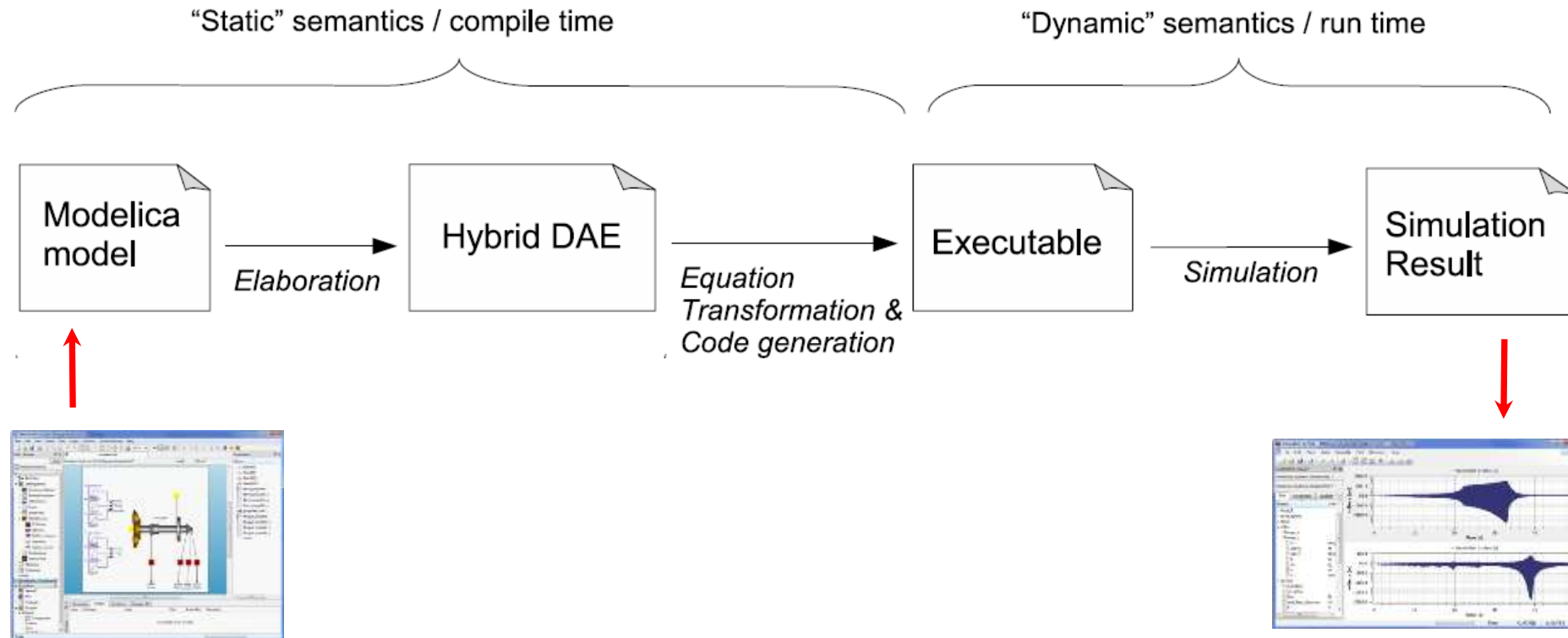
Hybrid
Modeling

Acausal Modeling

The order of computations is not decided at modeling time

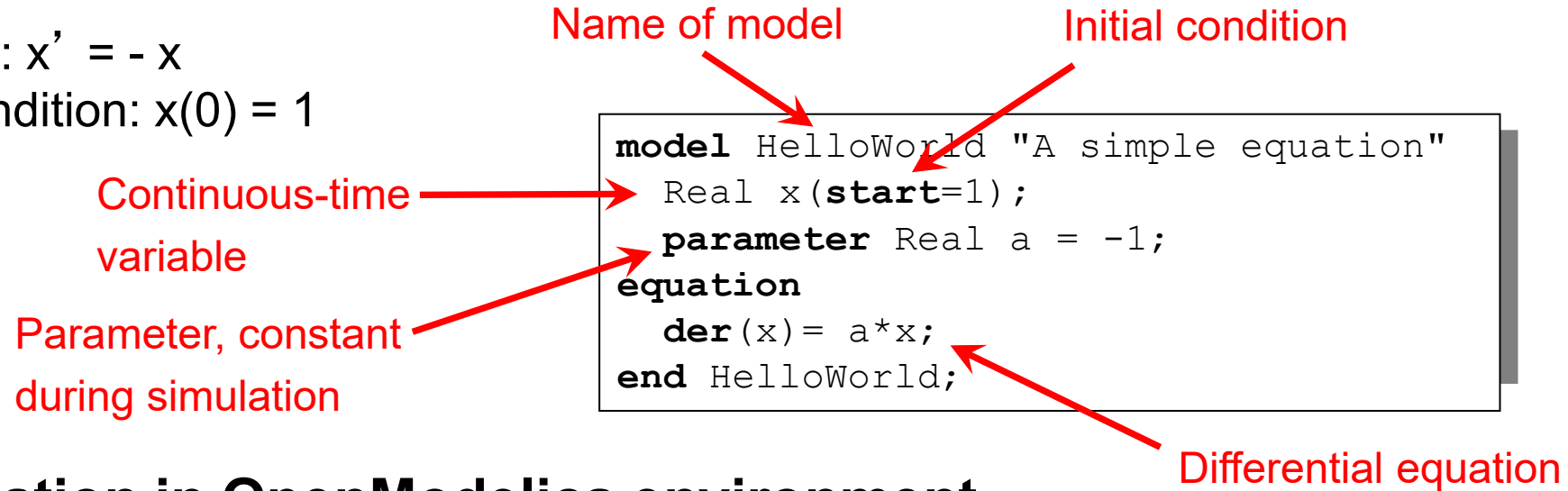
	Acausal	Causal
Visual Component Level	 <p>The diagram shows a mechanical system with three components in series: a torque source 'Torque1' (represented by a blue arrow and a box with a ramp graph), an inertia 'Inertia1' (J=10), a spring 'Spring1' (c=5), and another inertia 'Inertia2' (J=2). The components are connected by lines representing mechanical links. The torque source is labeled 'Pump1' and 'duration=2'.</p>	 <p>The diagram shows a causal mechanical system with two parallel paths. The top path starts with a constant 'Constant2' (10) connected to a 'Divide' block (2/1), followed by an 'Integrator 3' (1/s), then an 'Integrator 1' (1/s), and finally a 'Gain' block (5). The bottom path starts with a constant 'Constant' (2) connected to a 'Divide' block (1/2), followed by an 'Integrator 2' (1/s), and finally a 'Gain' block (-5). The outputs of these paths are connected to a summing junction. The system also includes a 'Scope' block and a 'Constant1' (2) block.</p>
Equation Level	<p>A resistor <i>equation</i>:</p> $R \cdot i = v;$	<p>Causal possibilities:</p> $i := v/R;$ $v := R \cdot i;$ $R := v/i;$

Typical Simulation Process

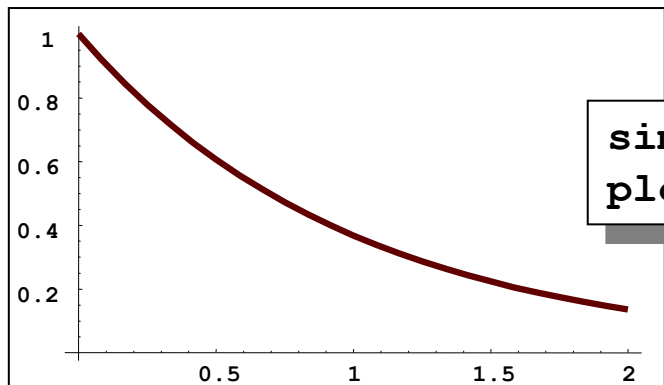


Simple model - Hello World!

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



Simulation in OpenModelica environment



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

Modelica Variables and Constants

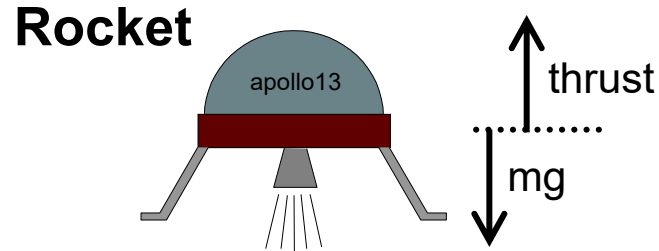
- Built-in primitive data types

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

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

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

A Simple Rocket Model

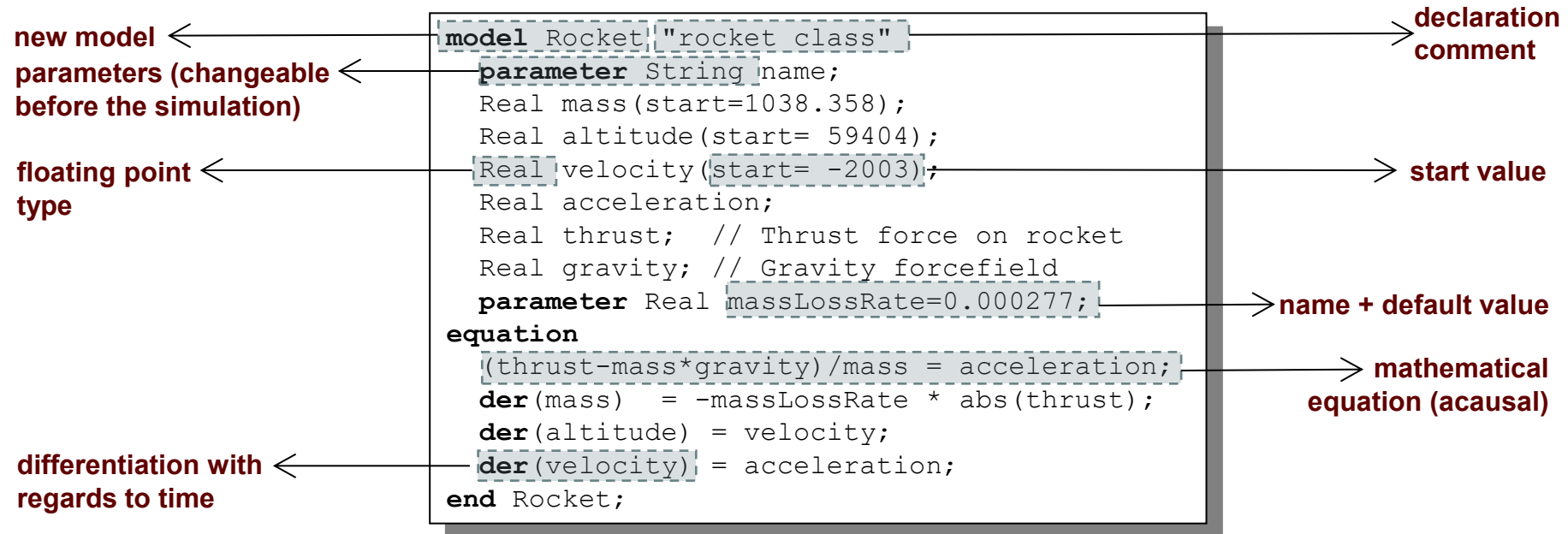


$$acceleration = \frac{thrust - mass \cdot gravity}{mass}$$

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

$$altitude' = velocity$$

$$velocity' = acceleration$$



Celestial Body Class

A class/model 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;
```

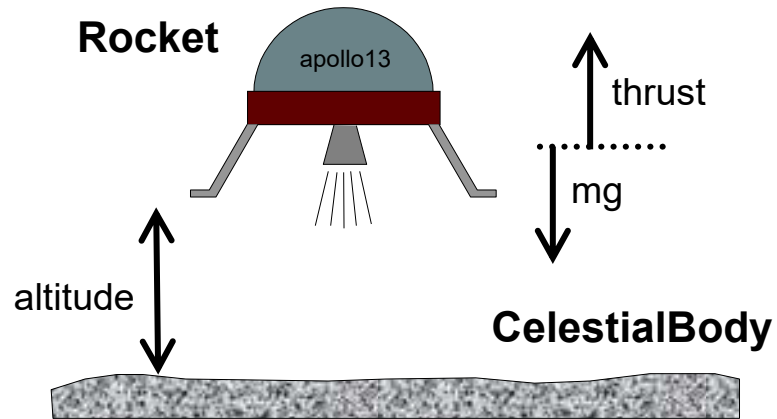


An *instance* of the class can be declared by *prefixing* the type name to a variable name

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

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

Moon Landing



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

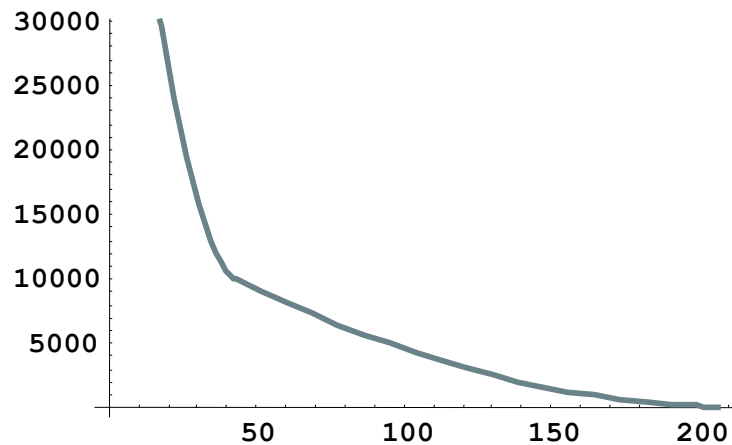
only access
inside the class

access by dot
notation outside
the class

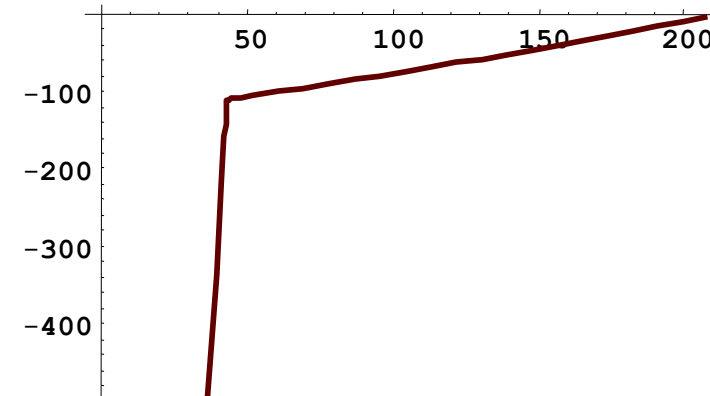
```
model MoonLanding
  parameter Real force1 = 36350;
  parameter Real force2 = 1308;
  protected
    parameter Real thrustEndTime = 210;
    parameter Real thrustDecreaseTime = 43.2;
  public
    Rocket apollo(name="apollo13");
    CelestialBody moon(name="moon", mass=7.382e22, radius=1.738e6);
  equation
    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
- In Modelica 3.0 to 3.4 the `class` keyword means the same as `model`
- After Modelica 3.5 `class` cannot have variables and equation sections, i.e. not 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:

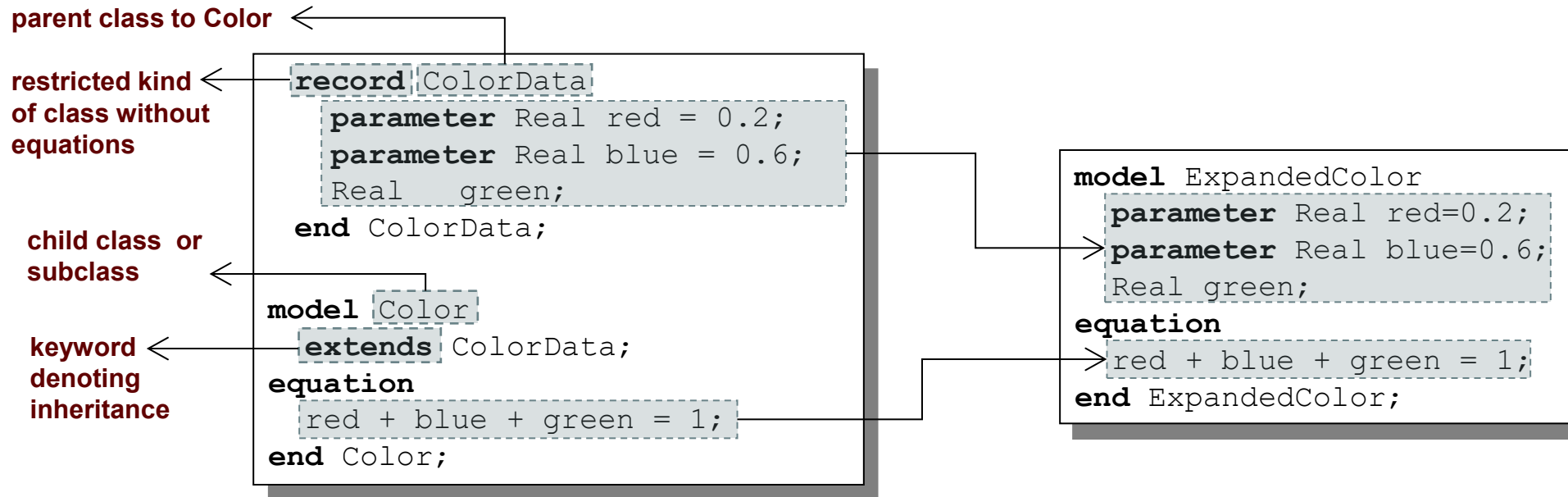
```
...  
p = polynomialEvaluator({1,2,3,4},21)
```

{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.

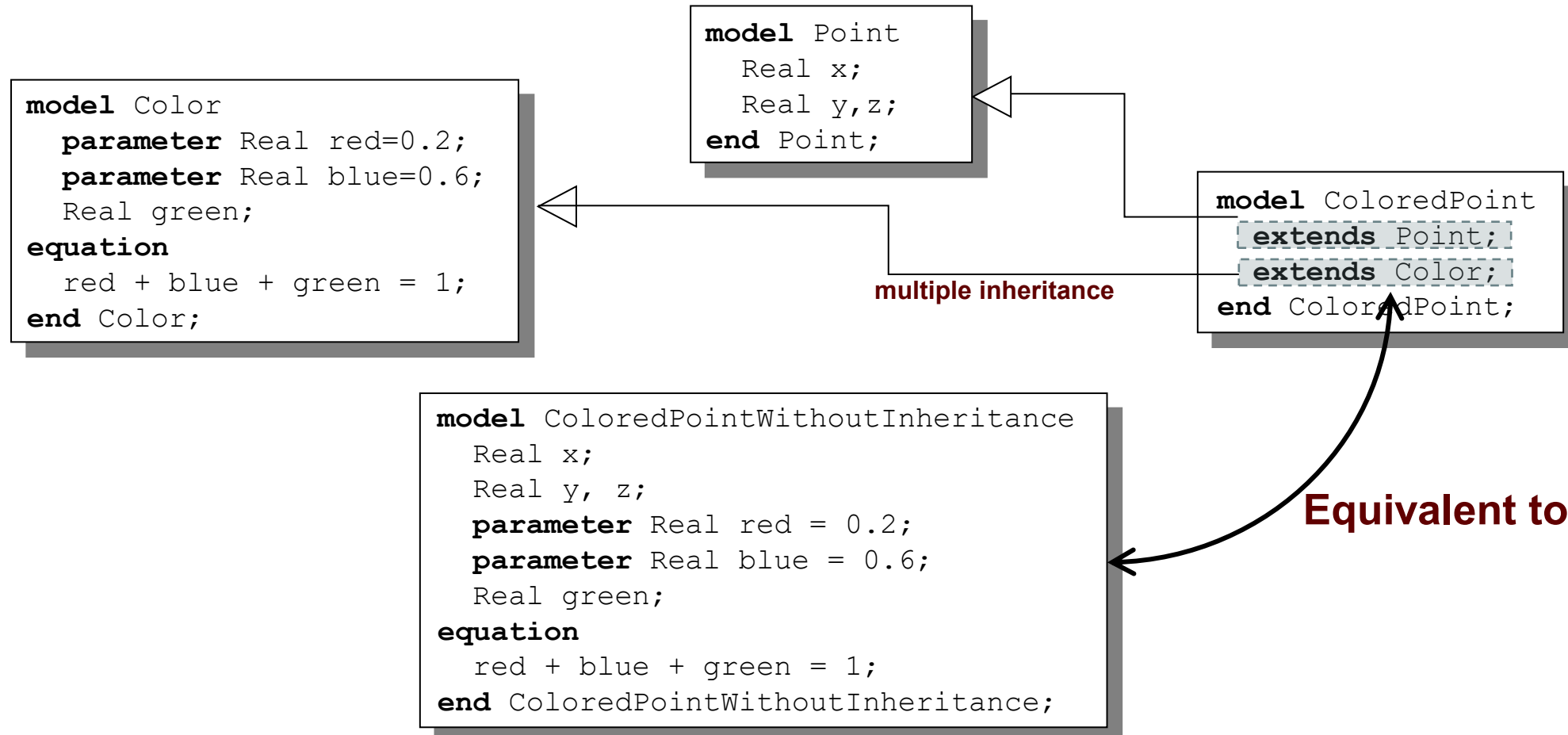
Inheritance



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

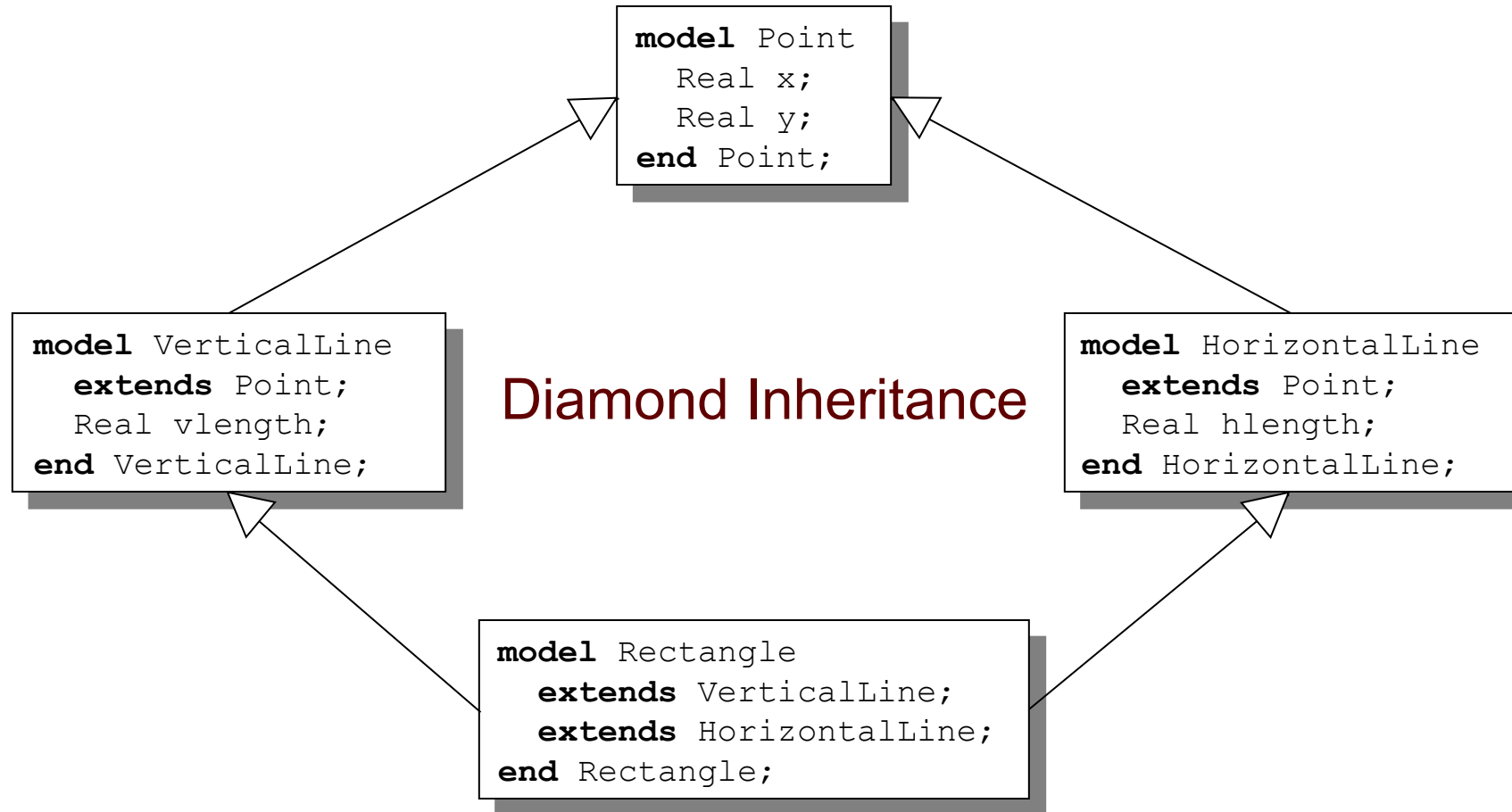
Multiple Inheritance

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:

```
model SameColor = Color;
```

Equivalent to:

inheritance ←

```
model SameColor  
  extends Color;  
end SameColor;
```

- Often used for introducing new names of types:

```
type Resistor = Real;
```

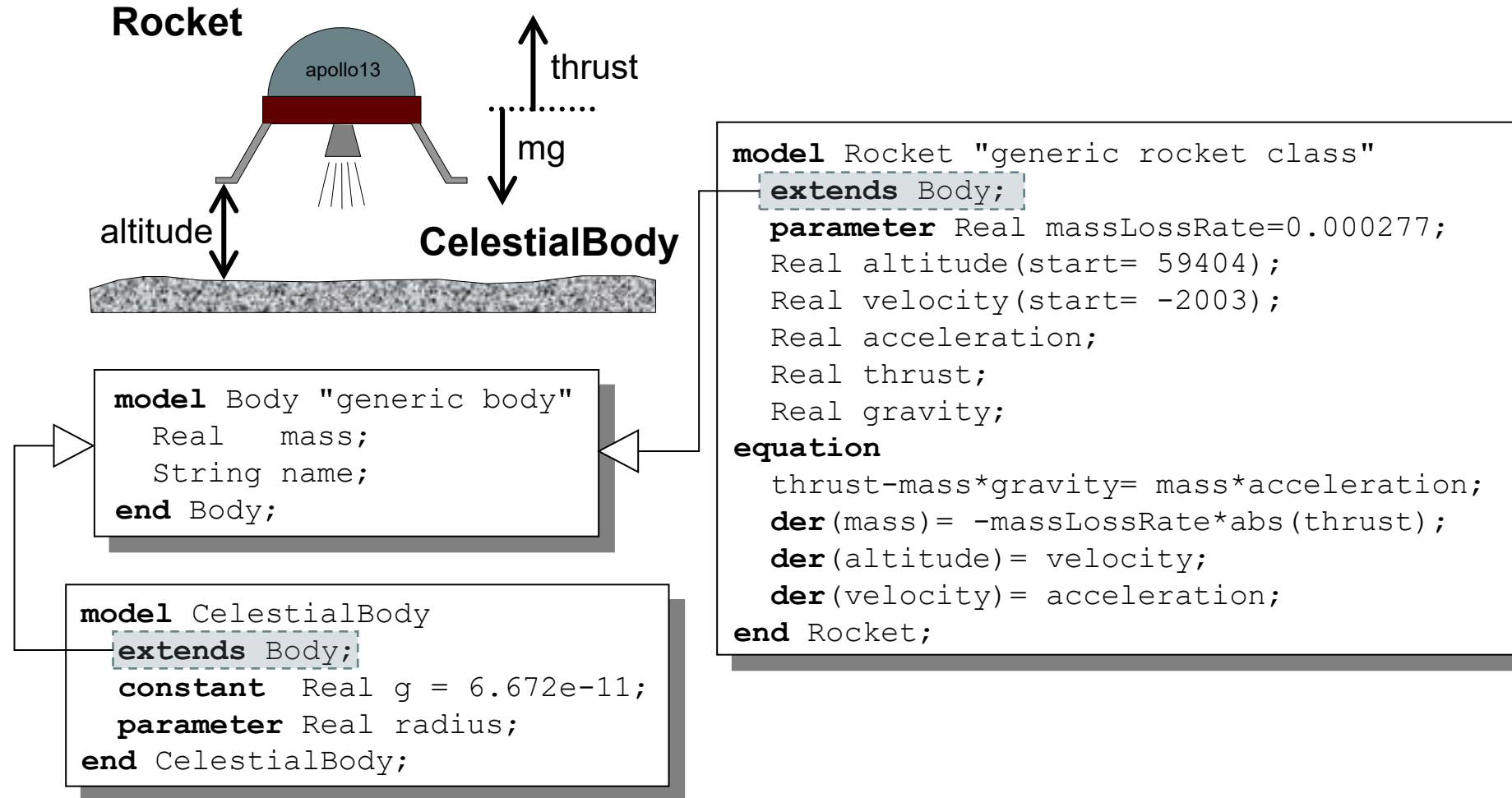
```
connector MyPin = Pin;
```


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:

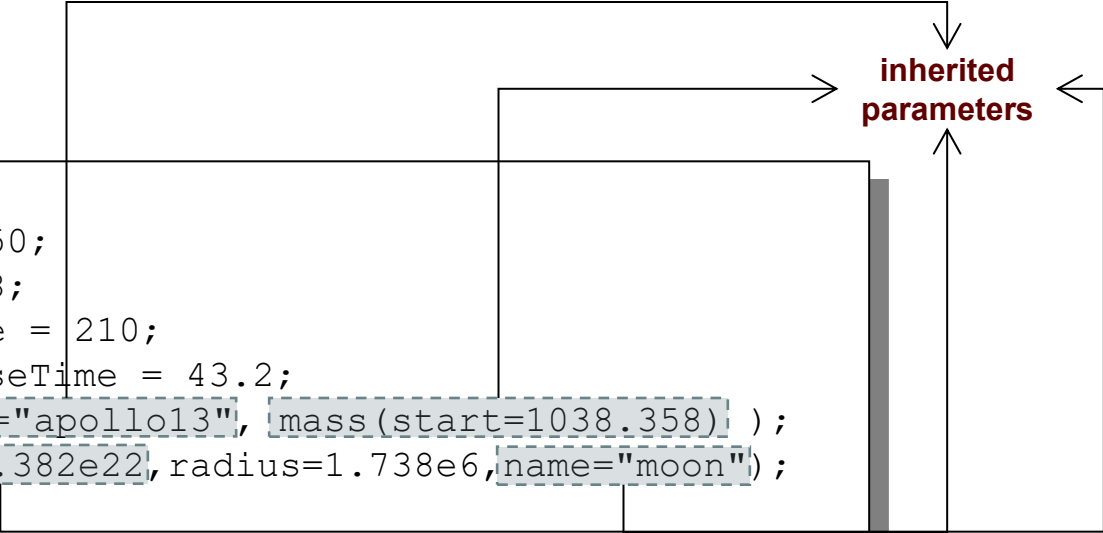
```
...  
Real altitude(start= 59404);  
...
```

The Moon Landing - Example Using Inheritance (I)



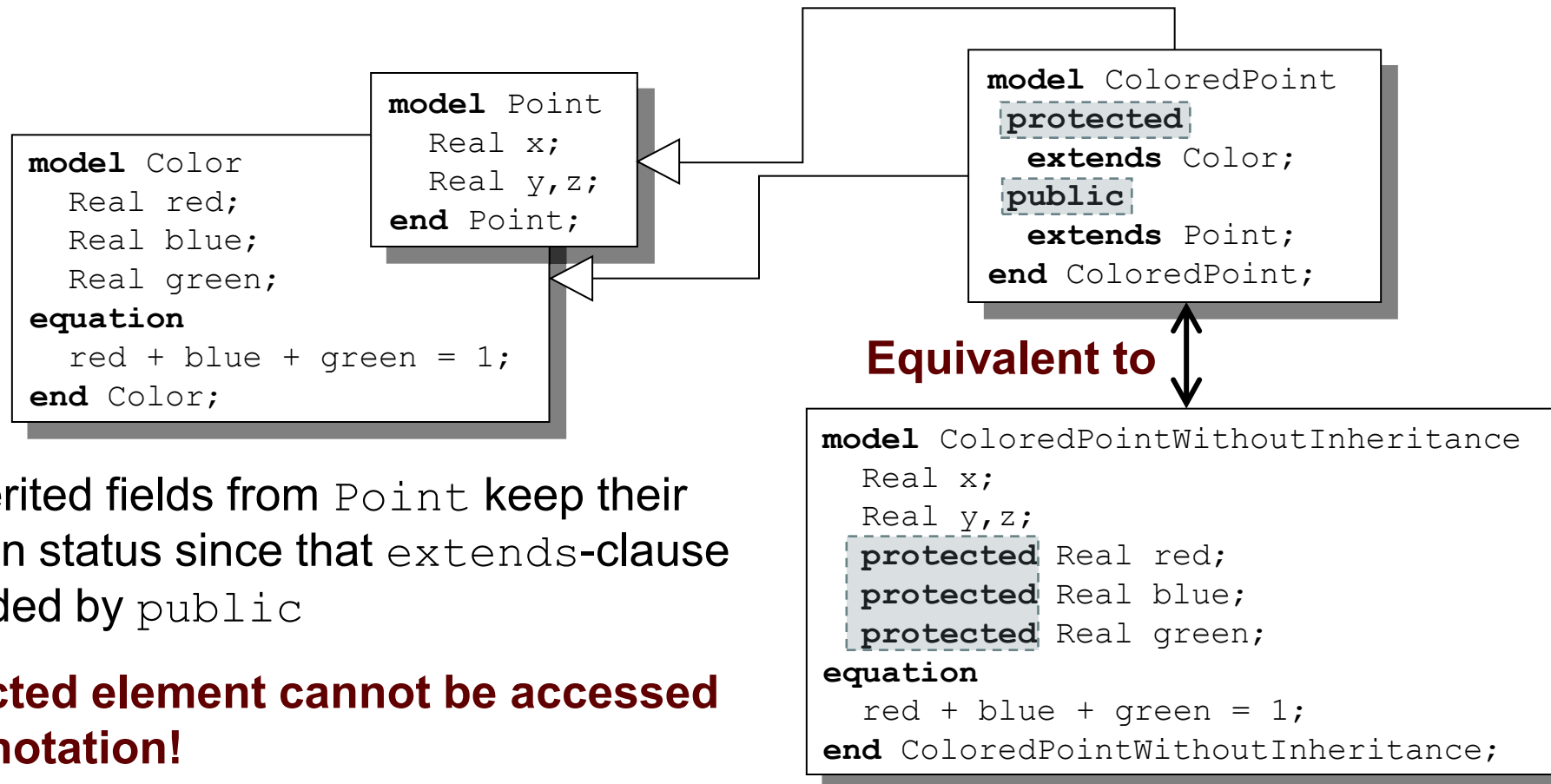
The Moon Landing - Example using Inheritance (II)

```
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;
```



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.

```
model 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!

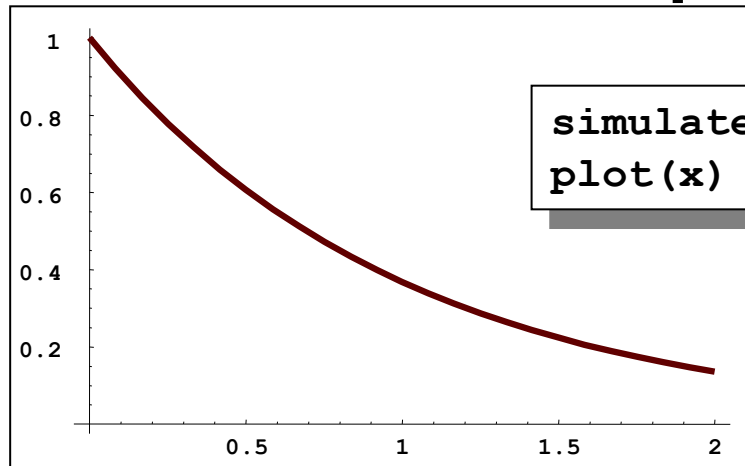
A Modelica “Hello World” model

Equation: $x' = -x$

Initial condition: $x(0) = 1$

```
model 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

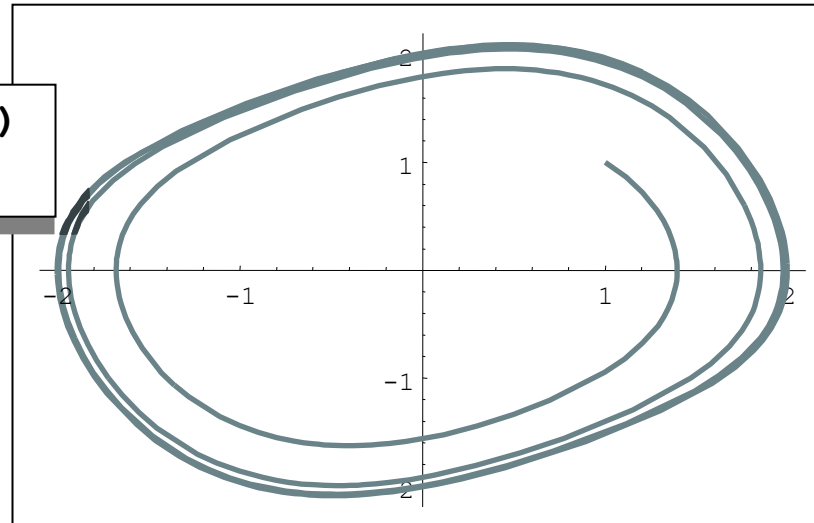


```
simulate(HelloWorld, stopTime = 2)  
plot(x)
```

(extra) Exercise 2.2 – Van der Pol Oscillator

```
model 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

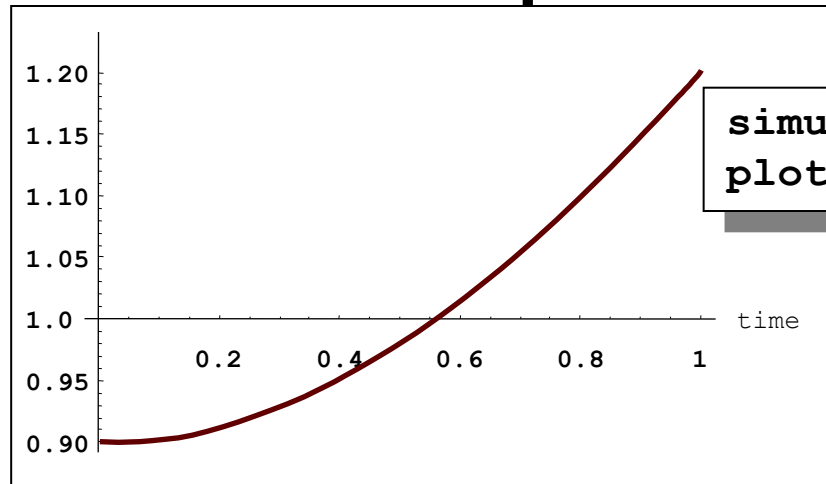
Include algebraic equation

Algebraic equations contain
no derivatives

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

```
model 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

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

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

$$x(0) = 2$$

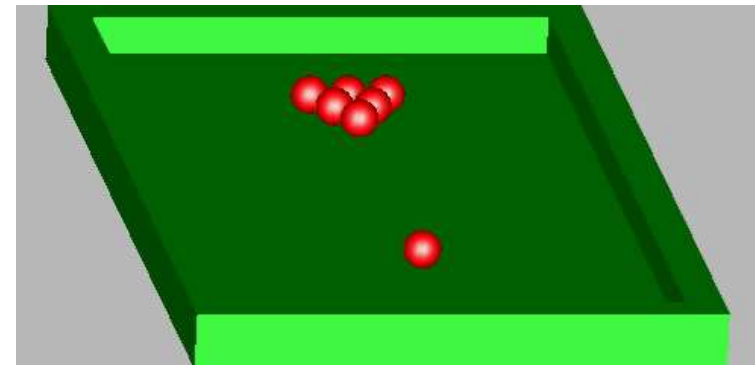
$$y(0) = 3$$

(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**.

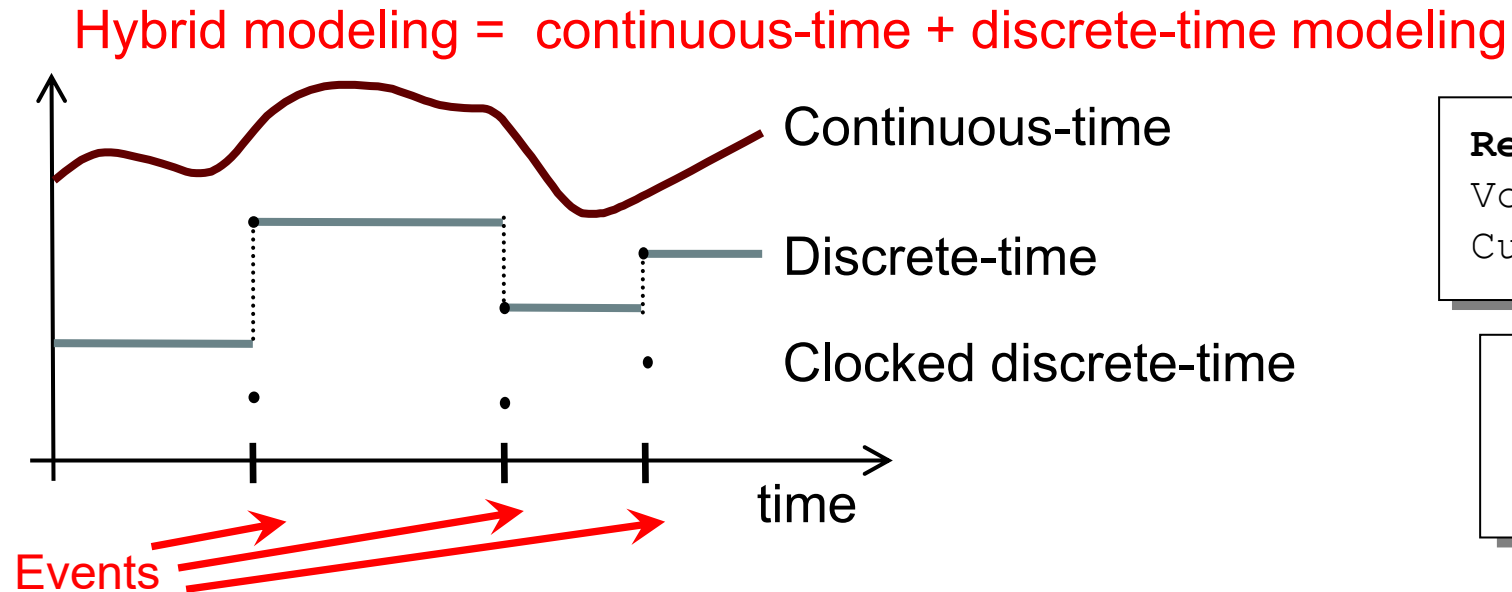
Part III b

Discrete Events and Hybrid Systems



Picture: Courtesy Hilding Elmqvist

Modelica Hybrid Modeling



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

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

- A *point* in time that is instantaneous, i.e., has zero duration
- An event *condition* 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

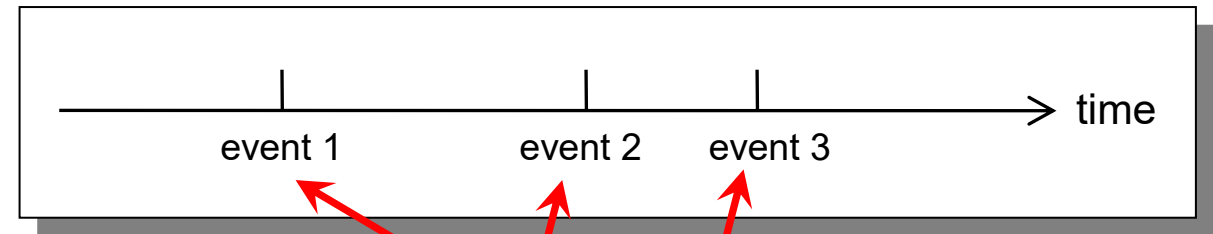
If-expression

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

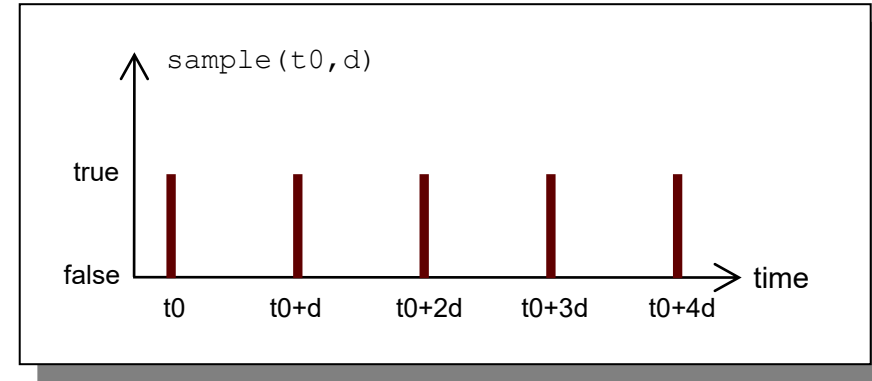
State event

```
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*d , where $i=0,1,\dots$

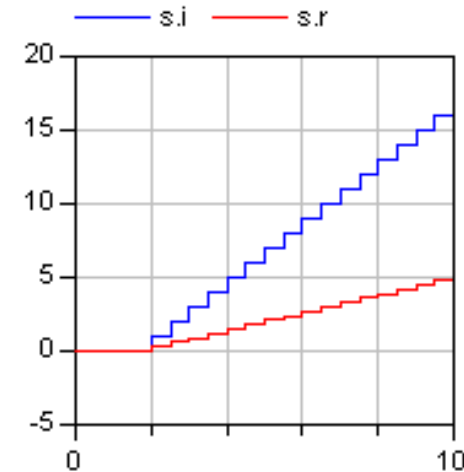


Variables need to be discrete

```
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.

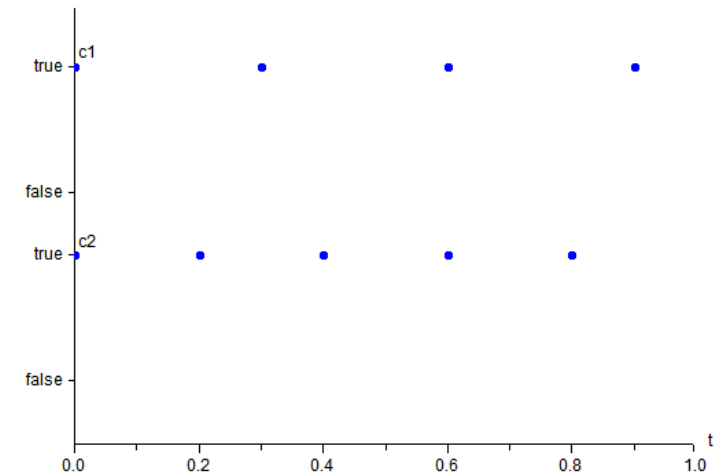


Generating Clock Tick Events using Clock()

(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

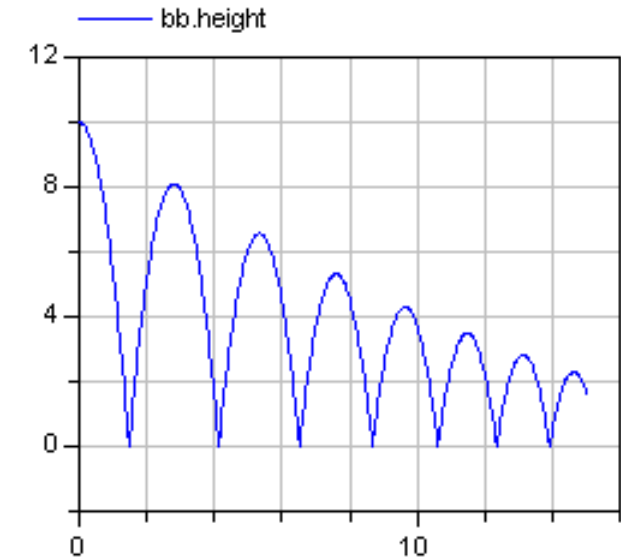
```
model ClockTicks
  // Integer quotient rational number interval clock
  Clock c1 = Clock(3,10);      // ticks: 0, 3/10, 6/10, ..
  // Clock with real value interval between ticks
  Clock c2 = Clock(0.2);      // ticks: 0.0, 0.2, 0.4, ...
end ClockTicks;
```



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;
```

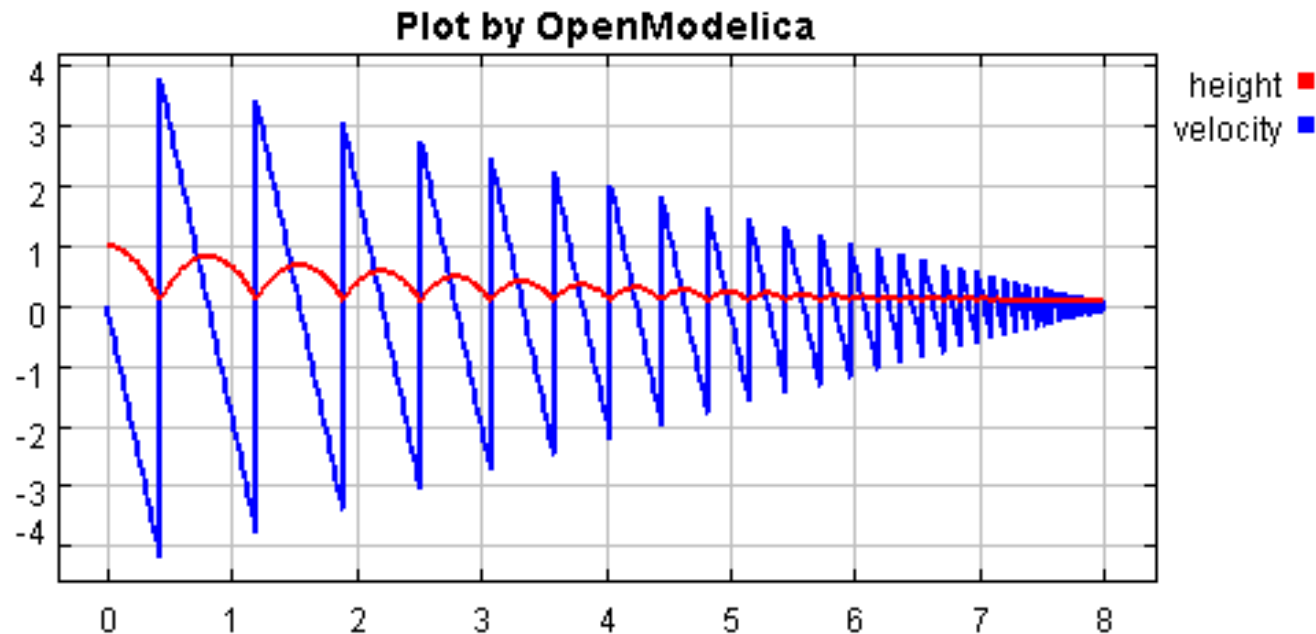


Reinit "assigns"
continuous-time variable
`velocity` a new value

Initial conditions

Exercise 2.6 – BouncingBall

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



Part IIc

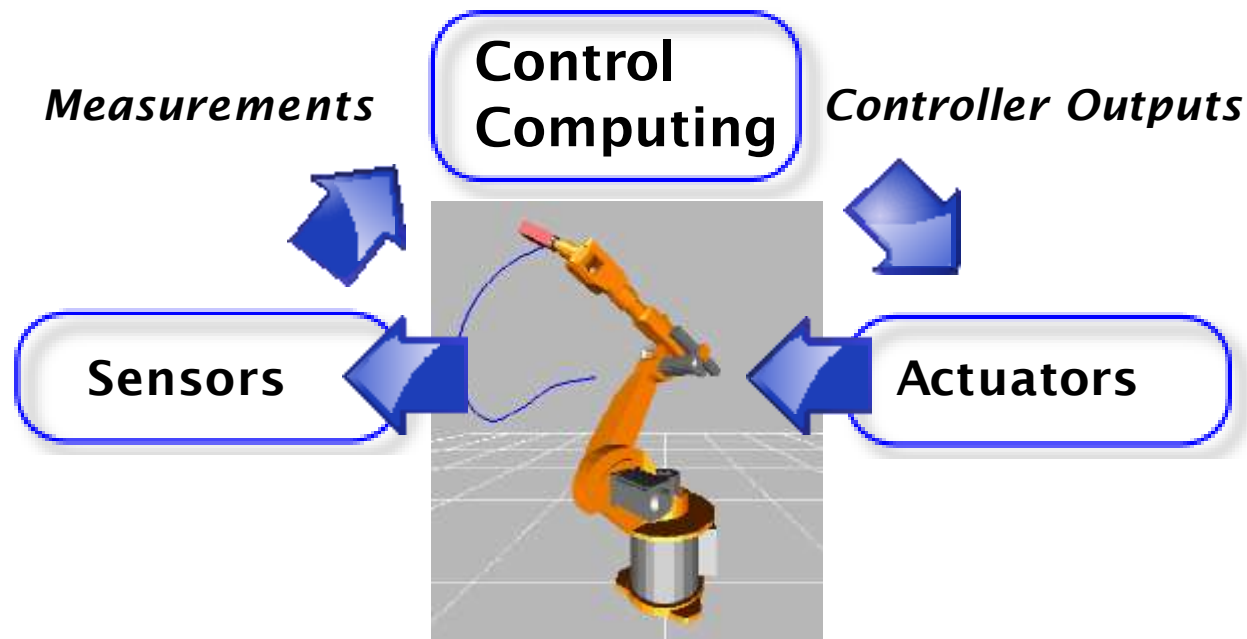
Clocked Synchronous Models and State Machines

and Applications for Digital Controllers

Control System Applications

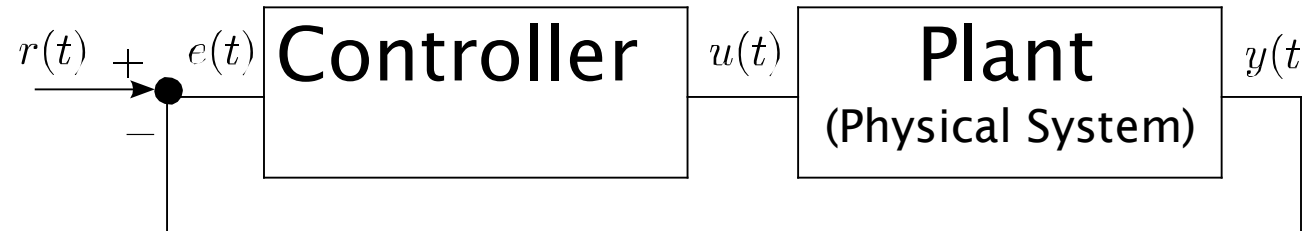
Control System

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

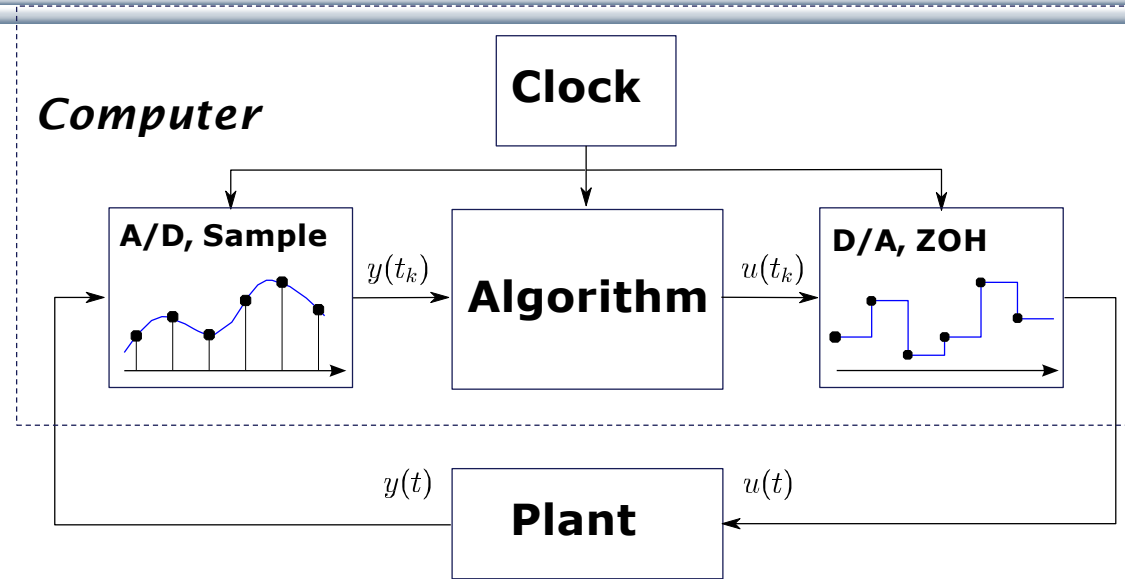


- $r(t)$ reference (setpoint)
- $e(t)$ error
- $y(t)$ measured process variable (plant output)
- $u(t)$ control output variable (plant input)

Usual Objective

Plant output should follow the reference signal.

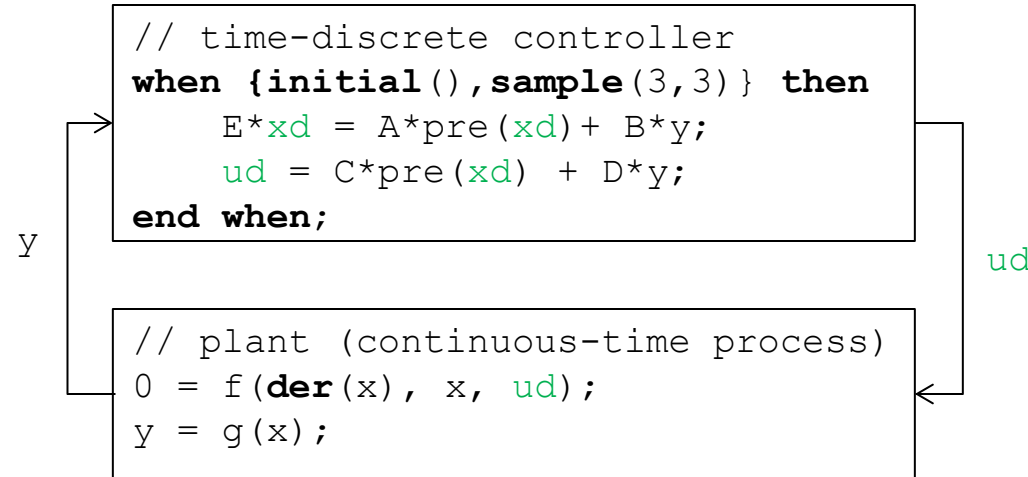
Embedded Real-Time Control System



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

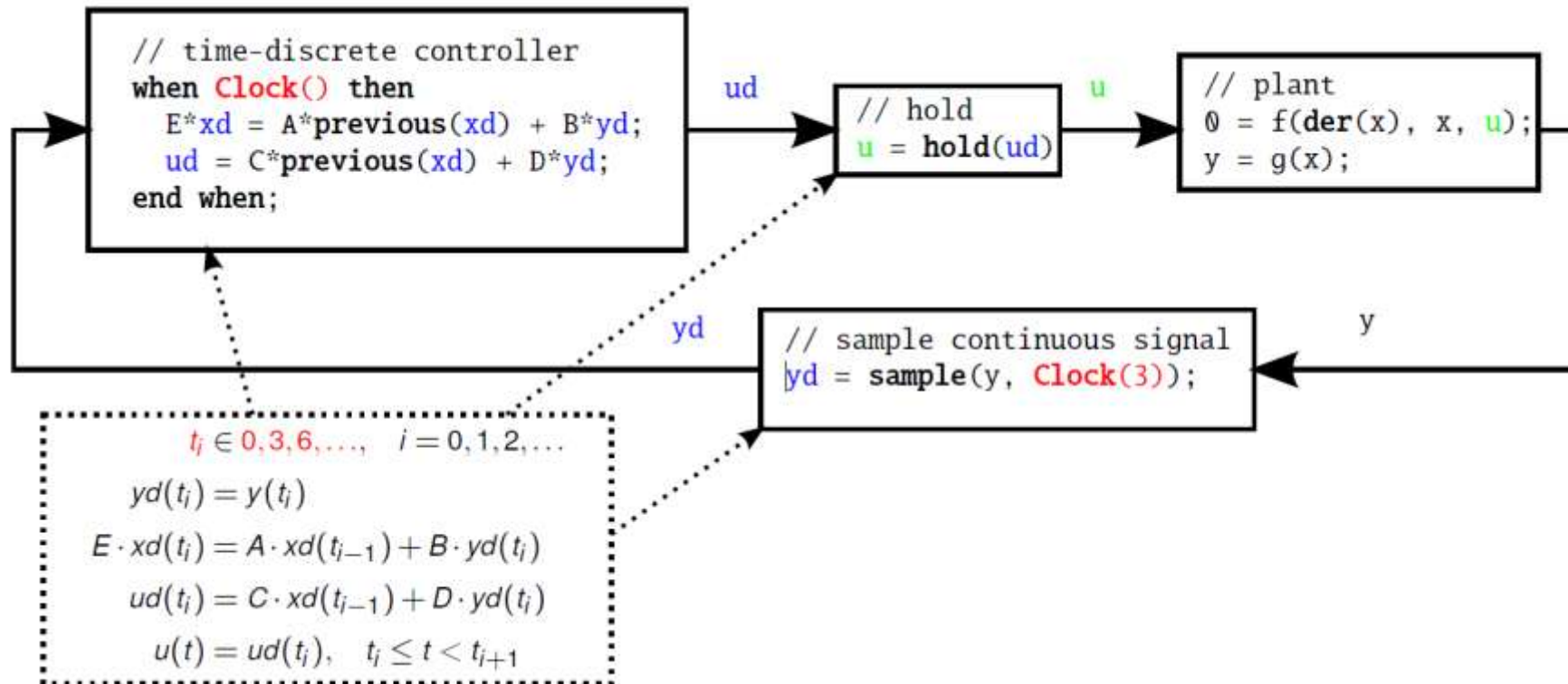
(unclocked models, using pre() and sample())



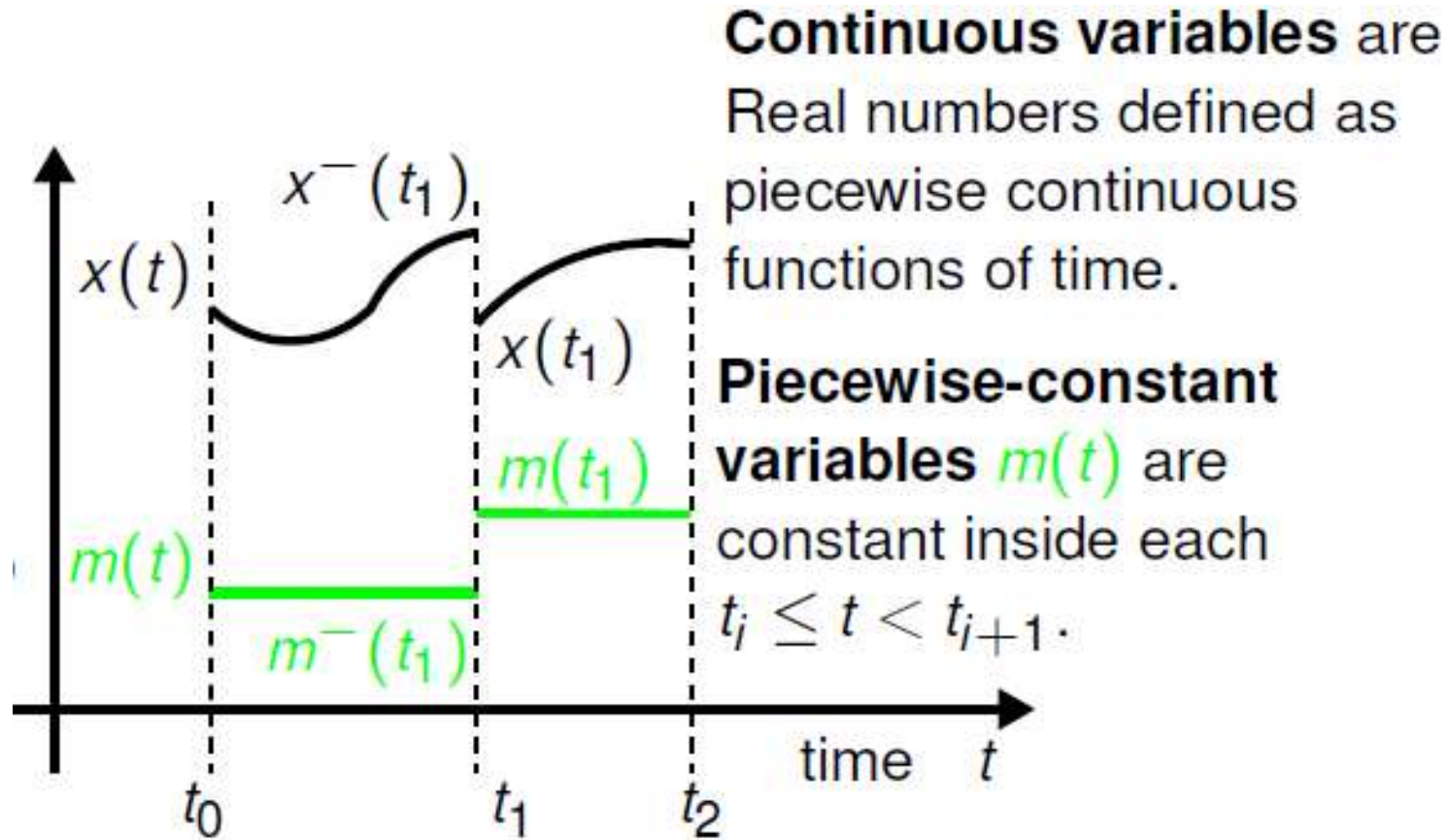
- *y* is automatically sampled at $t = 3, 6, 9, \dots$;
- *xd*, *u* are piecewise-constant variables that change values at sampling events (implicit zero-order hold)
- `initial()` triggers event at initialization ($t=0$)

Controller with Clocked Synchronous Constructs

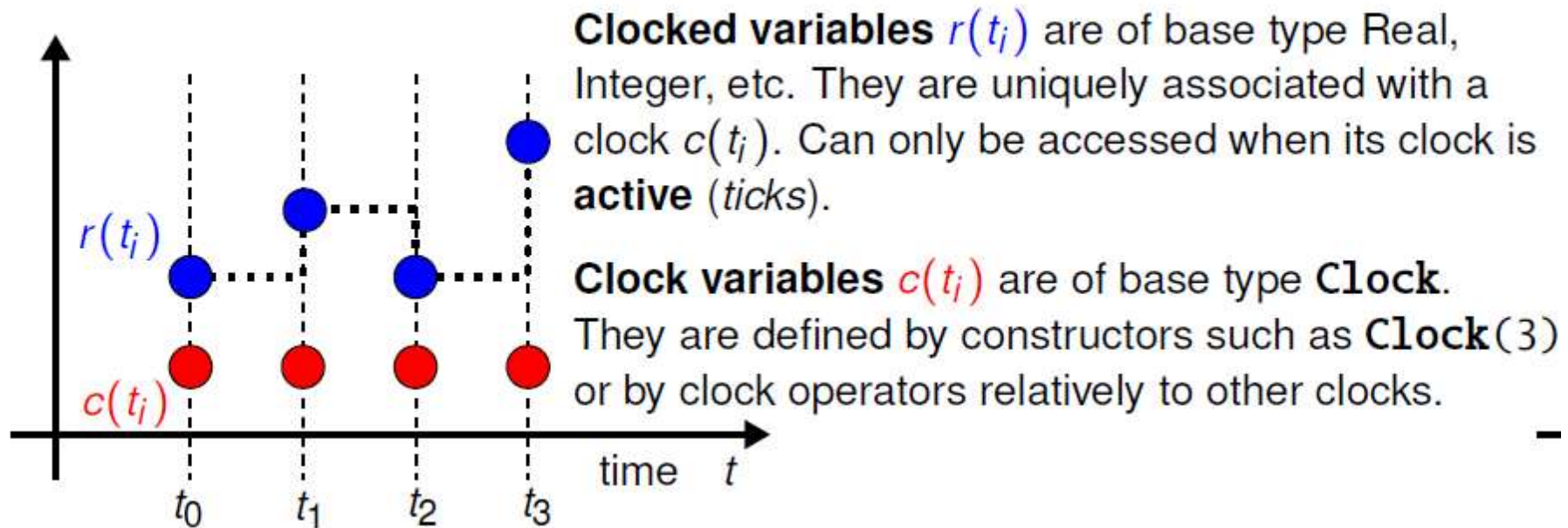
clocked models using Clock(), previous(), hold() in Modelica 3.3



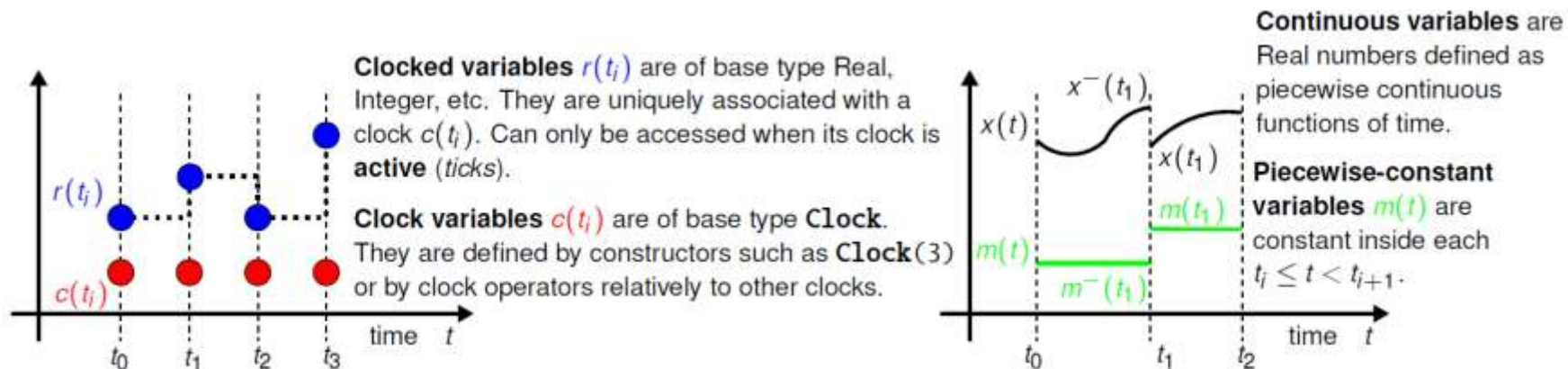
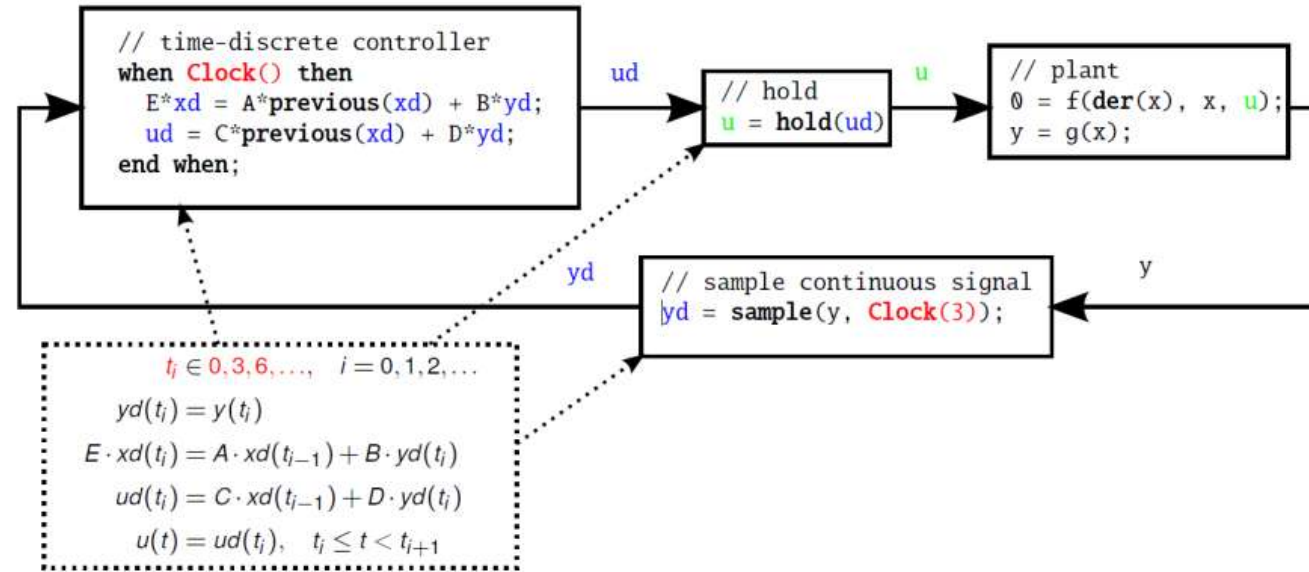
Unlocked Variables in Modelica 3.2



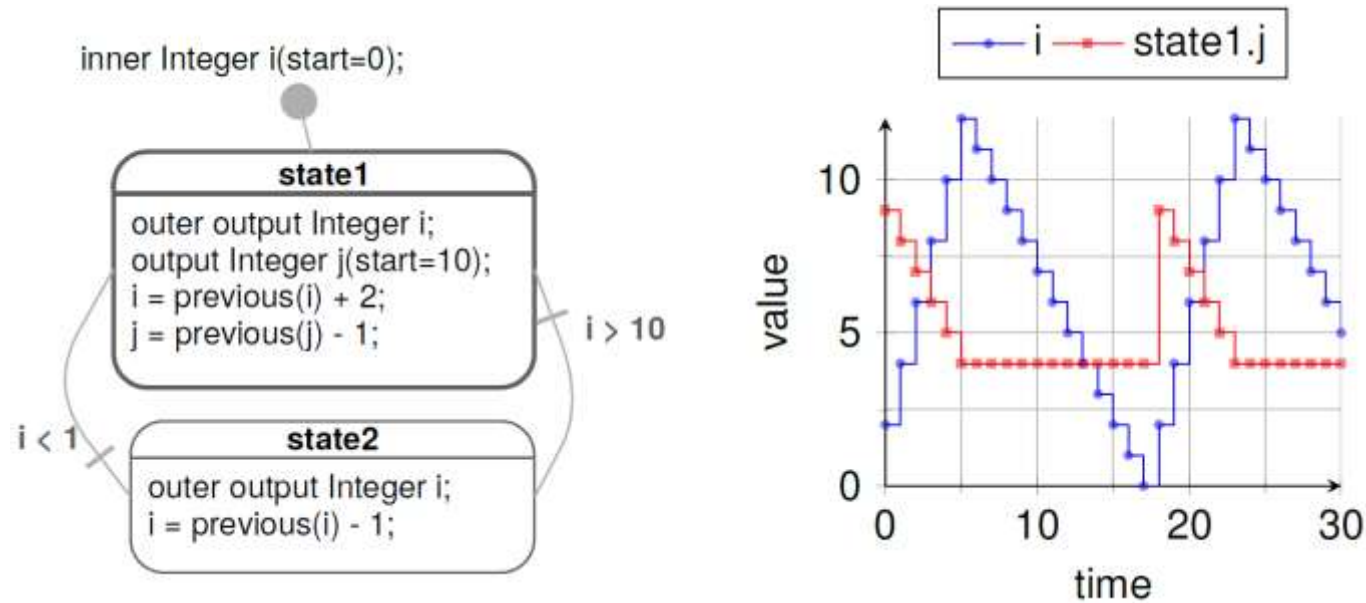
Clock variables (Clock) and Clocked Variables (Real) (in Modelica 3.3)



Clocked Synchronous Extension in Modelica 3.3



State Machines in Modelica 3.3: Simple Example

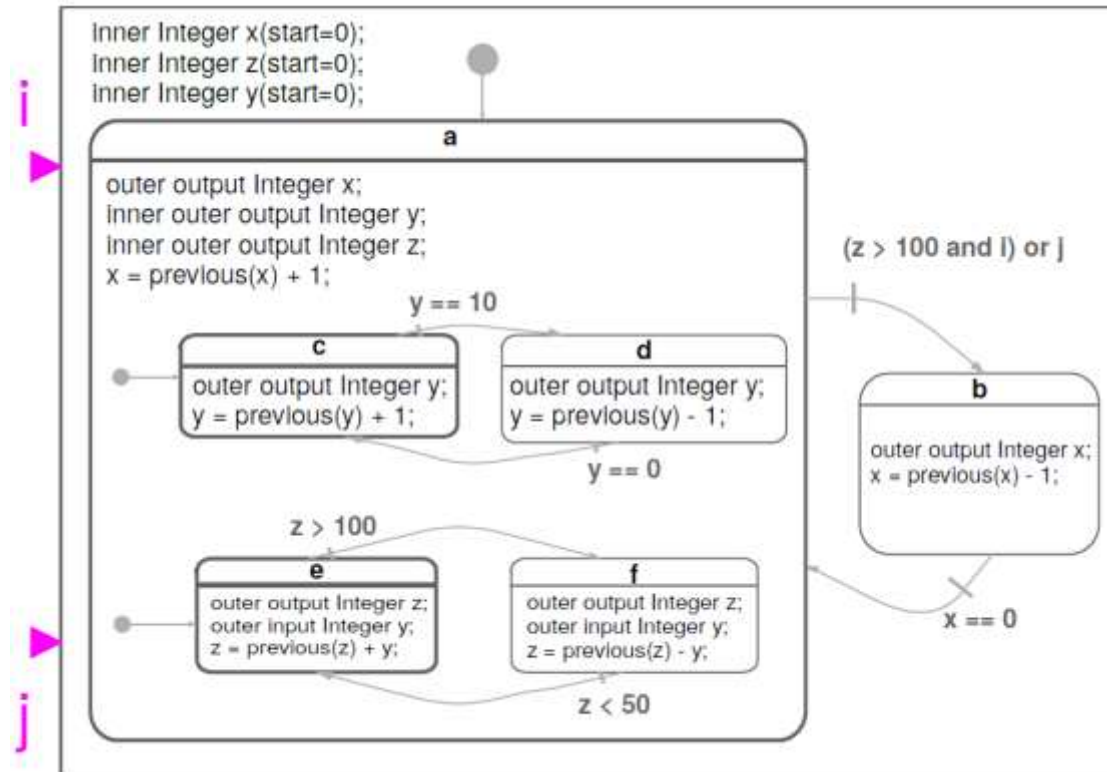


- Equations are active if corresponding *clock* ticks. Defaults to periodic clock with 1.0 s sampling period
- “i” is a shared variable, “j” is a local variable. Transitions are “*delayed*” and enter states by “*reset*”

Simple Example: Modelica Code

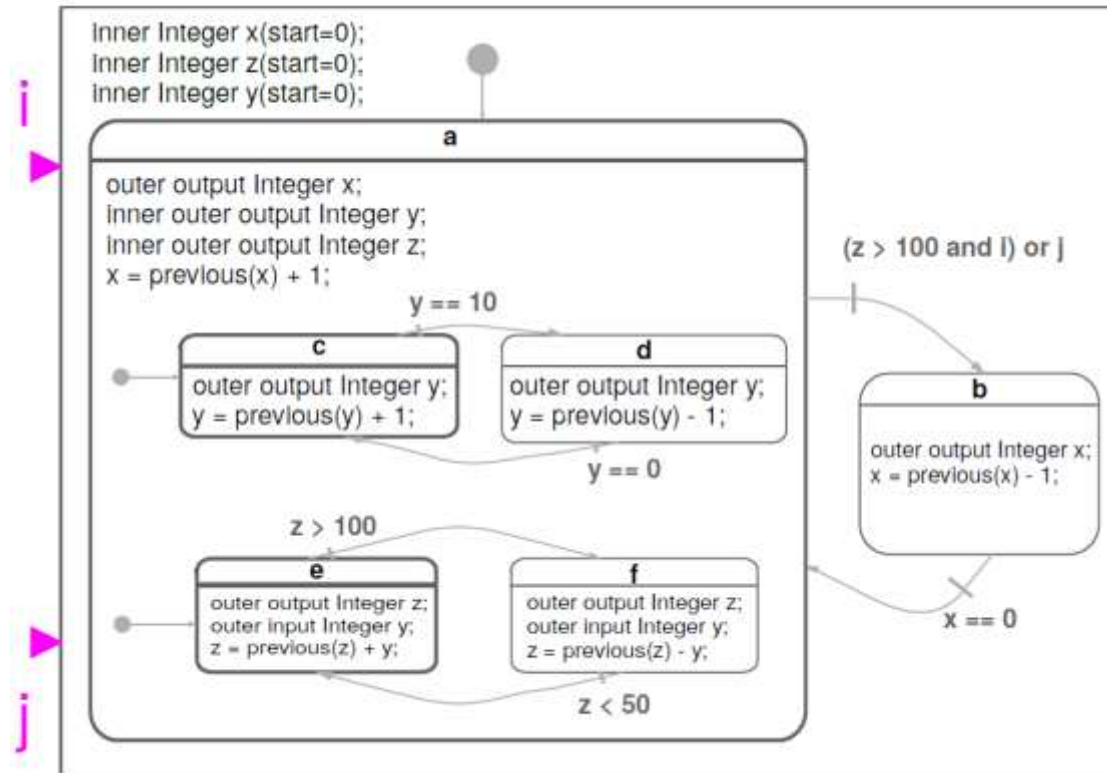
```
model Simple_NoAnnotations "Simple state machine"
  inner Integer i(start=0);
  block State1
    outer output Integer i;
    output Integer j(start=10);
  equation
    i = previous(i) + 2;
    j = previous(j) - 1;
  end State1;
  State1 state1;
  block State2
    outer output Integer i;
  equation
    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 Sychrone 3.0.

Hierarchical and Parallel Composition

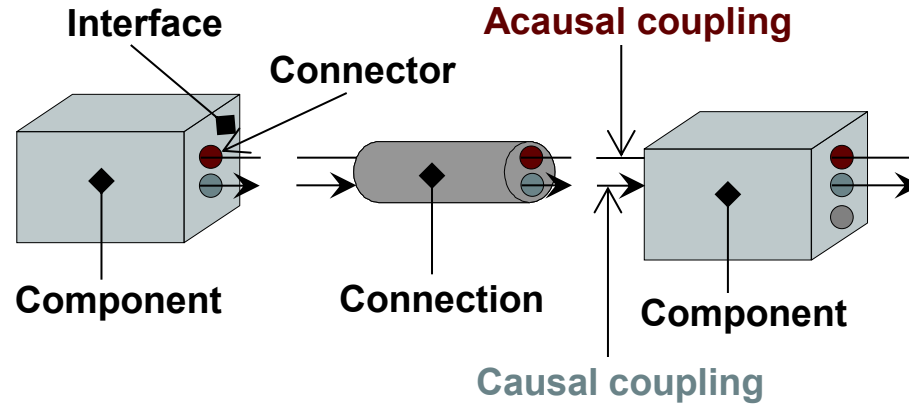


Semantics of Modelica state machines (and example above) inspired by Florence Maraninchi & Yann Rémond's "Mode-Automata" and by Marc Pouzet's Lucid Synchrone 3.0.

Part IV

Components, Connectors and Connections – Modelica Libraries and Graphical Modeling

Software Component Model



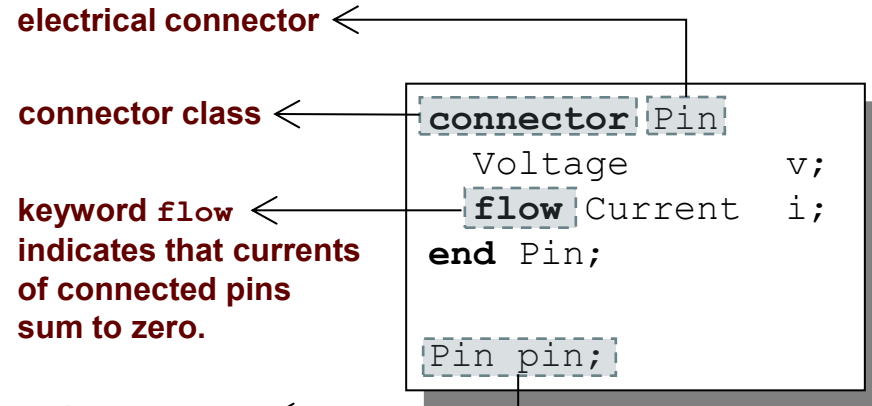
A component class should be defined *independently of the environment*, very essential for *reusability*

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

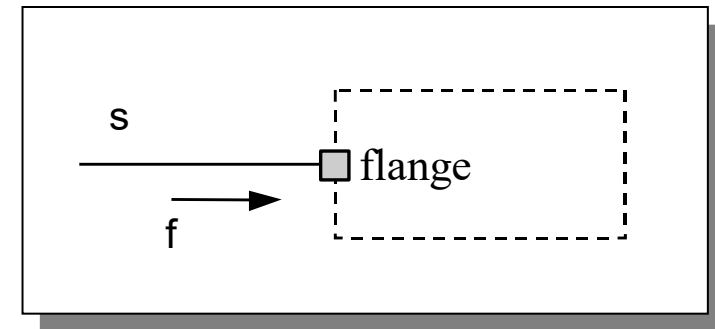
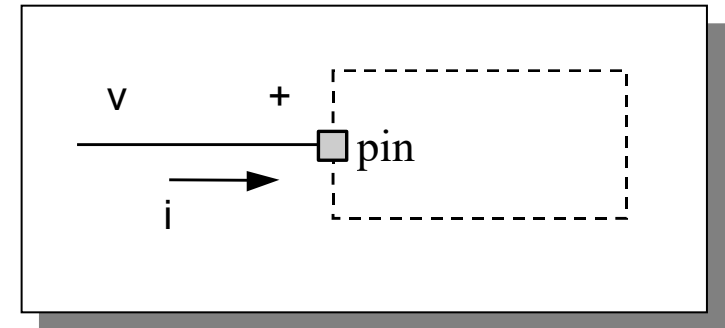
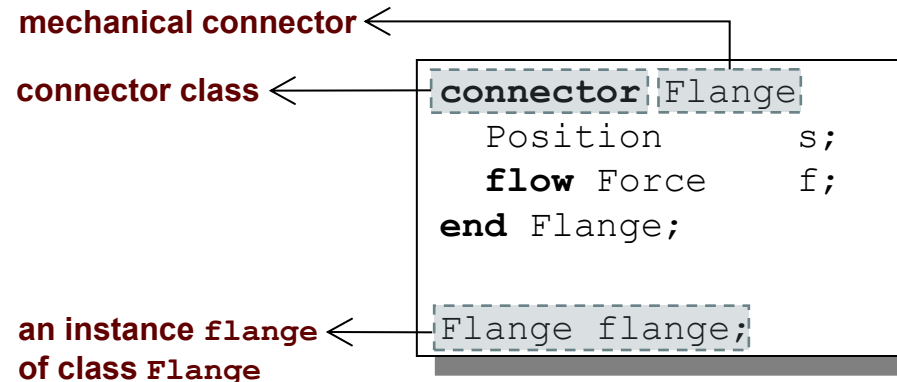
Complex systems usually consist of large numbers of *connected* components

Connectors and Connector Classes

Connectors are instances of *connector classes*



an instance **pin** ←
of class **Pin**



The `flow` prefix

Three possible kinds of variables in connectors:

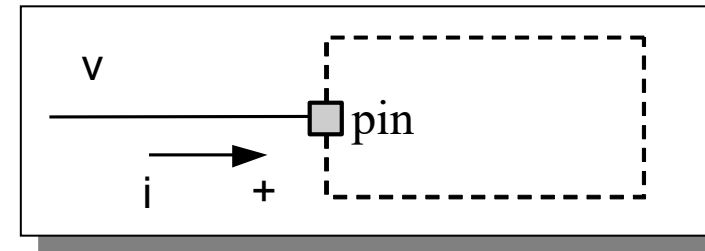
- *Potential variables* *potential* or energy level
- *Flow variables* represent some kind of flow
- *Stream variables* represent fluid flow in convective transport

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

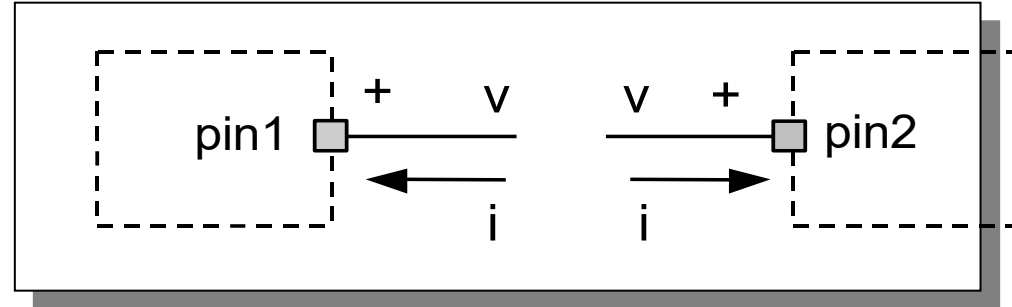
Domain Type	Potential	Flow	Carrier	Modelica Library
Electrical	Voltage	Current	Charge	Electrical. Analog
Translational	Position	Force	Linear momentum	Mechanical. Translational
Rotational	Angle	Torque	Angular momentum	Mechanical. Rotational
Magnetic	Magnetic potential	Magnetic flux rate	Magnetic flux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

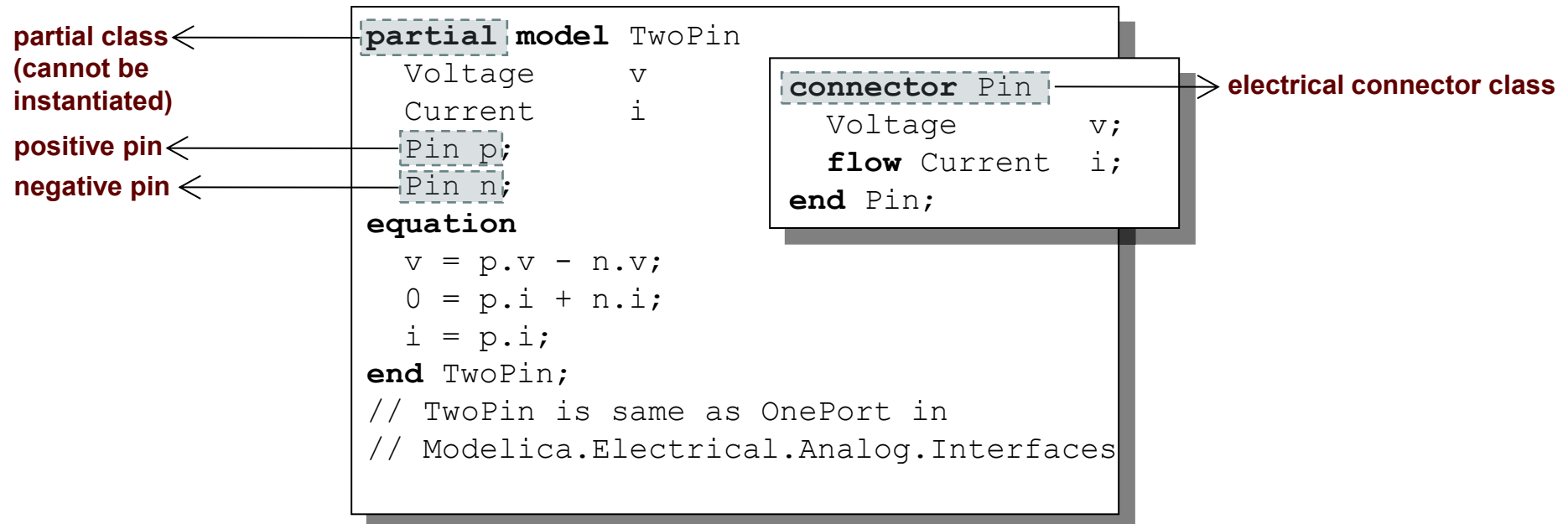
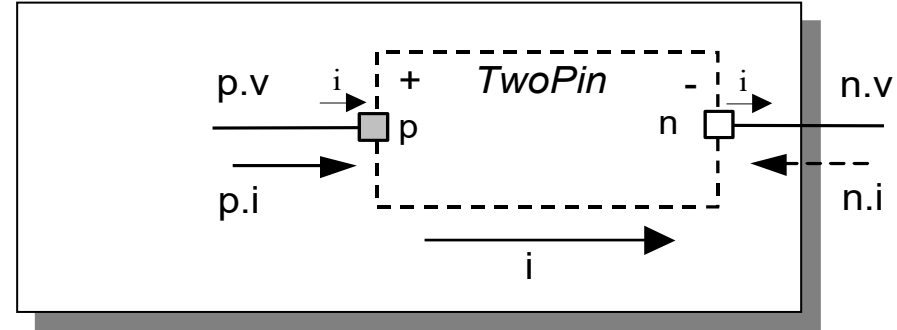
$$v_1 = v_2 = v_3 = \dots v_n$$

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

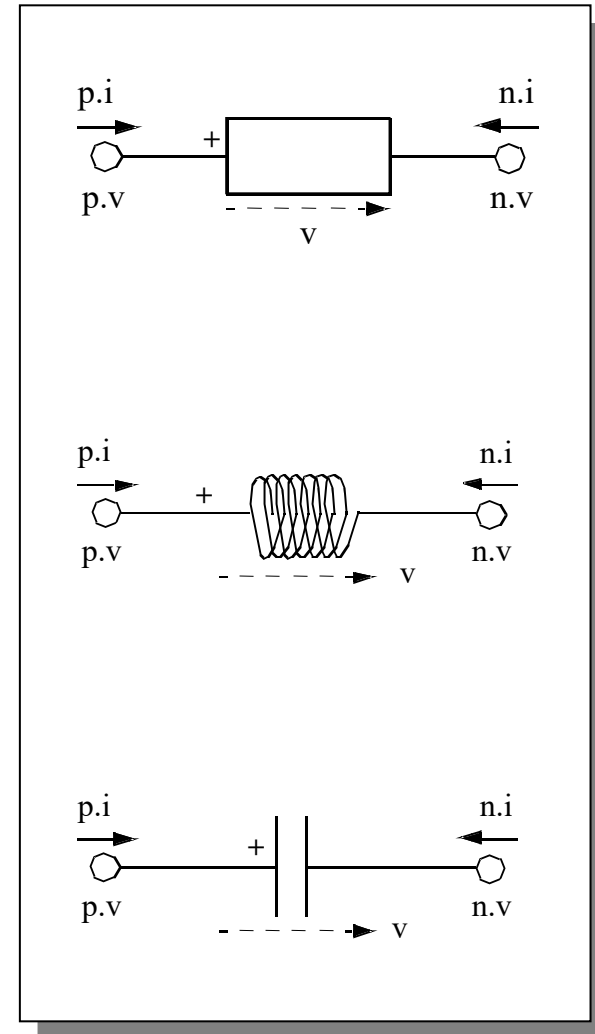


Electrical Components

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

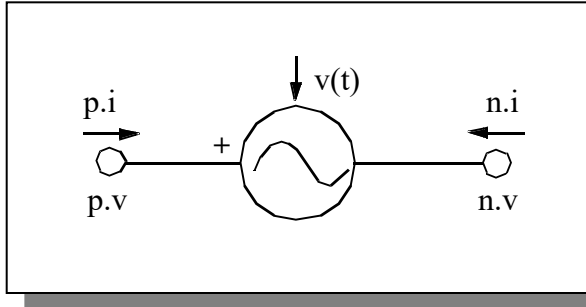
```
model Inductor "Ideal electrical inductor"  
  extends TwoPin;  
  parameter Real L "Inductance";  
equation  
     $L \cdot \text{der}(i) = v$ ;  
end Inductor;
```

```
model Capacitor "Ideal electrical capacitor"  
  extends TwoPin;  
  parameter Real C ;  
equation  
     $i = C \cdot \text{der}(v)$ ;  
end Capacitor;
```

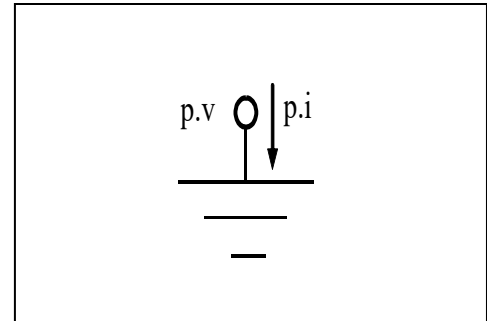


Electrical Components cont'

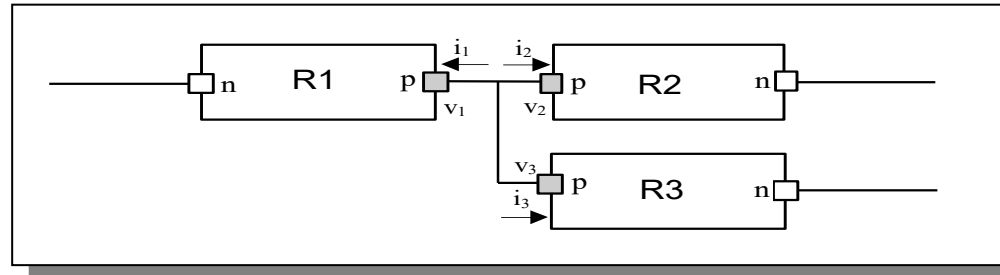
```
model Source
  extends TwoPin;
  parameter Real A,w;
equation
  v = A*sin(w*time);
end Resistor;
```



```
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

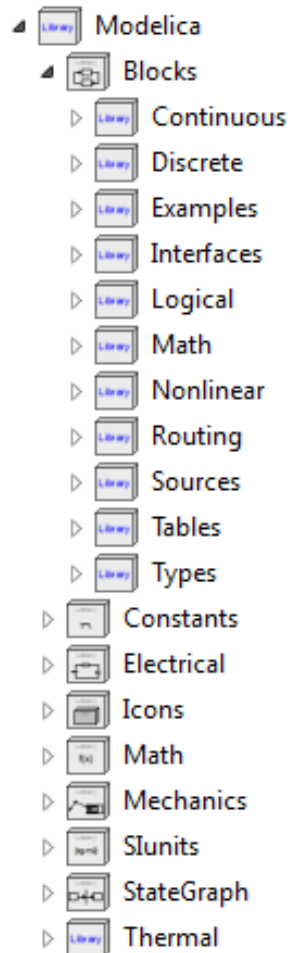
- *Modelica Standard Library* (called Modelica) is a standardized predefined package developed by Modelica Association
- It can be used freely for both commercial and non-commercial purposes under the conditions of *The Modelica License*.
- Modelica libraries are available online including documentation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

Modelica Standard Library cont'

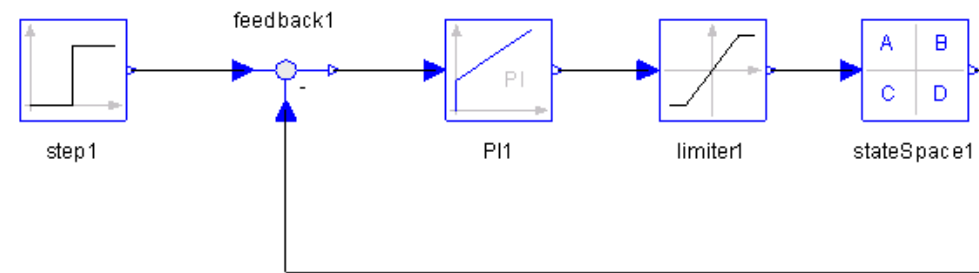
The Modelica Standard Library contains components from various application areas, including the following sublibraries:

- 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

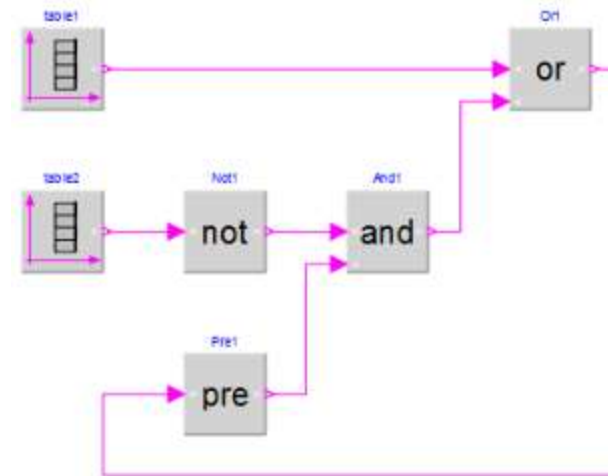
Modelica.Blocks



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

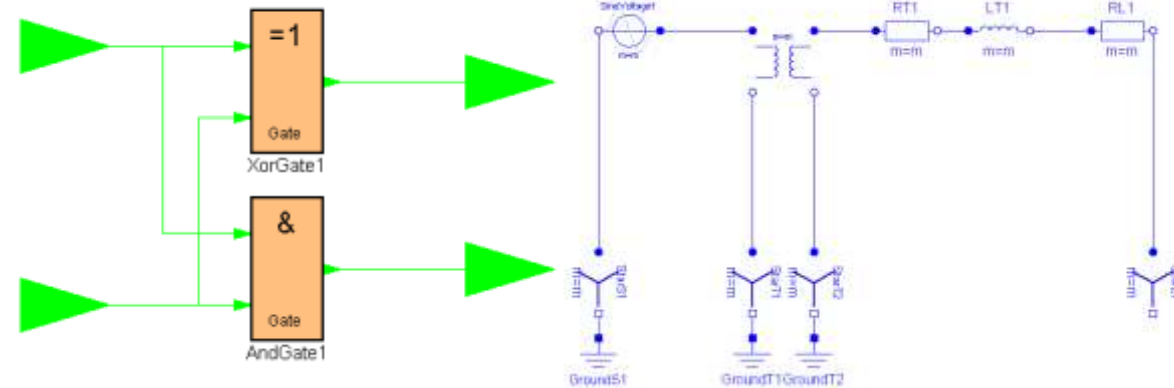
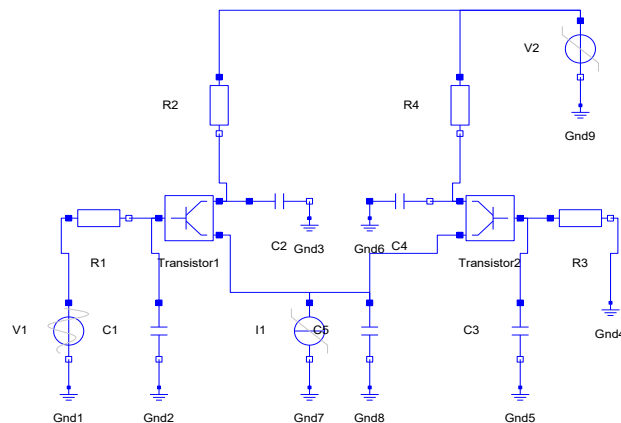


Machines



MultiPhase

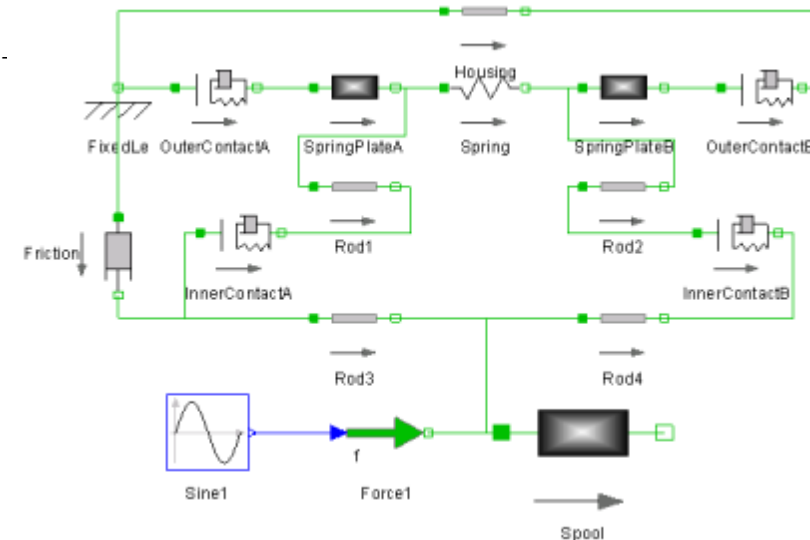
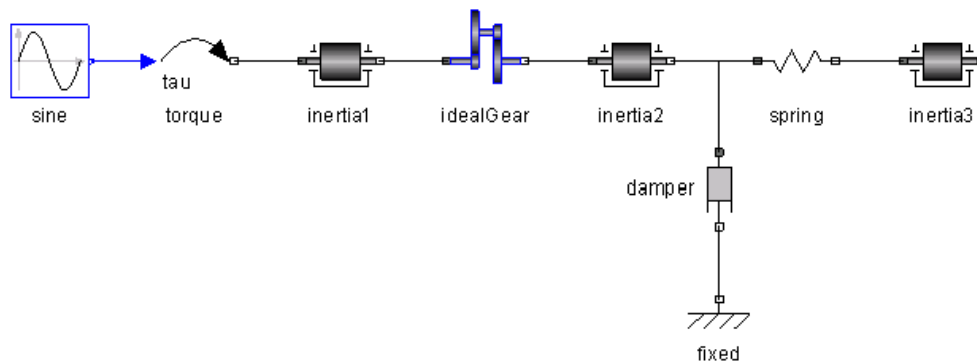
Examples:



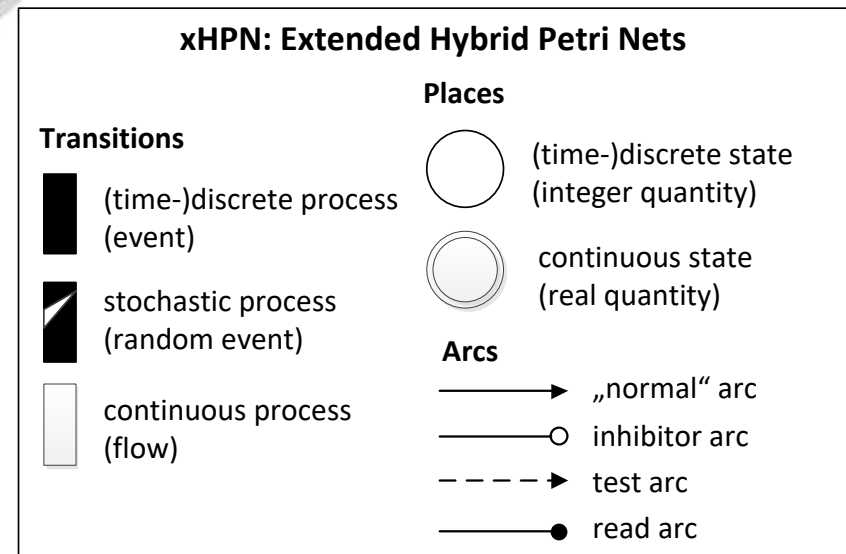
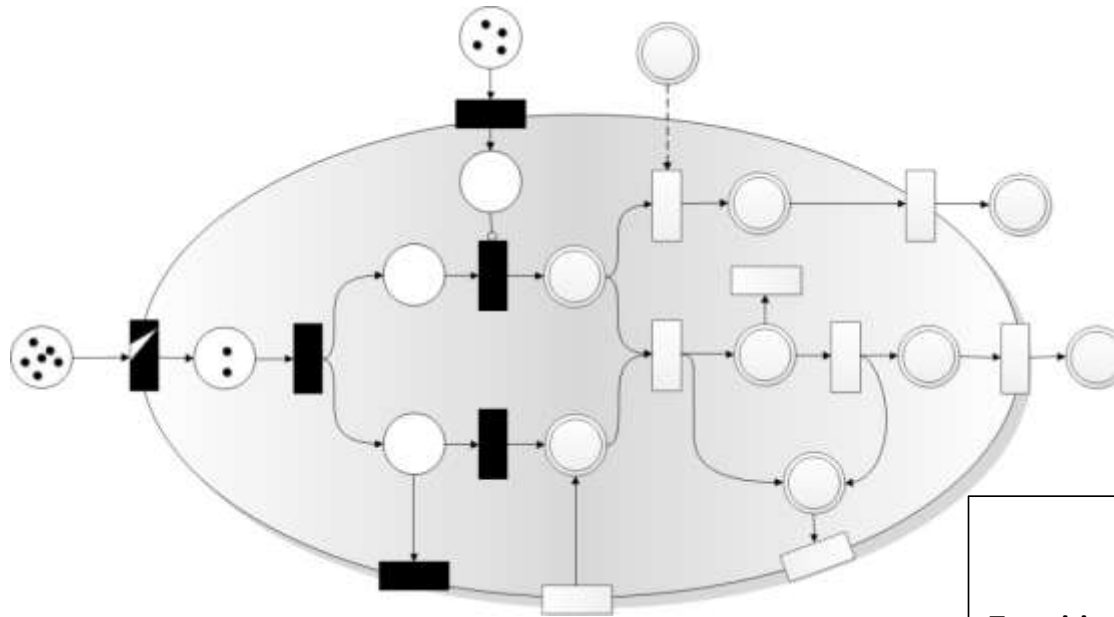
Package containing components for mechanical systems

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



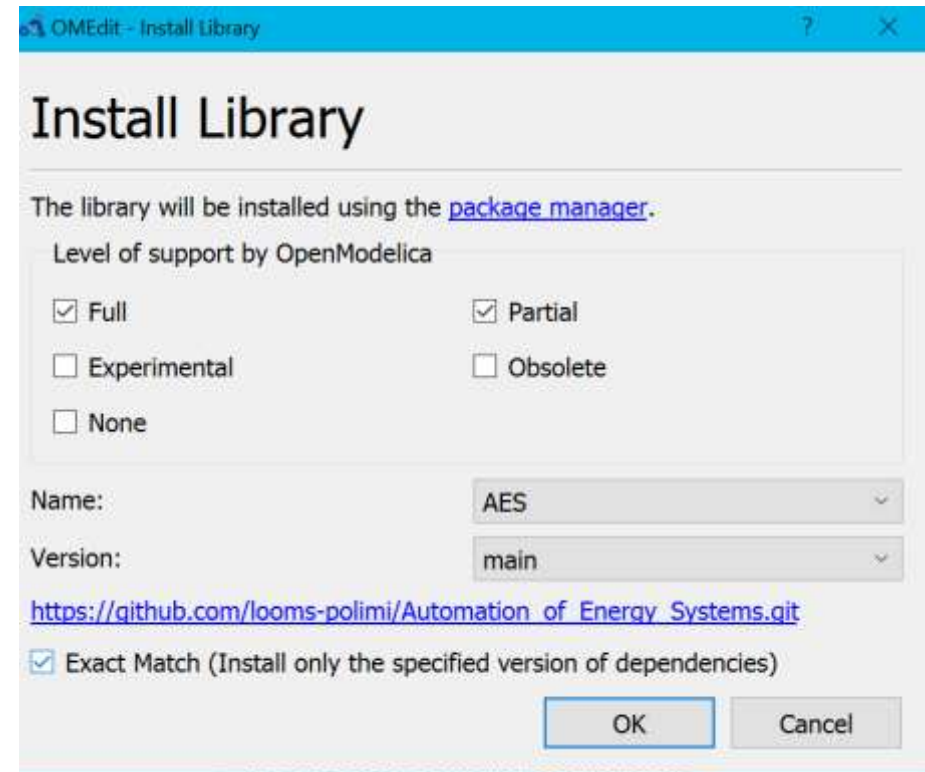
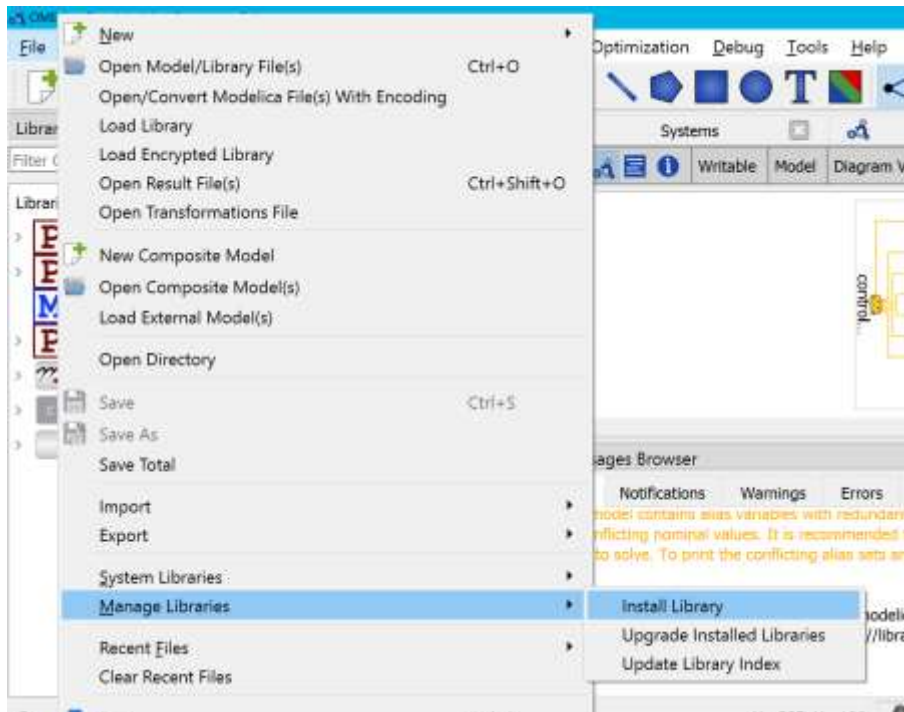
Other Free Libraries

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

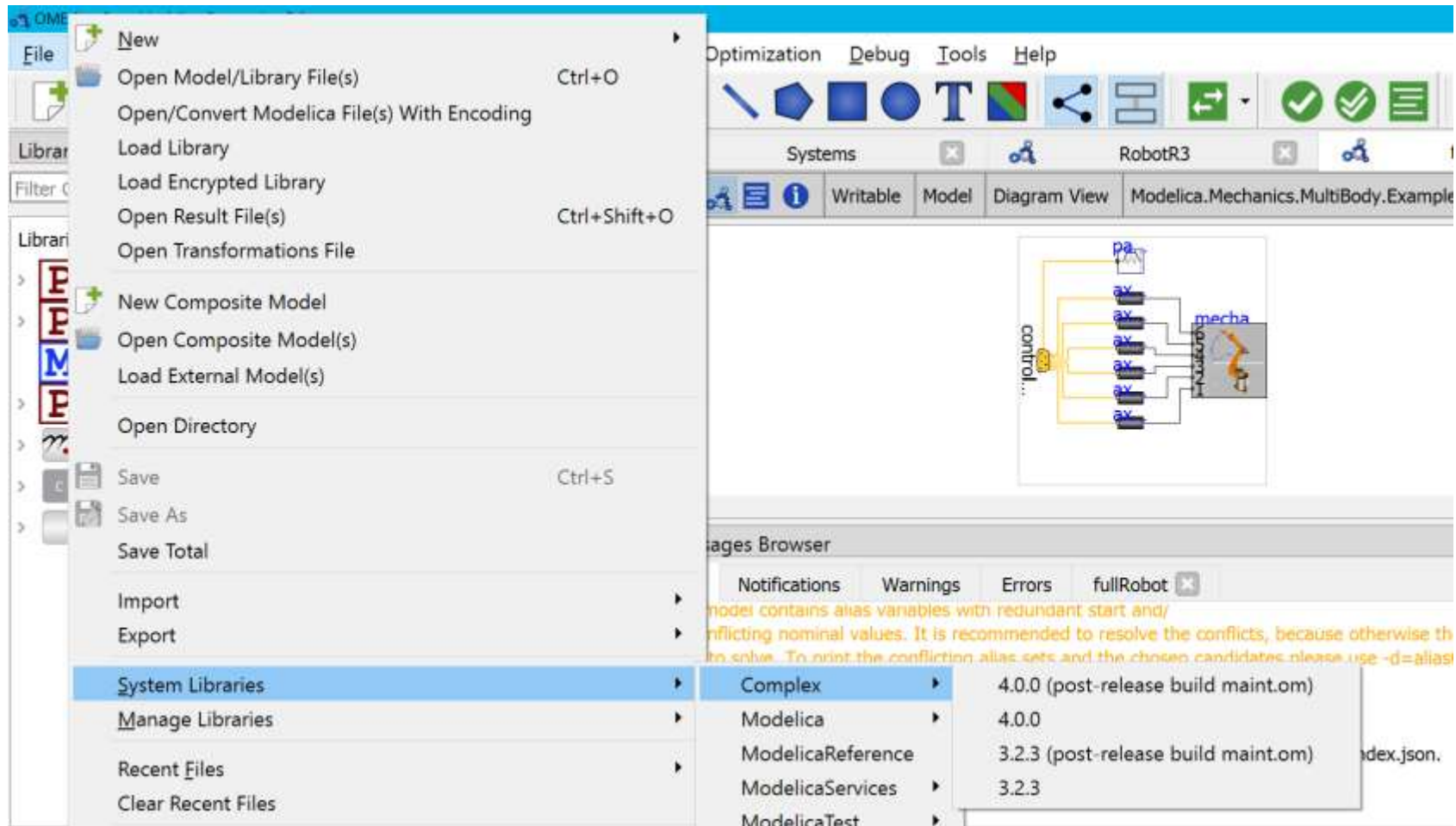
- WasteWater Wastewater treatment plants, 2003
- ATPlus Building simulation and control (fuzzy control included), 2005
- MotorCycleDynamics Dynamics and control of motorcycles, 2009
- NeuralNetwork Neural network mathematical models, 2006
- VehicleDynamics Dynamics of vehicle chassis (obsolete), 2003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

Install Libraries using OMEdit

- OMEdit can be used to install libraries



View installed libraries under System Libraries



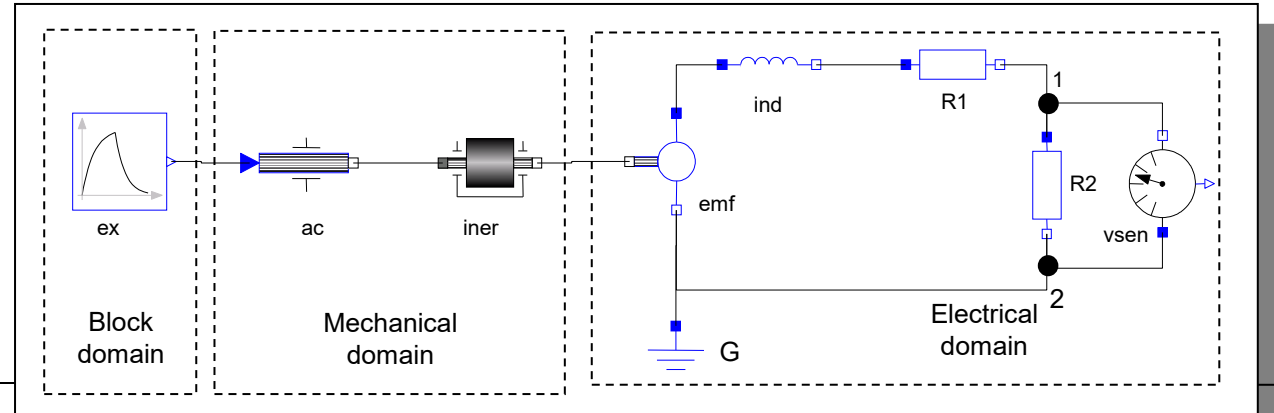
Some Commercial Libraries

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

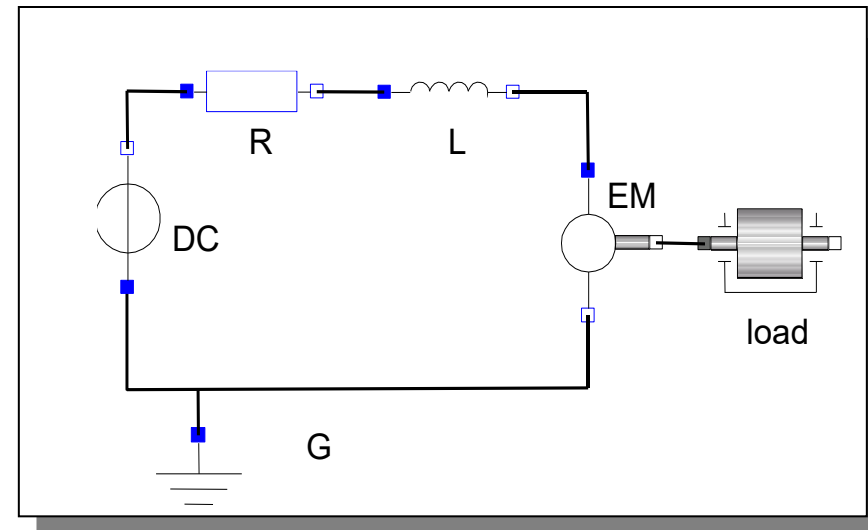
```
connect(ac.flange_b, iner.flange_a); connect(iner.flange_b, emf.flange_b);  
connect(emf.p, ind.p); connect(ind.n, R1.p); connect(emf.n, G.p);  
connect(emf.n, R2.n); connect(R1.n, R2.p); connect(R2.p, vsens.n);  
connect(R2.n, vsens.p); connect(ex.outPort, ac.inPort);
```

```
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

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



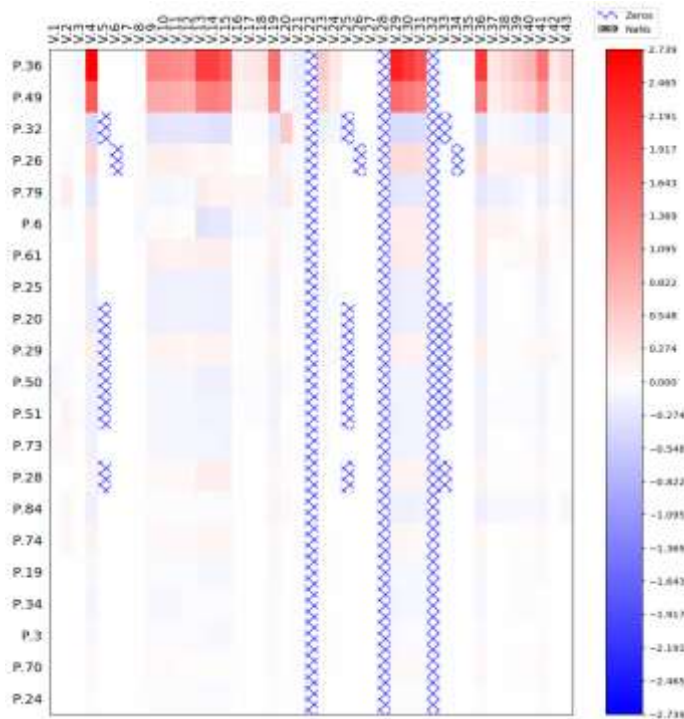
Approaches to Sensitivity Analysis

- **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

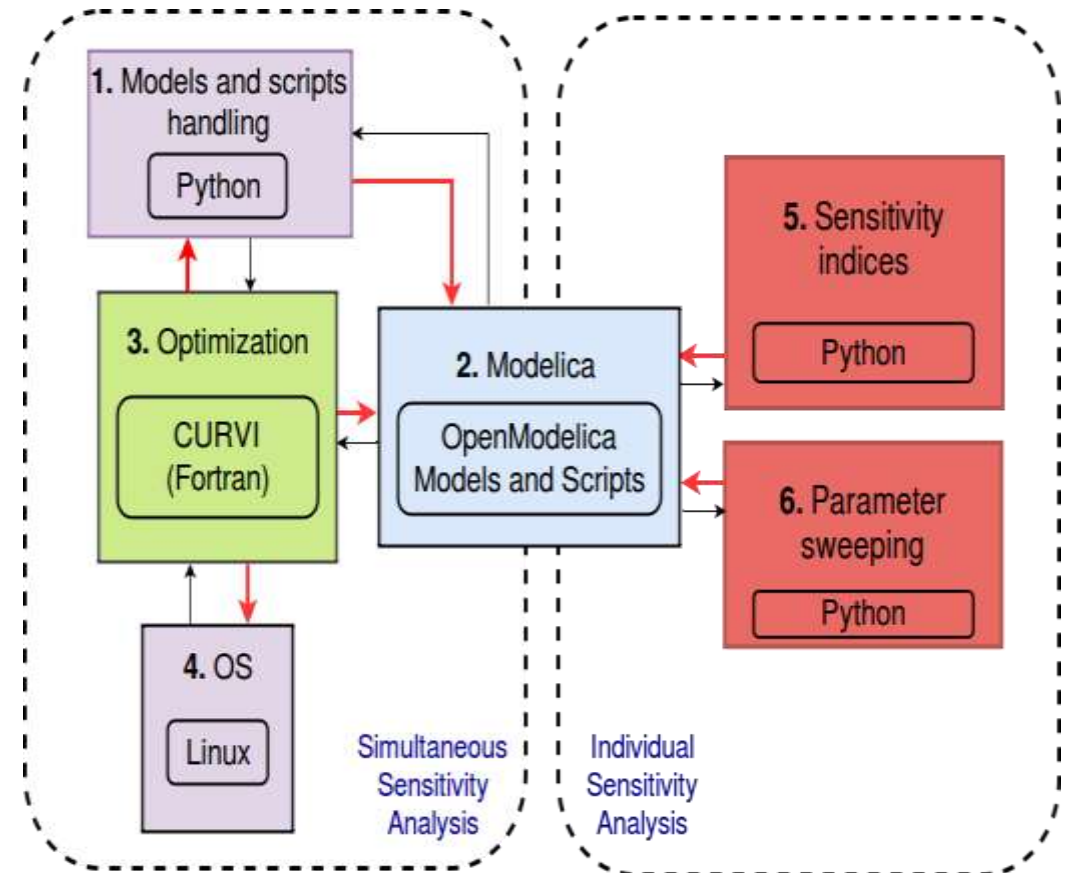
OMSens – Multi-Parameter Sensitivity Analysis

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

Heatmap visualization



Tool architecture



CURVIF: robust derivative-free optimization algorithm

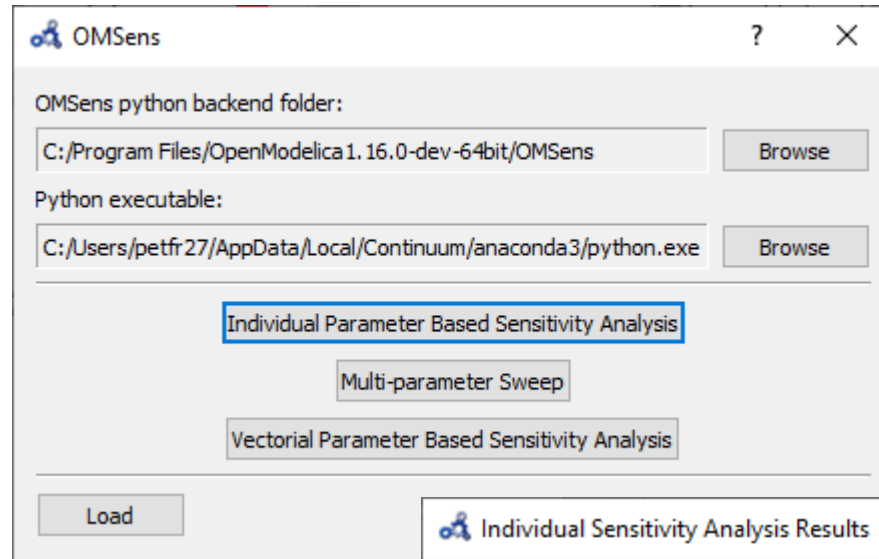
- The CURVI family
 - Curvilinear search approach
- Three versions: CURVIF, CURVIG, CURVIH
 - Function values, function values plus Gradients, and the latter plus Hessians.
 - Globally convergent
 - In general uses **fewer evaluations** than other algorithms
- CURVIF: the flavor adopted for OMSens
 - 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(prey_pop) = prey_pop*(alpha-beta*pred_pop);  
  der(pred_pop) = pred_pop*(delta*prey_pop-gamma);  
end LotkaVolterra ;
```

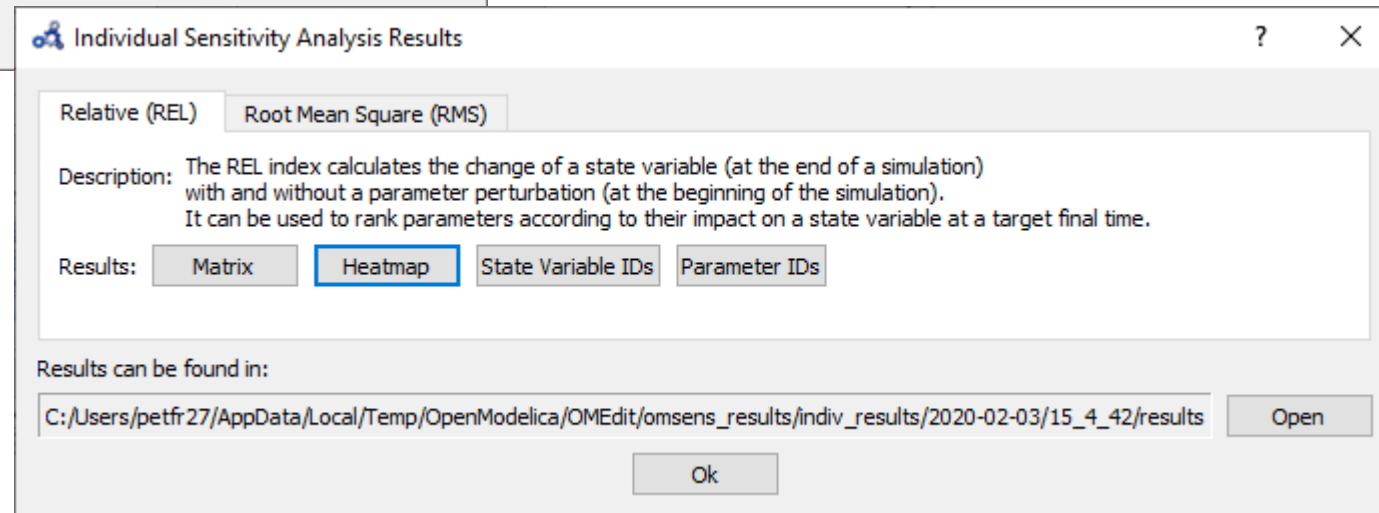
OMSens Exercise – Locate Python

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:

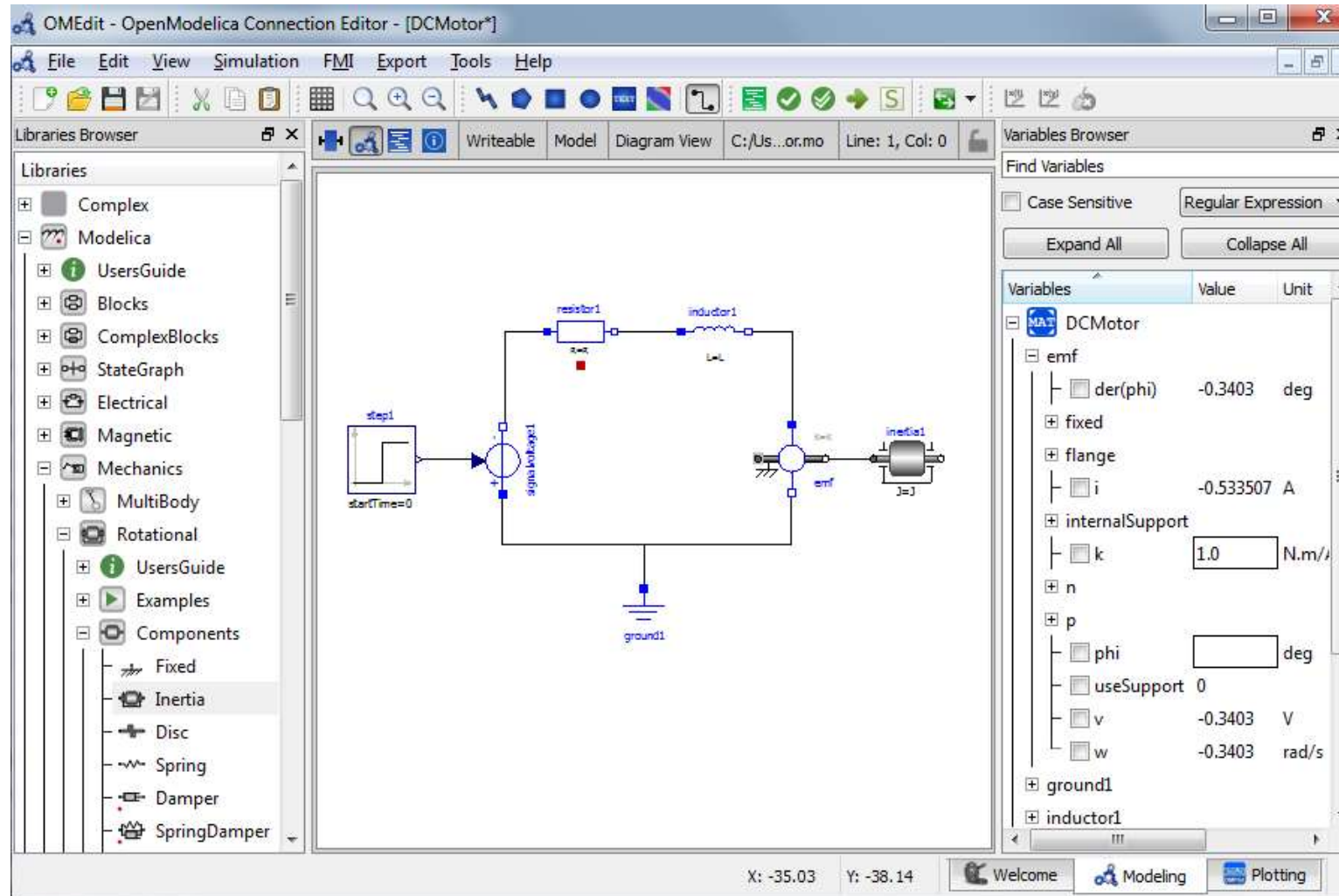
OMSens Example_Exercise_ Lotka-Volterra.pdf



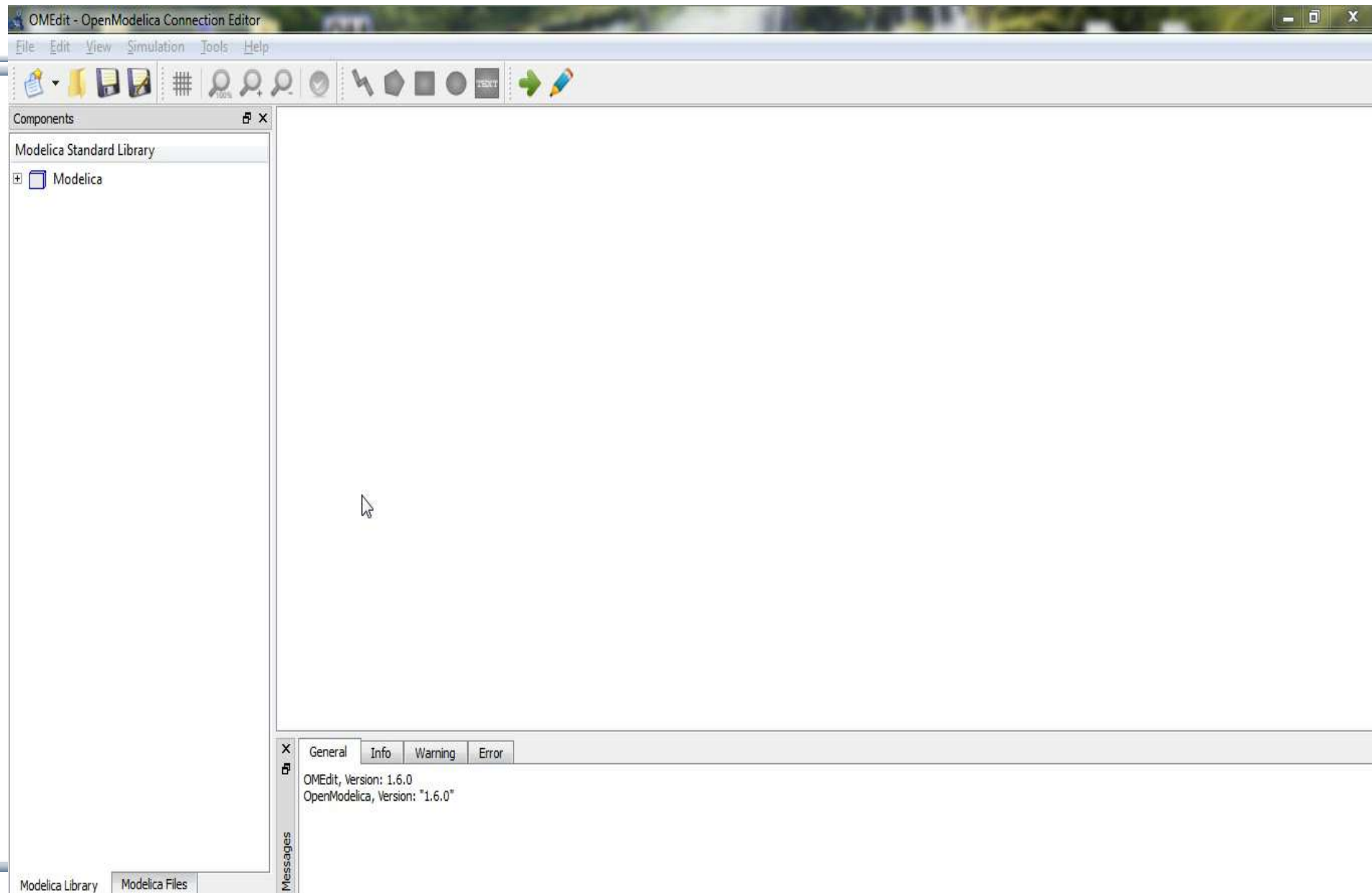
Part Vb
More
Graphical Modeling Exercises

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);

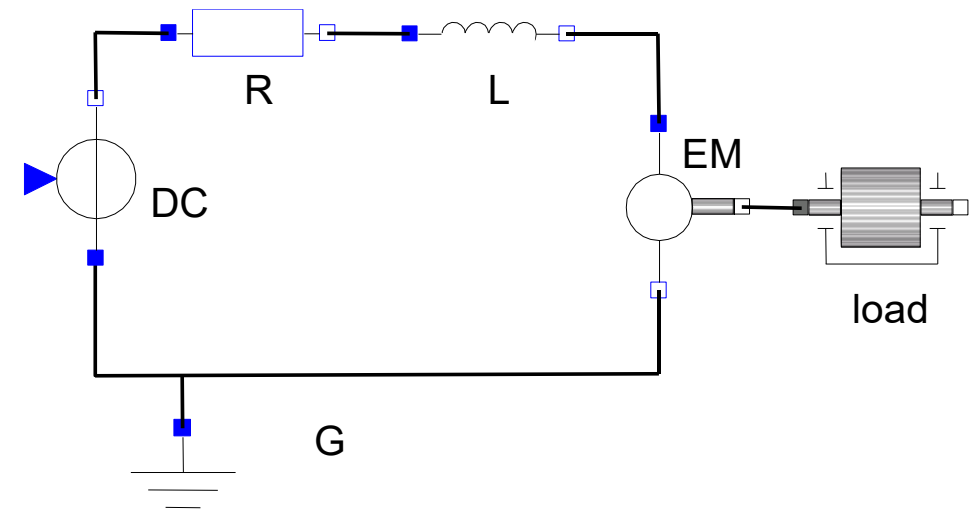
connect (L.p, EM.n);

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		

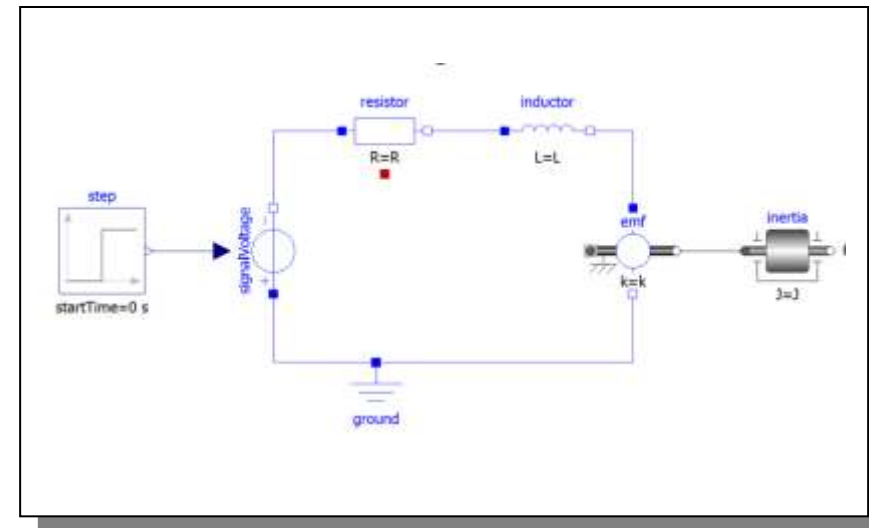
(load component not included)

Automatic transformation to ODE or DAE for simulation:

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

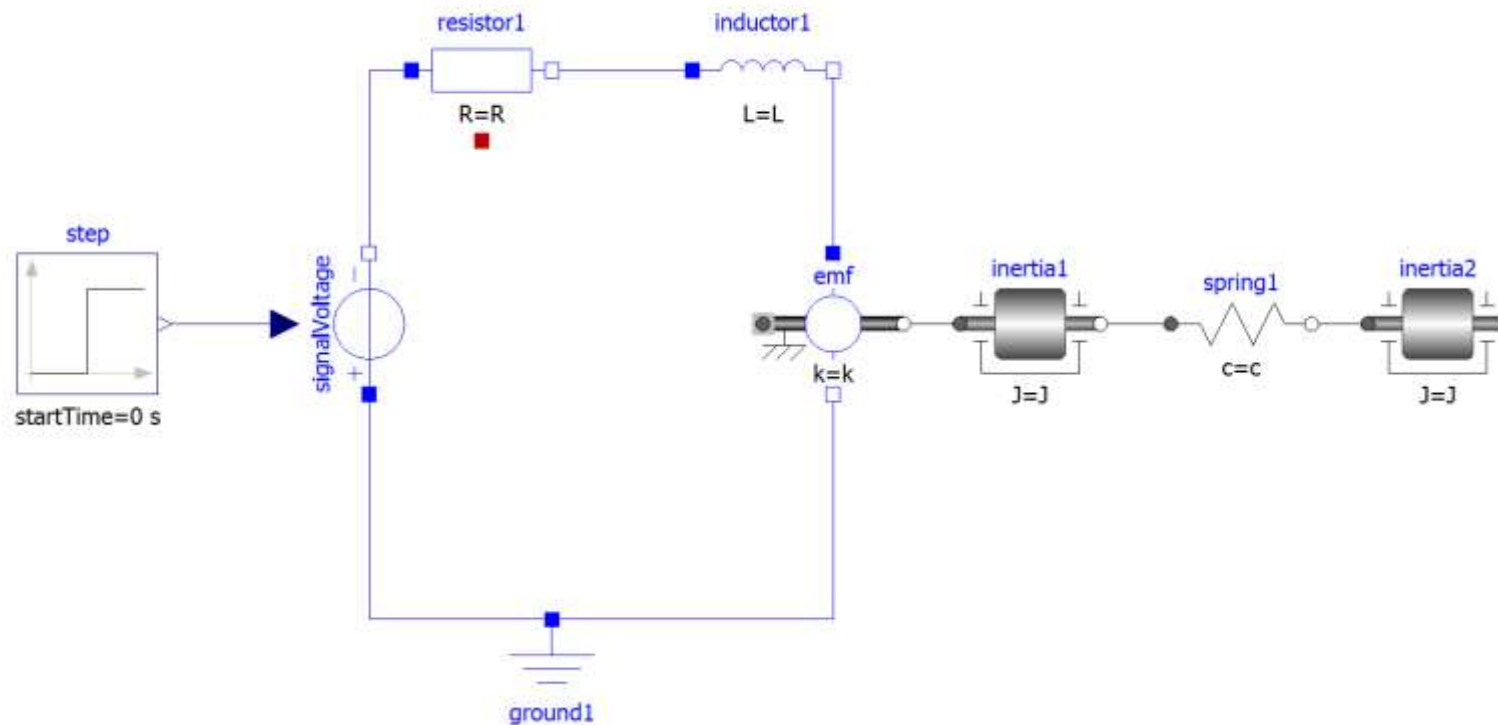
Exercise 3.1

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



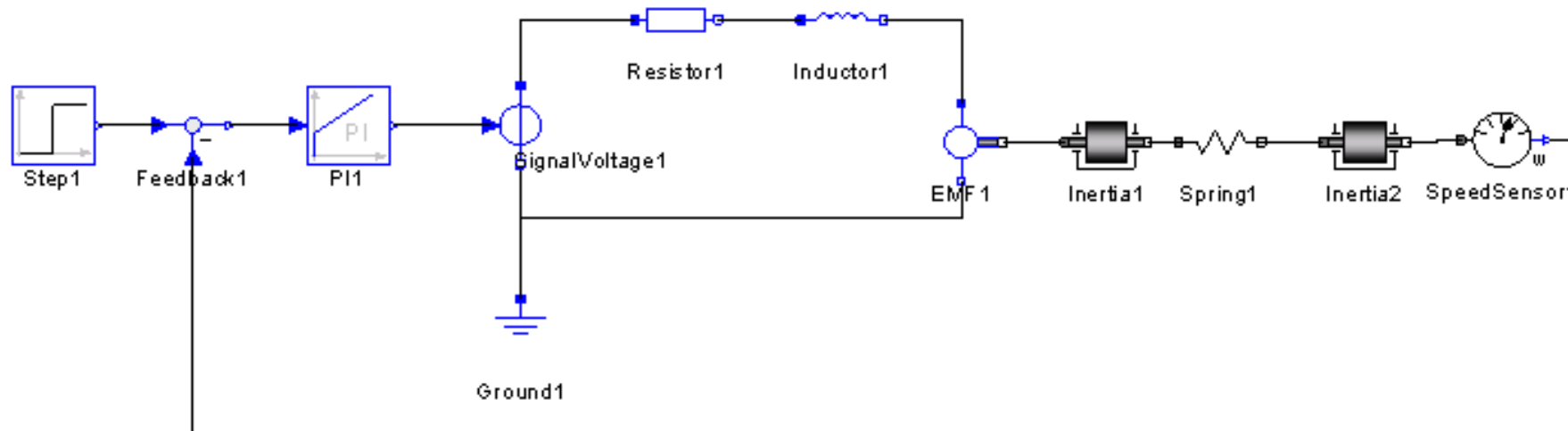
Exercise 3.2

- 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.

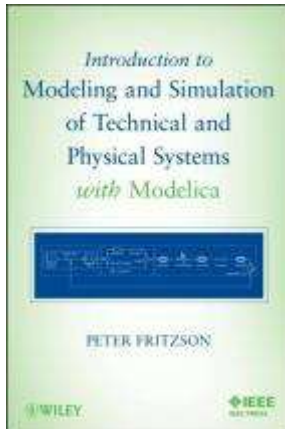
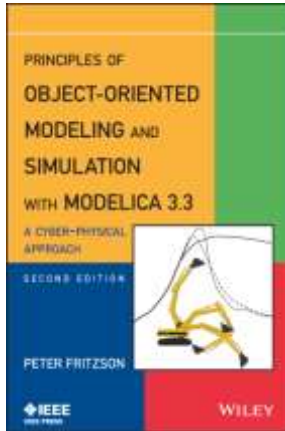


Exercise 3.3

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

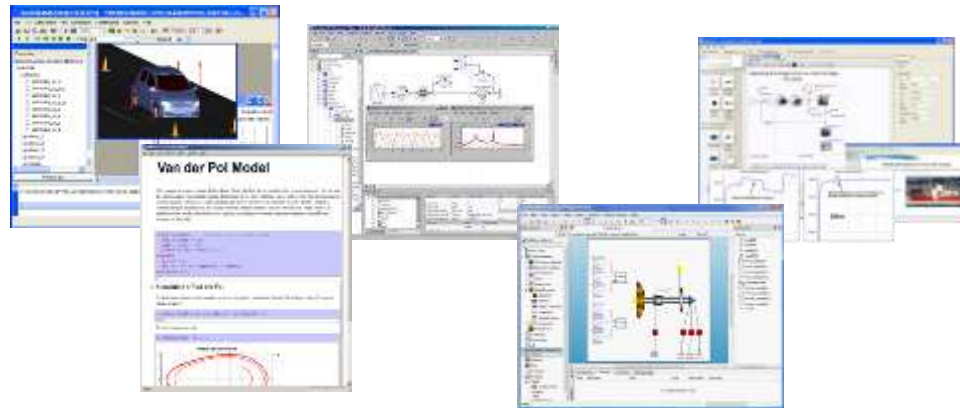
Summary

Multi-Domain
Modeling

Visual Acausal
Component
Modeling



www.OpenModelica.org
OpenModelica@ida.liu.se



Typed
Declarative
Textual Language

Hybrid
Modeling

Thank you for listening!