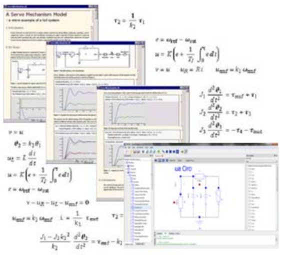
Introduction to Object-Oriented Modeling, Simulation, Debugging and Dynamic Optimization with Modelica using OpenModelica

MOSES 2016 Workshop



Tutorial, Version May 18, 2016

Peter Fritzson
Linköping University, peter.fritzson@liu.se
Director of the Open Source Modelica Consortium
Vice Chairman of Modelica Association

Bernhard Thiele, Ph.D., bernhard.thiele@liu.se
Researcher at PELAB, Linköping University

Slides
Based on book and lecture notes by Peter Fritzson
Contributions 2004-2005 by Emma Larsdotter Nilsson, Peter Bunus
Contributions 2006-2008 by Adrian Pop and Peter Fritzson
Contributions 2009 by David Broman, Peter Fritzson, Jan Brugård, and Mohsen Torabzadeh-Tari
Contributions 2010 by Peter Fritzson
Contributions 2011 by Peter F., Mohsen T,. Adeel Asghar,
Contributions 2012, 2013, 2014, 2015, 2016 by Peter Fritzson, Lena Buffoni, Mahder Gebremedhin, Bernhard Thiele

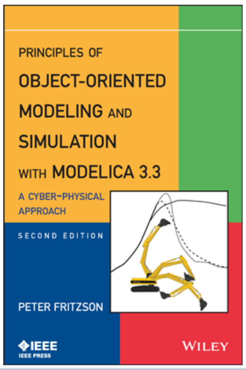
LINKÖPING UNIVERSITY

2016-05-16

Modelica/OpenModelica Tutorial Plan for the MOSES 2016 Workshop

- Wednesday slot 24. The Modelica language part 1. Introductory hands-on Modelica modeling.
  Slides 8 – 49
- Thursday slot 28. The OpenModelica tool. (slides 50 – 78) The Modelica language part 2. (slides 95-115)
- Thursday slot 29. Hands-on with Modelica textual equation-based modeling (slides 95-115)

Tutorial Based on Book, Decembr 2014 Download OpenModelica Software



Peter Fritzson
Principles of Object Oriented Modeling and Simulation with Modelica 3.3
A Cyber-Physical Approach

Can be ordered from Wiley or Amazon

Wiley-IEEE Press, 2014, 1250 pages

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

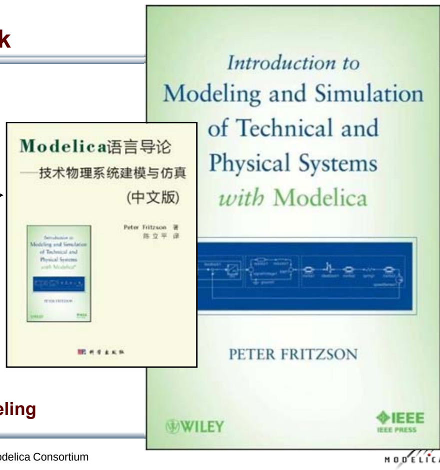
Introductory Modelica Book

September 2011
232 pages

2015 –Translations available in
Chinese,
Japanese,
Spanish

Wiley
IEEE Press

For Introductory Short Courses on Object Oriented Mathematical Modeling



Acknowledgements, Usage, Copyrights

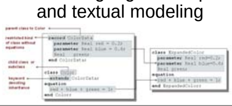
- If you want to use the Powerpoint version of these slides in your own course, send an email to: peter.fritzson@ida.liu.se
- Thanks to Emma Larsdotter Nilsson, Peter Bunus, David Broman, Jan Brugård, Mohsen-Torabzadeh-Tari, Adeel Asghar, Lena Buffoni, for contributions to these slides.
- Most examples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

Outline

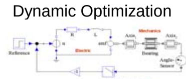
Part I
Introduction to Modelica and a demo example

Part II
Modelica environments

Part III
Modelica language concepts and textual modeling



Part IV and Part V
Graphical modeling and the Modelica standard library
Dynamic Optimization



Detailed Schedule (morning version) 09.00-12.30

09:00 - Introduction to Modeling and Simulation
- Start installation of **OpenModelica** including **OMEdit** graphic editor

09:10 - Modelica – The Next Generation Modeling Language
09:25 - *Exercises Part I (15 minutes)*
- Short hands-on exercise on graphical modeling using **OMEdit**– RL Circuit

09:50 – Part II: Modelica Environments and the OpenModelica Environment
10:10 – Part III: Modelica Textual Modeling
10:15 - *Exercises Part IIIa (10 minutes)*
- Hands-on exerciseson textual modeling using the **OpenModelica** environment

10:25 – Coffee Break
10:40 - Modelica Discrete Events, Hybrid, Clocked Properties (Bernhard Thiele)
11:00- *Exercises Part IIIb (15 minutes)*
- Hands-on exercises on textual modeling using the **OpenModelica** environment

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

11:30 – *Part V* Dynamic Optimization (Bernhard Thiele)
- Hands-on exercise on dynamic optimization using **OpenModelica**

12:00 – Exercise Graphical Modeling DCMotor using OpenModelica

Software Installation - Windows

- Start the software installation
- Install OpenModelica-1.9.4beta.exe from the USB Stick

Software Installation – Linux (requires internet connection)

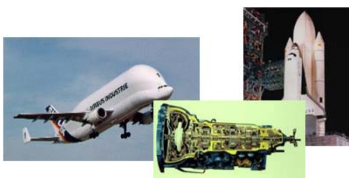
- Go to https://openmodelica.org/index.php/download/download-linux and follow the instructions.

Software Installation – MAC (requires internet connection)

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

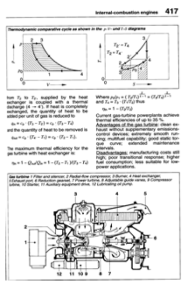
Part I

Introduction to Modelica and a demo example

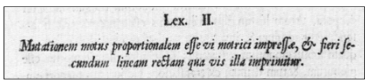


Modelica Background: Stored Knowledge

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

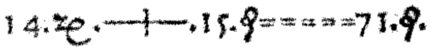


"*The change of motion is proportional to the motive force impressed*"
– Newton


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

Modelica Background: The Form – Equations

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



Newton still wrote text (Principia, vol. 1, 1686)
"The change of motion is proportional to the motive force impressed"
CSSL (1967) introduced a special form of "equation":

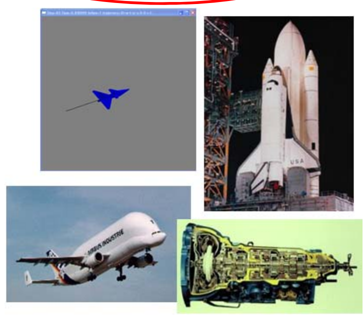
```
variable = expression
v = INTEG(F)/m
```

Programming languages usually do not allow equations!

What is Modelica?

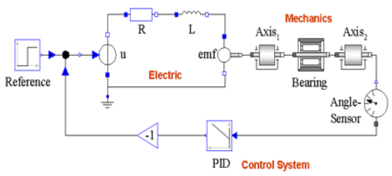
A language for modeling of complex cyber-physical systems

- 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 several free and commercial tools, for example:

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

Modelica – The Next Generation Modeling Language

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

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

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

Visual component programming
Hierarchical system architecture capabilities

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

Modelica Acausal Modeling

What is *acausal* modeling/design?

Why does it increase *reuse*?

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

Example: a resistor *equation*:

R*i = v;

can be used in three ways:

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

What is Special about Modelica?

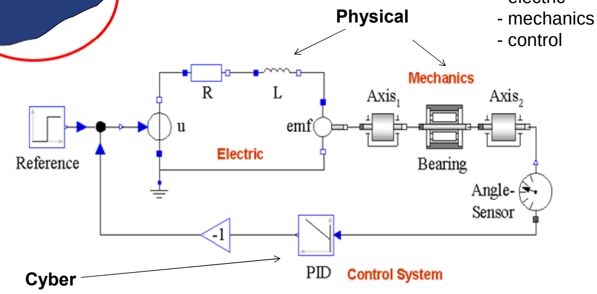
- Multi-Domain Modeling
- Visual acausal hierarchical component modeling
- Typed declarative equation-based textual language
- Hybrid modeling and simulation

What is Special about Modelica?

Multi-Domain Modeling

Cyber-Physical Modeling

3 domains
- electric
- mechanics
- control



What is Special about Modelica?

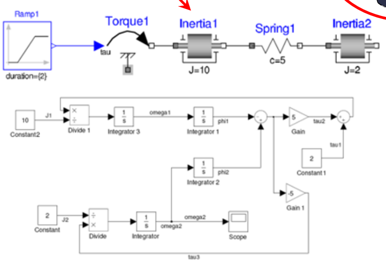
Multi-Domain Modeling

Visual Acausal Hierarchical Component Modeling

Keeps the physical structure

Acausal model (Modelica)

Causal block-based model (Simulink)

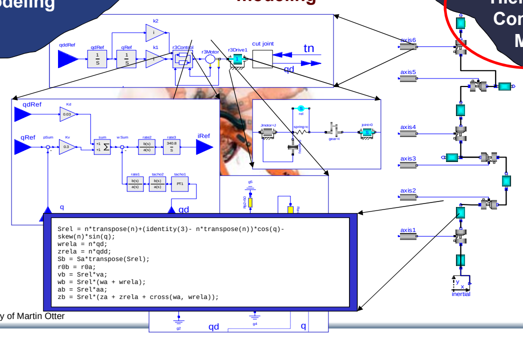


What is Special about Modelica?

Multi-Domain Modeling

Hierarchical system modeling

Visual Acausal Hierarchical Component Modeling



Courtesy of Martin Otter

What is Special about Modelica?

Multi-Domain Modeling

Visual Acausal Hierarchical Component Modeling

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

Behaviour described declaratively using
- Differential algebraic equations (DAE) (continuous-time)
- Event triggers (discrete-time)

Variable declarations

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

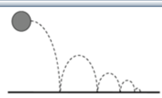
Differential equations

Typed Declarative Equation-based Textual Language

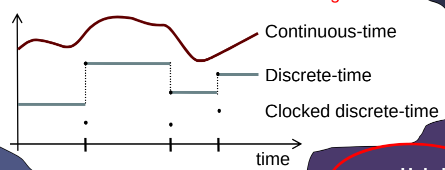
What is Special about Modelica?

Multi-Domain Modeling

Visual Acausal Component Modeling



Hybrid modeling = continuous-time + discrete-time modeling

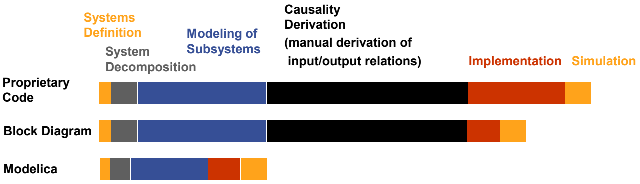


Typed Declarative Equation-based Textual Language

Hybrid Modeling

Modelica – Faster Development, Lower Maintenance than with Traditional Tools

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

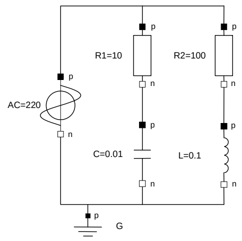


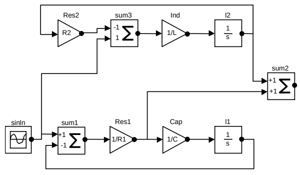
Modelica vs Simulink Block Oriented Modeling Simple Electrical Model

Modelica:
Physical model – easy to understand

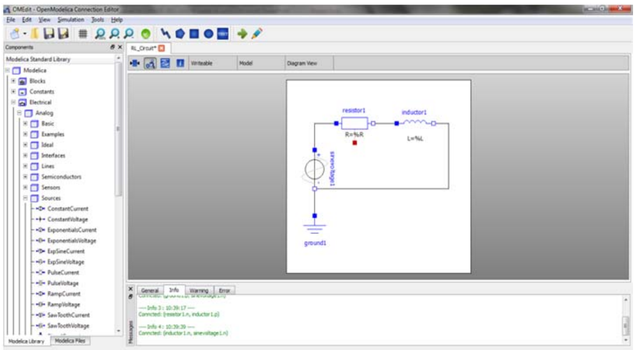
Keeps the physical structure

Simulink:
Signal-flow model – hard to understand





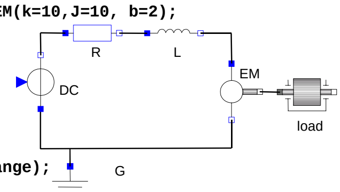
Graphical Modeling - Using Drag and Drop Composition



Multi-Domain (Electro-Mechanical) Modelica Model

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

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



Corresponding DCMotor Model Equations

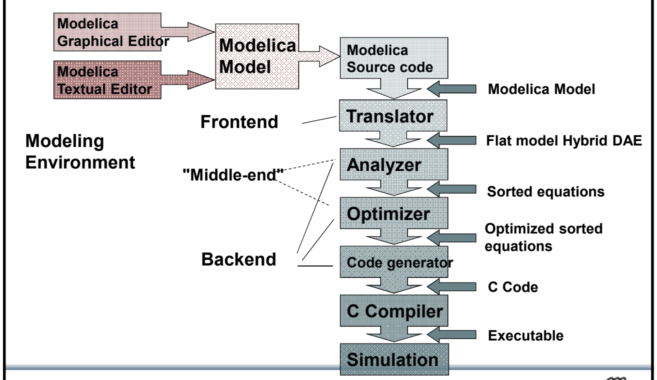
The following equations are automatically derived from the Modelica model:

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

0 == DC.n.i + G.p.i
DC.n.v == G.p.v                        (load component not included)
```

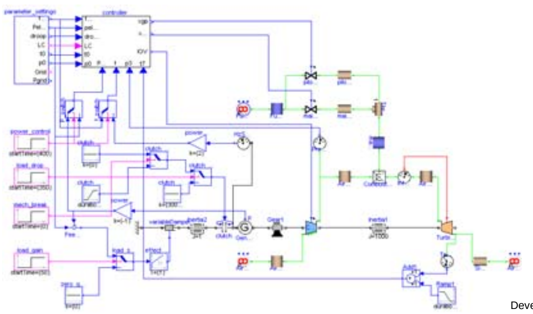
Automatic transformation to ODE or DAE for simulation:

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

Model Translation Process to Hybrid DAE to Code



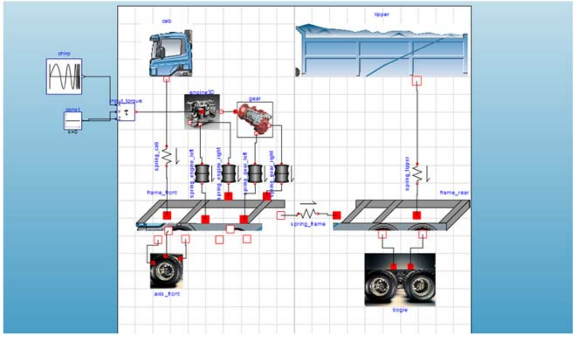
Modelica in Power Generation
GTX Gas Turbine Power Cutoff Mechanism



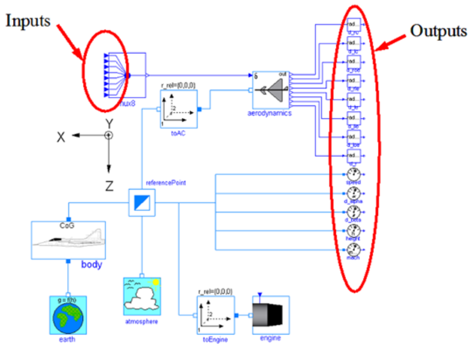
Developed by MathCore for Siemens

Courtesy of Siemens Industrial Turbomachinery AB

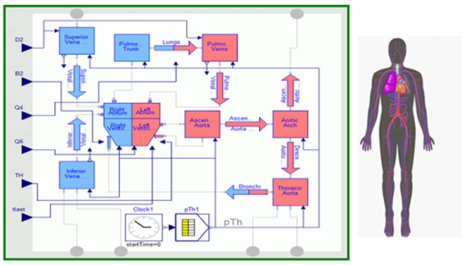
Modelica in Automotive Industry



Modelica in Avionics



Modelica in Biomechanics



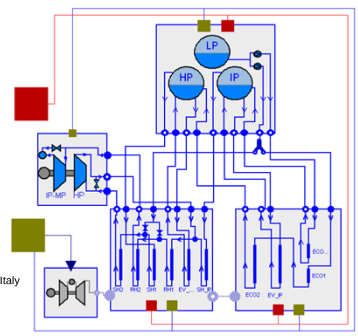
Application of Modelica in Robotics Models
Real-time Training Simulator for Flight, Driving

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

Courtesy of Tobias Bellmann, DLR, Oberphaffenhofen, Germany

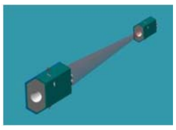
Combined-Cycle Power Plant
Plant model – system level

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



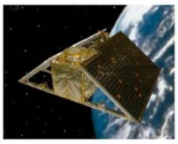
Courtesy Francesco Casella, Politecnico di Milano – Italy and Francesco Pretolani, CESI SpA - Italy

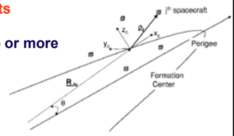
Modelica Spacecraft Dynamics Library



Formation flying on elliptical orbits

Control the relative motion of two or more spacecraft



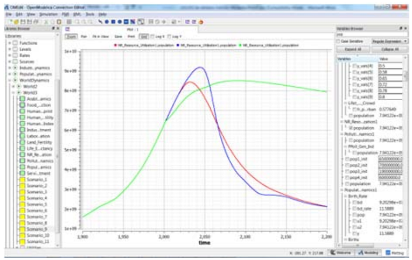


Attitude control for satellites using magnetic coils as actuators

Torque generation mechanism: interaction between coils and geomagnetic field

Courtesy of Francesco Casella, Politecnico di Milano, Italy

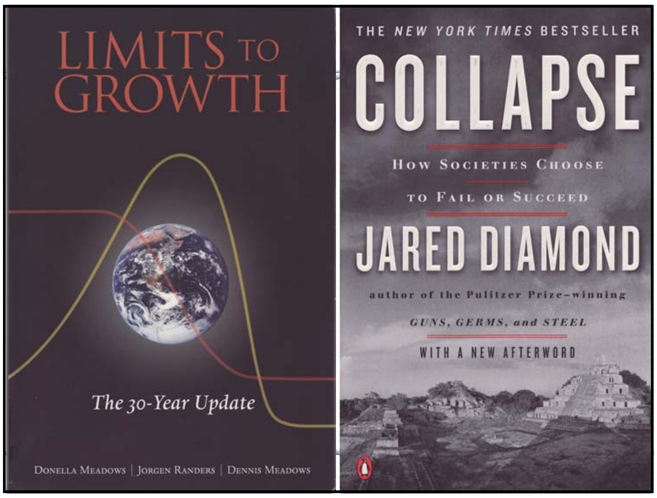
System Dynamics – World Society Simulation
Limits to Material Growth; Population, Energy and Material flows



Left. World3 simulation with OpenModelica
- 2 collapse scenarios (close to current developments)
- 1 sustainable scenario (green).

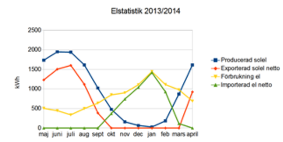
CO_2 Emissions per person:
- USA 17 ton/yr
- Sweden 7 ton/yr
- India 1.4 ton/yr
- Bangladesh 0.3 ton/yr

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



What Can You Do?
Need Global Sustainability Mass Movement

- Book: Current catastrophic scenarios: ***Mark Lynas***: **"6 Degrees"**
  Book: How to address the problems: ***Tim Jackson*** **"Prosperity without Growth"**
- 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 CO_2 emission per year

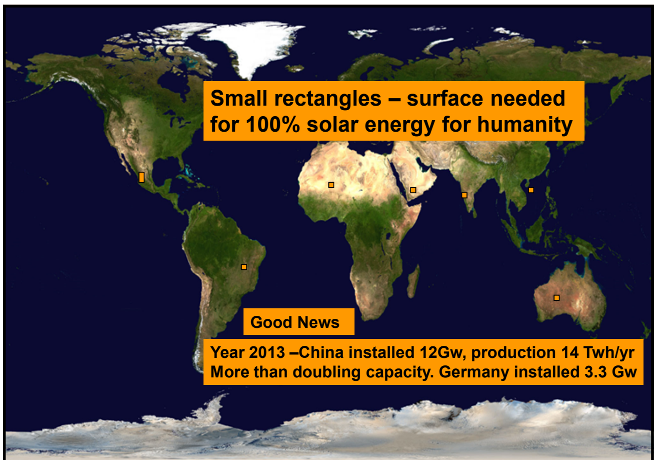
Example Electric Cars
Can be charged by electricity from own solar panels

Renault ZOE; 5 seat; Range:
- **EU-drive cycle 210 km**
- **Realistic Swedish drive cycle:**
- **Summer: 165 km**
- **Winter: 100 – 110 km**

Cheap fast supercharger

DLR ROboMObil
- **experimental electric car**
- **Modelica models**

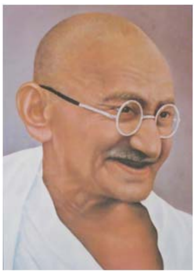
Tesla model S range 480 km



Sustainable Society Necessary for Human Survival

Almost Sustainable
- India, 1.4 ton C02/person/year
- Healthy vegetarian food
- Small-scale agriculture
- Small-scale shops
- Simpler life-style (Mahatma Gandhi)

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



Gandhi – role model for future less materialistic life style

Brief Modelica History

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

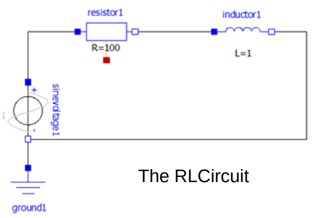
Modelica Conferences

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

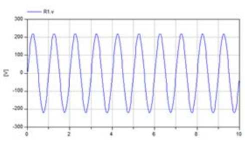
Exercises Part I
Hands-on graphical modeling
(15 minutes)

Exercises Part I – Basic Graphical Modeling

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



The RLCircuit



Simulation

Exercises Part I – OMEdit Instructions (Part I)

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

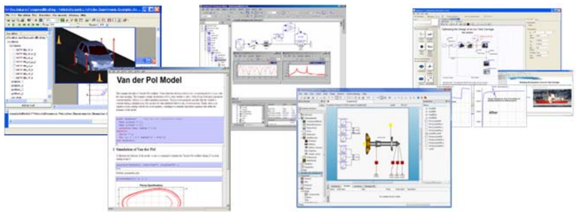
- Under the **Modelica Library:**
  - Contains The standard Modelica library components
  - The **Modelica files** contains the list of models you have created.

Exercises Part I – OMEdit Instructions (Part II)

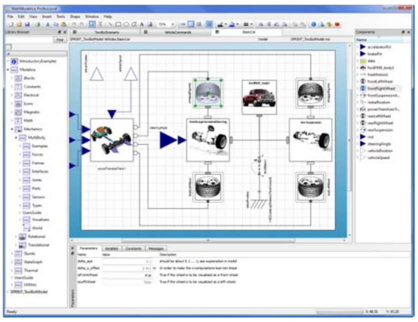
- For the RLCircuit model, browse the Modelica standard library and add the following component models:
  - Add `Ground`, `Inductor` and `Resistor` component models from Modelica.Electrical.Analog.Basic package.
  - Add SineVoltage component model from Modelica.Electrical.Analog.Sources package.
- Make the corresponding connections between the component models as shown in slide 38.
- Simulate the model
  - Go to Simulation menu and choose simulate or click on the simulate button in the toolbar.
- Plot the instance variables
  - Once the simulation is completed, a plot variables list will appear on the right side. Select the variable that you want to plot.

Part II

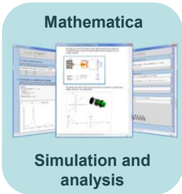
Modelica environments and OpenModelica



Wolfram System Modeler – Wolfram MathCore



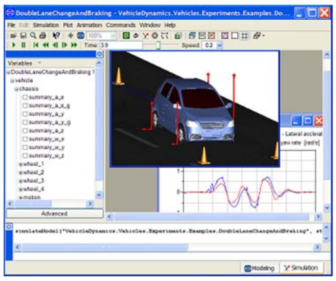
- Wolfram Research
- USA, Sweden
- General purpose
- Mathematica integration
- www.wolfram.com
- www.mathcore.com



Courtesy Wolfram Research

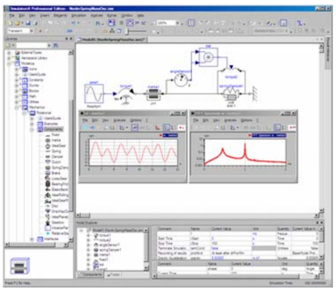
Car model graphical view

Dymola



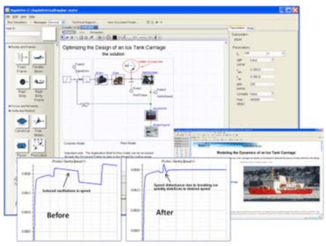
- Dassault Systemes Sweden
- Sweden
- First Modelica tool on the market
- Initial main focus on automotive industry
- www.dymola.com

Simulation X



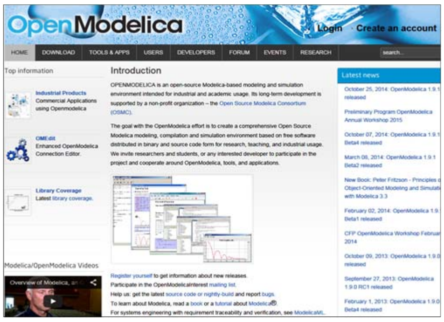
- ITI Gmbh (Just bought by ESI Group)
- Germany
- Mechatronic systems
- www.simulationx.com

MapleSim



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

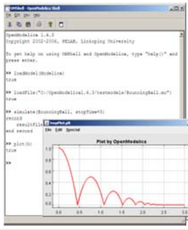
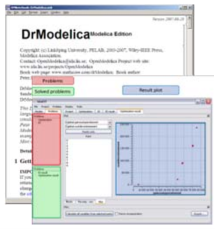
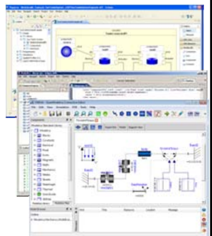
The OpenModelica Environment
www.OpenModelica.org



The OpenModelica Open Source Environment
www.openmodelica.org

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

- **OMEdit** graphic Editor
- **OMDebugger** for equations
- **OMOptim** optimization tool
- **OM Dynamic optimizer** collocation
- **ModelicaML** UML Profile
- **MetaModelica** extension
- **ParModelica** extension

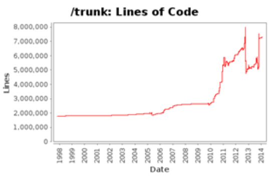




OSMC – International Consortium for Open Source Model-based Development Tools, 48 members Jan 2016

Founded Dec 4, 2007

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

Code Statistics



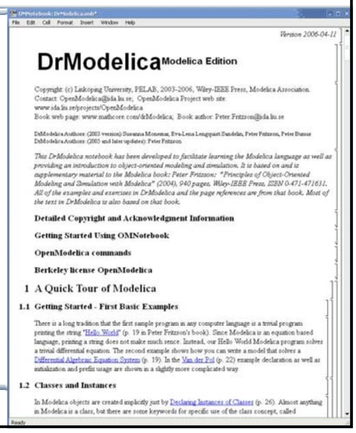
Industrial members
- **ABB AB, Sweden**
- **Bosch Rexroth AG, Germany**
- **Siemens Turbo, Sweden**
- **CDAC Centre, Kerala, India**
- **Creative Connections, Prague**
- **DHI, Aarhus, Denmark**
- **Dynamica s.r.l., Cremona, Italy**
- **EDF, Paris, France**
- **Equa Simulation AB, Sweden**
- **Fraunhofer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

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

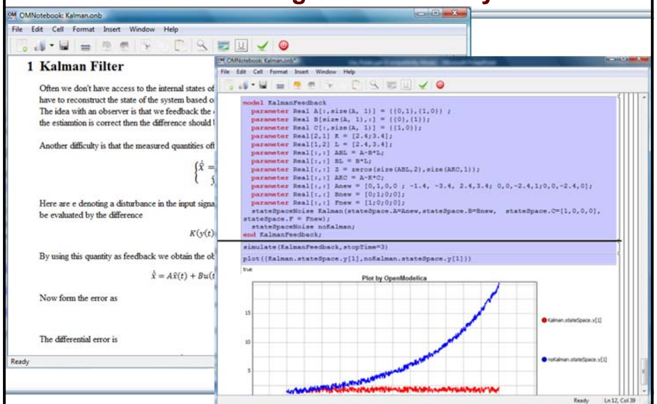
OMNotebook Electronic Notebook with DrModelica

- Primarily for teaching
- Interactive electronic book
- Platform independent

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

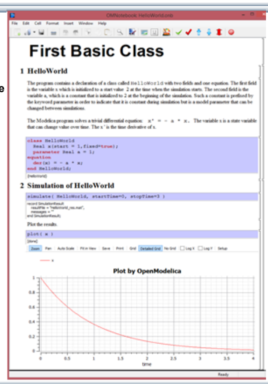
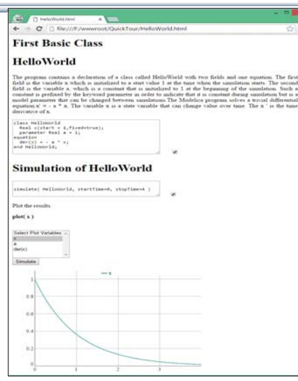


OMnotebook Interactive Electronic Notebook Here Used for Teaching Control Theory



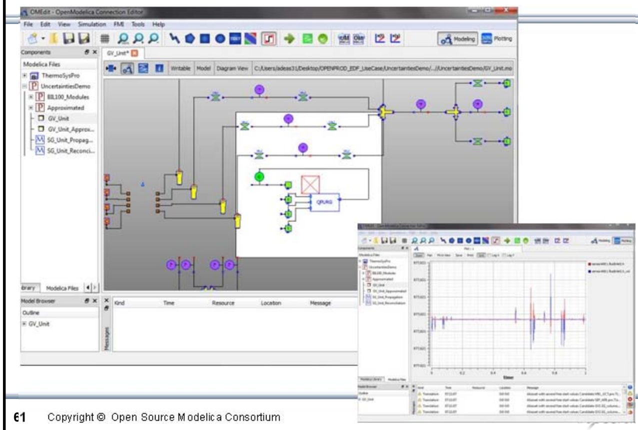
OM Web Notebook Generated from OMNotebook
Edit, Simulate, and Plot Models on a Web Page

OMNotebook

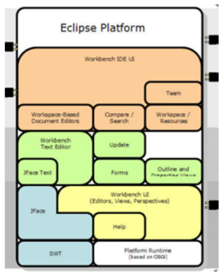
OMwebbook

OpenModelica Environment Demo

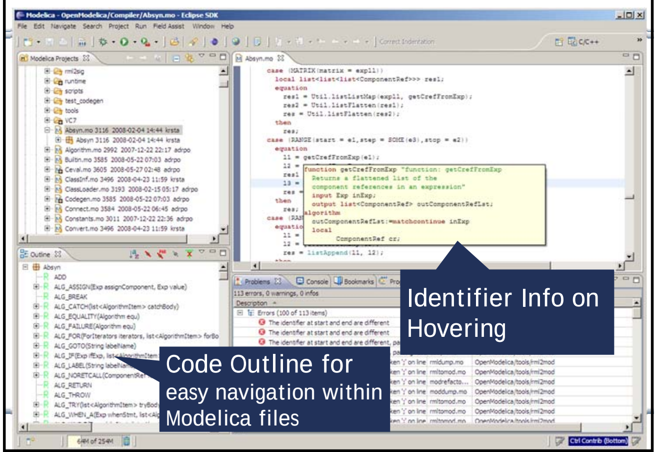


OpenModelica MDT – Eclipse Plugin

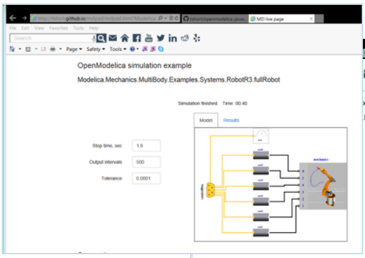
- Browsing of packages, classes, functions
- Automatic building of executables; separate compilation
- Syntax highlighting
- Code completion, Code query support for developers
- Automatic Indentation
- Debugger (Prel. version for algorithmic subset)



OpenModelica MDT: Code Outline and Hovering Info



OpenModelica Simulation in Web Browser Client



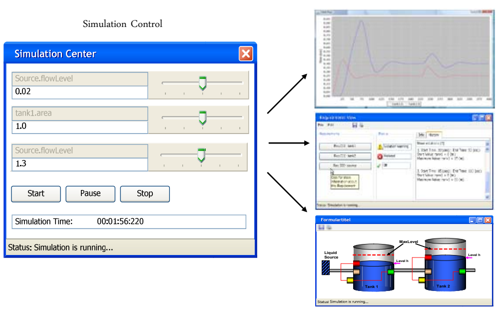
MultiBody RobotR3.FullRobot

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

Interactive Simulation

Examples of Simulation Visualization

Simulation Control



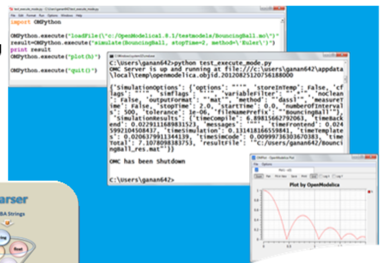
Plot View

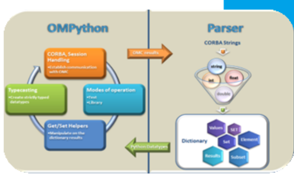
Requirements Evaluation View in ModelicaML

Domain-Specific Visualization View

OMPython – Python Scripting with OpenModelica

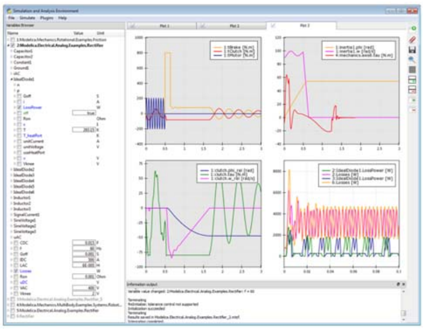
- Interpretation of Modelica commands and expressions
- Interactive Session handling
- Library / Tool
- Optimized Parser results
- Helper functions
- Deployable, Extensible and Distributable





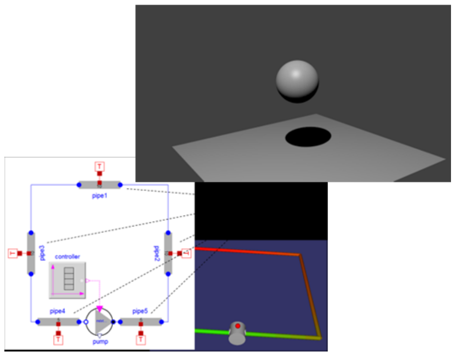
PySimulator Package

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



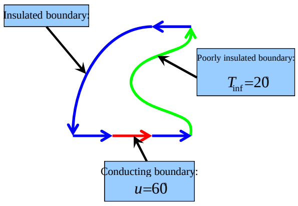
Modelica3D Library

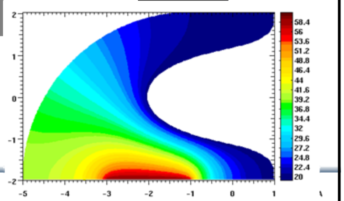
- Modelica 3D Graphics Library by Fraunhofer FIRST, Berlin
- Part of OpenModelica distribution
- Can be used for 3D graphics in OpenModelica



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

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

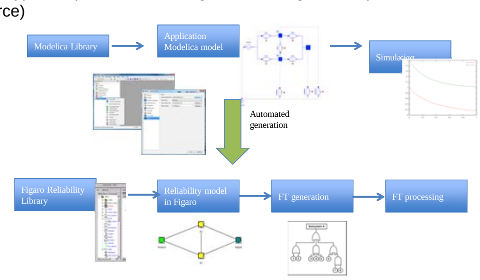




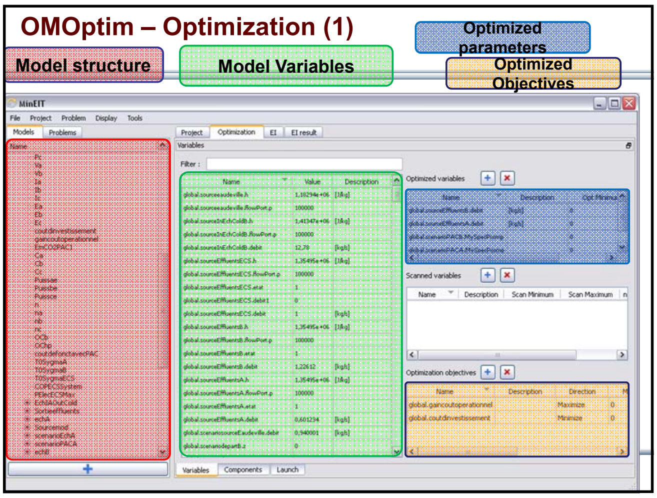
Prototype in OpenModelica 2005
PhD Thesis by Levon Saldamli
www.openmodelica.org
Currently not operational

Failure Mode and Effects Analysis (FMEA) in OM

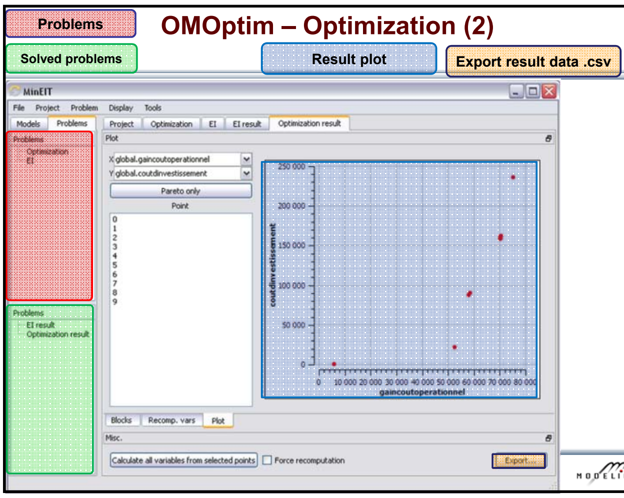
- Modelica models augmented with reliability properties can be used to generate reliability models in Figaro, which in turn can be used for static reliability analysis
- Prototype in OpenModelica integrated with Figaro tool (which is becoming open-source)



OMOptim – Optimization (1)



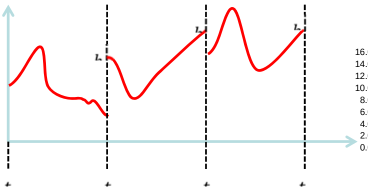
OMOptim – Optimization (2)



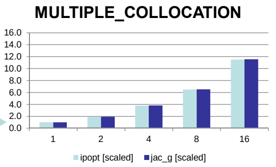
Multiple-Shooting and Collocation Dynamic Trajectory Optimization

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

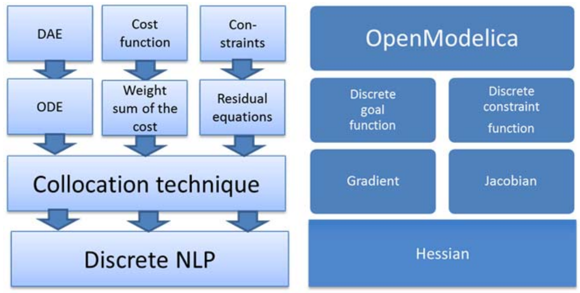
$$x_i(t_{i+1}) = h_i + \int_{t_i}^{t_{i+1}} f(x_i(t),u(t),t)\,dt \approx F(t_i,t_{i+1},h_i,u_i), \qquad x_i(t_i) = h_i$$



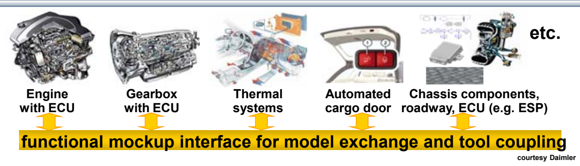
Example speedup, 16 cores:



OpenModelica Dynamic Optimization Collocation



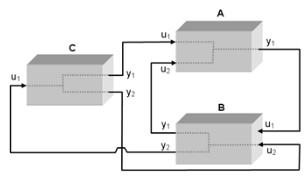
General Tool Interoperability & Model Exchange Functional Mock-up Interface (FMI)



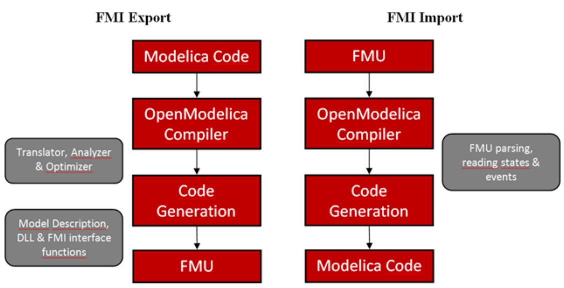
- FMI development was started by ITEA2 MODELISAR project. FMI is a Modelica Association Project now
- **Version 1.0**
- FMI for Model Exchange (released Jan 26,2010)
- FMI for Co-Simulation (released Oct 12,2010)
- **Version 2.0**
- FMI for Model Exchange and Co-Simulation (released July 25,2014)
- **>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

Functional Mockup Units

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



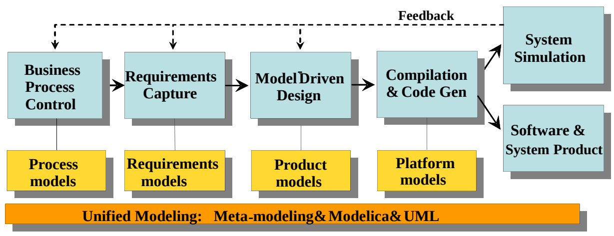
OpenModelica Functional Mockup Interface (FMI)



FMI in OpenModelica

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

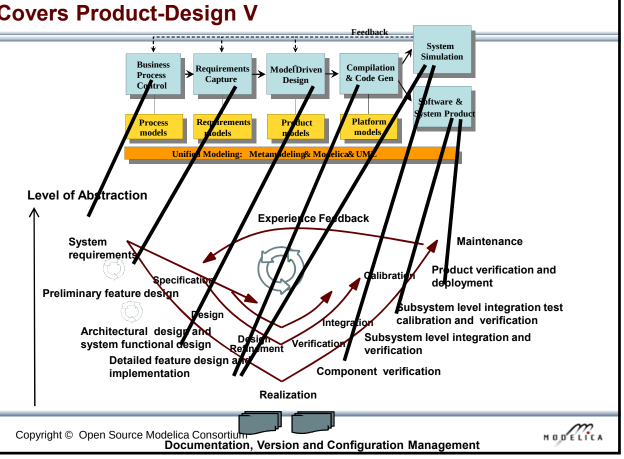
OPENPROD – Large 28-partner European Project, 2009-2012
Vision of Cyber-Physical Model-Based Product Development



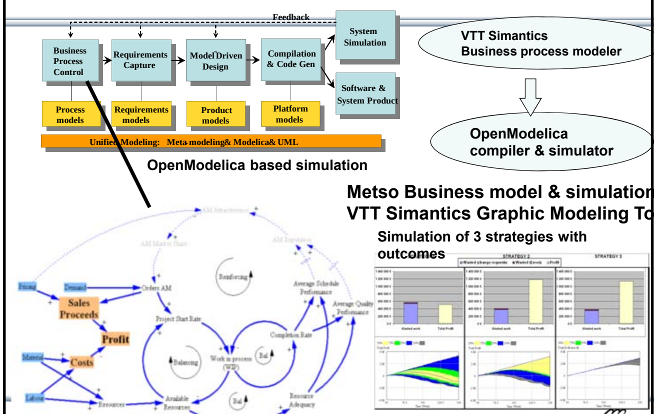
OPENPROD Vision of unified modeling framework for model-based product development.

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

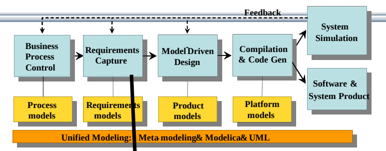
OPENPROD Model-Based Development Environment Covers Product-Design V



Business Process Control and Modeling



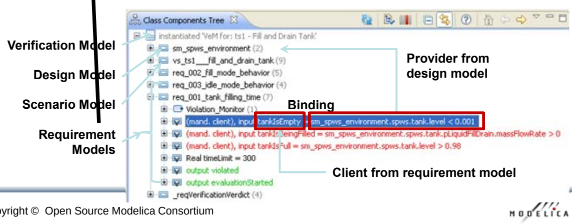
Requirement Capture



vVDR (virtual Verification of Designs against Requirements)

in ModelicaML UML/Modelica Profile, part of OpenModelica

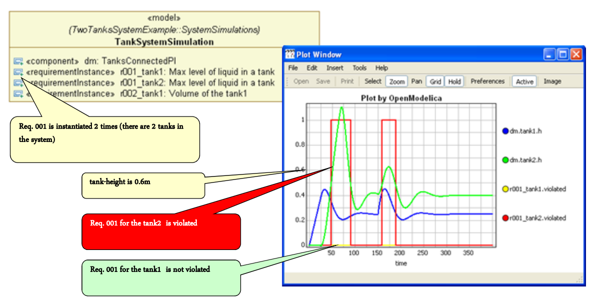
OpenModelica based simulation



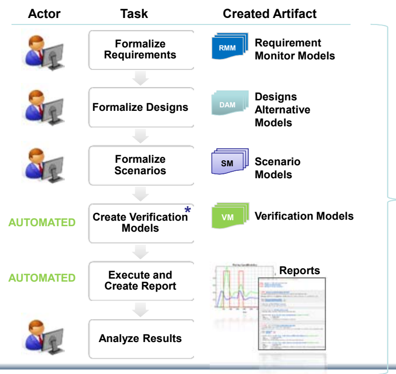
OpenModelica – ModelicaML UML Profile
SysML/UML to Modelica OMG Standardization

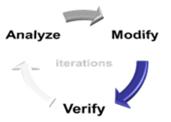
- ModelicaML is a UML Profile for SW/HW modeling
  - Applicable to "pure" UML or to other UML profiles, e.g. SysML
- Standardized Mapping UML/SysML to Modelica
  - Defines transformation/mapping for **executable** models
  - Being **standardized** by OMG
- ModelicaML
  - Defines graphical concrete syntax (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

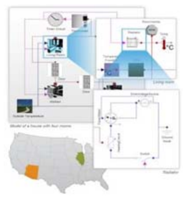




Goal: Enable on-demand verification of designs against requirements using automated model composition at any time during development.

Industrial Product with OEM Usage of OpenModelica

- **The Wolfram SystemModeler** modeling and simulation product by Wolfram, www.wolfram.com
- Includes a large part of the OpenModelica compiler using the OSMC OEM license.
- Images show a house heating application and an excavator dynamics simulation.



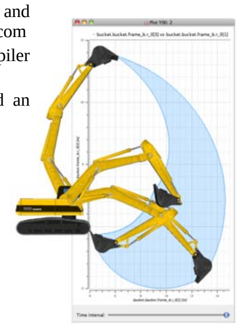


ABB Industry Use of OpenModelica FMI 2.0 and Debugger

- 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.*"

Performance Profiling
(Below: 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 platforms
- Integrated with the debugger so it is possible to show what the slow equations compute
- Suitable for real-time profiling (less information), or a complete view of all equation blocks and function calls

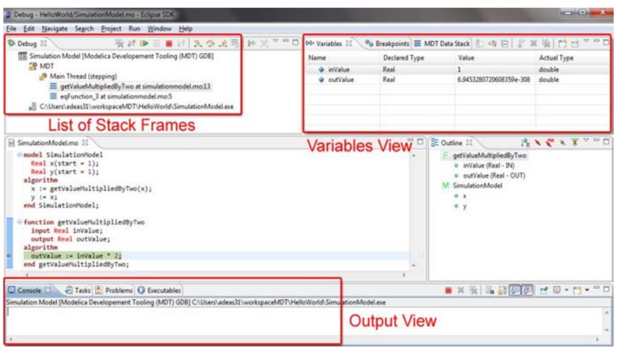
Equations Browser

Index	Type	Equation	Executi	Max time	Time	Fraction
876	regular	linear, size 2	4602	0.000501	0.0134	75.7%
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%

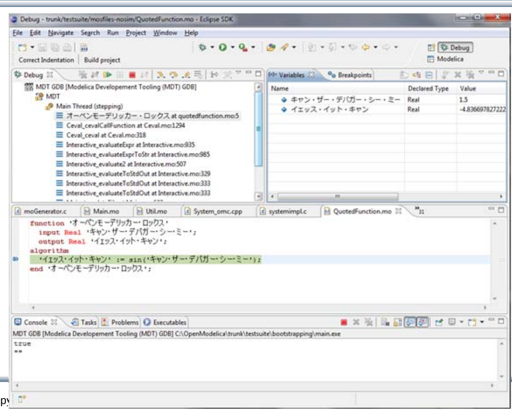
Defines

Variable
damper.a_rel
revolute2.frame_b.f[2]

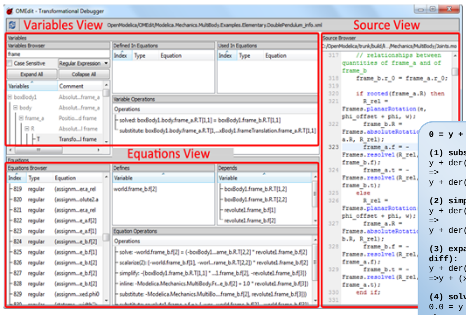
OpenModelica MDT Algorithmic Code Debugger



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



OpenModelica Equation Model Debugger

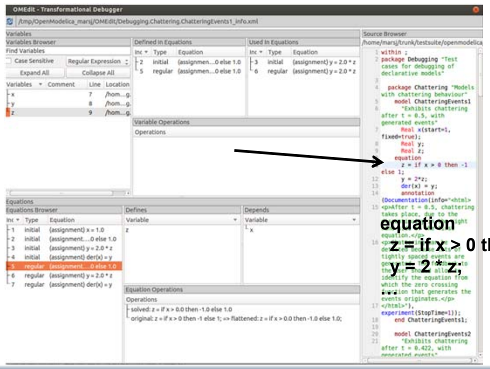


Showing equation transformations of a model:

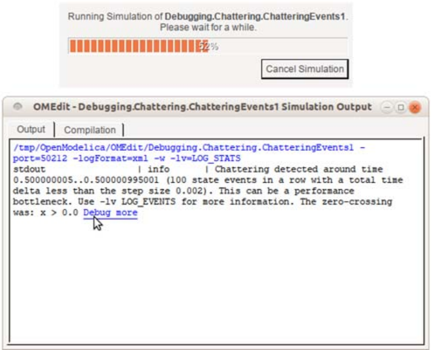
```
0 = y + der(x * time * z); z = 1.0;

(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

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



Error Indication – Simulation Slows Down

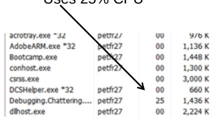


Exercise 1.2 – Equation-based Model Debugger

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

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

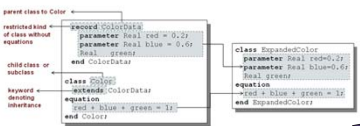
Uses 25% CPU



- 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

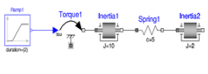
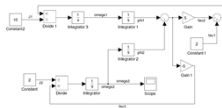


Typed Declarative Equation-based Textual Language

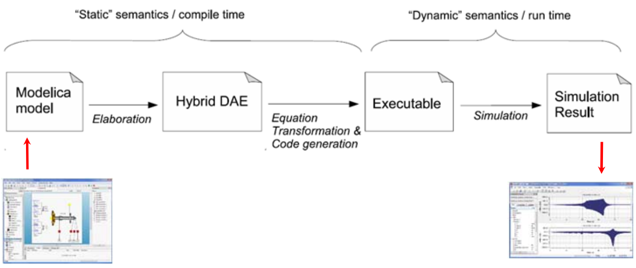
Hybrid Modeling

Acausal Modeling

The order of computations is not decided at modeling time

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

Typical Simulation Process



Simple model - Hello World!

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

Name of model

Initial condition

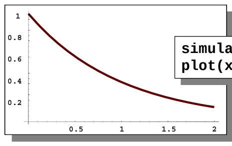
Continuous-time variable

Parameter, constant during simulation

Differential equation

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

Simulation in OpenModelica environment




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


Modelica Variables and Constants

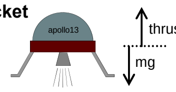
- Built-in primitive data types
  - **Boolean** **true** or **false**
  - **Integer** Integer value, e.g. **42** or **–3**
  - **Real** Floating point value, e.g. **2.4e-6**
  - **String** String, e.g. **"Hello world"**
  - **Enumeration** Enumeration literal e.g. **ShirtSize.Medium**

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

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


A Simple Rocket Model

Rocket



$$acceleration = \frac{thrust - mass \cdot gravity}{mass}$$
$$mass' = -massLossRate \cdot \text{abs}(thrust)$$
$$altitude' = velocity$$
$$velocity' = acceleration$$

new model

parameters (changeable before the simulation)

floating point type

differentiation with regards to time

declaration comment

start value

name + default value

mathematical equation (acausal)

```
class Rocket "rocket class"
  parameter String name;
  Real mass(start=1038.358);
  Real altitude(start= 59404);
  Real velocity(start= -2003);
  Real acceleration;
  Real thrust;  // Thrust force on rocket
  Real gravity; // Gravity forcefield
  parameter Real massLossRate=0.000277;
equation
  (thrust-mass*gravity)/mass = acceleration;
  der(mass)  = -massLossRate * abs(thrust);
  der(altitude) = velocity;
  der(velocity) = acceleration;
end Rocket;
```


Celestial Body Class

A class declaration creates a *type name* in Modelica

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

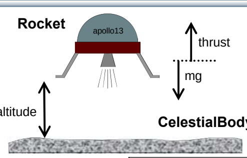
An *instance* of the class can be declared by *prefixing* the type name to a variable name

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

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

Moon Landing

Rocket



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

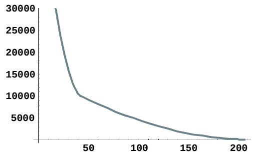
only access inside the class

access by dot notation outside the class

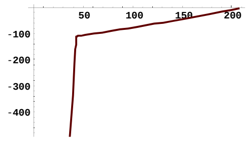
```
class MoonLanding
  parameter Real force1 = 36350;
  parameter Real force2 = 1308;
protected
  parameter Real thrustEndTime = 210;
  parameter Real thrustDecreaseTime = 43.2;
public
  Rocket        apollo(name="apollo13");
  CelestialBody   moon(name="moon",mass=7.382e22,radius=1.738e6);
equation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


Simulation of Moon Landing

```
simulate(MoonLanding, stopTime=230)
plot(apollo.altitude, xrange={0,208})
plot(apollo.velocity, xrange={0,208})
```



It starts at an altitude of 59404 (not shown in the diagram) at time zero, gradually reducing it until touchdown at the lunar surface when the altitude is zero



The rocket initially has a high negative velocity when approaching the lunar surface. This is reduced to zero at touchdown, giving a smooth landing

Specialized Class Keywords

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

- Example: (Modelica 2.2). A `model` is a class that cannot be used as a connector class
- Example: A `record` is a class that only contains data, with no equations
- Example: A `block` is a class with fixed input-output causality

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


Modelica Functions

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

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


Function Call – Example Function with for-loop

Example Modelica function call:

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

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

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

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

Inheritance

parent class to Color

restricted kind of class without equations

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

child class or subclass

keyword denoting inheritance

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

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

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

Extra slide

Multiple Inheritance

Multiple Inheritance is fine – inheriting both geometry and color

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

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

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

multiple inheritance

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

Equivalent to

Multiple Inheritance cont'

Extra slide

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

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

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

Diamond Inheritance

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

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


Simple Class Definition

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

```
class SameColor = Color;
```

Equivalent to:

inheritance

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

- Often used for introducing new names of types:

```
type Resistor = Real;
```

```
connector MyPin = Pin;
```

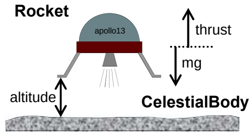

Inheritance Through Modification

- Modification is a concise way of combining inheritance with declaration of classes or instances
- A *modifier* modifies a declaration equation in the inherited class
- Example: The class `Real` is inherited, modified with a different `start` value equation, and instantiated as an `altitude` variable:

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


The Moon Landing - Example Using Inheritance (I)

Extra slide




```
model Body "generic body"
  Real   mass;
  String name;
end Body;
```

```
model CelestialBody
  extends Body;
  constant  Real g = 6.672e-11;
  parameter Real radius;
end CelestialBody;
```

```
model Rocket "generic rocket class"
  extends Body;
  parameter Real massLossRate=0.000277;
  Real altitude(start= 59404);
  Real velocity(start= -2003);
  Real acceleration;
  Real thrust;
  Real gravity;
equation
  thrust-mass*gravity= mass*acceleration;
  der(mass)= -massLossRate*abs(thrust);
  der(altitude)= velocity;
  der(velocity)= acceleration;
end Rocket;
```


The Moon Landing - Example using Inheritance (II)

Extra slide

inherited parameters

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


Inheritance of Protected Elements

Extra slide

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

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

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

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

Equivalent to

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

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

A protected element cannot be accessed via dot notation!

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

Exercises 2.1 and 2.2 (See also next two pages)

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

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

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

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

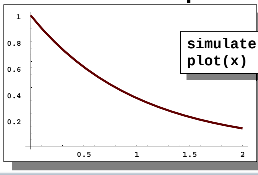
Exercise 2.1 – Hello World!

A Modelica "Hello World" model

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

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

Simulation in OpenModelica environment

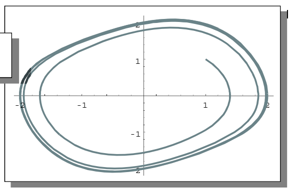



```
simulate(HelloWorld, stopTime = 2)
plot(x)
```


(extra) Exercise 2.2 – Van der Pol Oscillator

```
class VanDerPol  "Van der Pol oscillator model"
  Real x(start = 1)  "Descriptive string for x"; // x starts at 1
  Real y(start = 1)  "y coordinate";             // y starts at 1
  parameter Real lambda = 0.3;
equation
  der(x) = y;                           // This is the 1st diff equation //
  der(y) = -x + lambda*(1 - x*x)*y;  /* This is the 2nd diff equation */
end VanDerPol;
```

```
simulate(VanDerPol,stopTime = 25)
plotParametric(x,y)
```



(extra) Exercise 2.3 – DAE Example

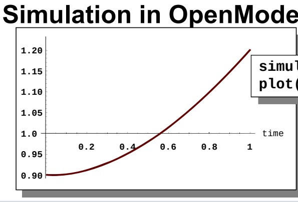
Include algebraic equation

Algebraic equations contain no derivatives

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

```
class DAEexample
  Real x(start=0.9);
  Real y;
equation
  der(y)+(1+0.5*sin(y))*der(x)
    = sin(time);
  x - y = exp(-0.9*x)*cos(y);
end DAEexample;
```

Simulation in OpenModelica environment




```
simulate(DAEexample, stopTime = 1)
plot(x)
```


Exercise 2.4 – Model the system below

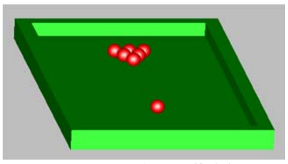
- Model this Simple System of Equations in Modelica

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

(extra) Exercise 2.5 – Functions

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

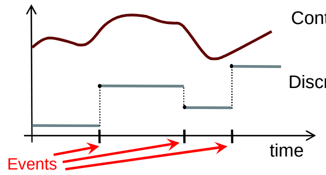
Part III b
Discrete Events and Hybrid Systems



Picture: Courtesy Hilding Elmqvist

Hybrid Modeling

Hybrid modeling = continuous-time + discrete-time modeling




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

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

- A *point* in time that is instantaneous, i.e., has zero duration
- An *event condition* so that the event can take place
- A set of *variables* that are associated with the event
- Some *behavior* associated with the event, e.g. *conditional equations* that become active or are deactivated at the event

Event Creation – if

if-equations, if-statements, and if-expressions

```
if <condition> then
  <equations>
elseif <condition> then
  <equations>
else
  <equations>
end if;
```

```
model Diode "Ideal diode"
  extends TwoPin;
  Real s;
  Boolean off;
equation
  off = s < 0;
  if off then
    v=s
  else
    v=0;
  end if;
  i = if off then 0 else s;
end Diode;
```

false if s<0

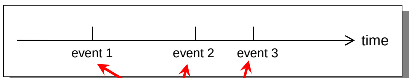
If-equation choosing equation for v

If-expression

Event Creation – when

when-equations

```
when <conditions> then
  <equations>
end when;
```



Equations only active at event times

Time event

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

Only dependent on time, can be scheduled in advance

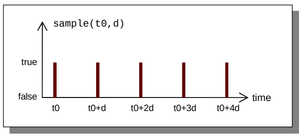
State event

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

Related to a state. Check for zero-crossing

Generating Repeated Events

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

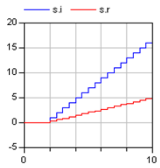


Variables need to be discrete

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

Creates an event after 2 s, then each 0.5 s

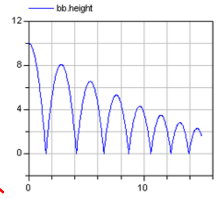
pre(...) takes the previous value before the event.



Reinit - Discontinuous Changes

The value of a *continuous-time* state variable can be instantaneously changed by a `reinit`-equation within a `when`-equation

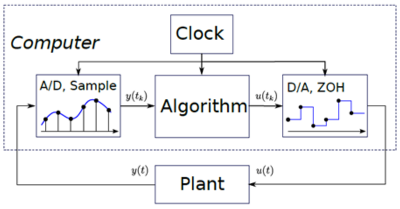
```
model BouncingBall "the bouncing ball model"
  parameter Real g=9.81;   //gravitational acc.
  parameter Real c=0.90;   //elasticity constant
  Real height(start=10),velocity(start=0);
equation
  der(height) = velocity;
  der(velocity)=-g;
  when height<0 then
    reinit(velocity, -c*velocity);
  end when;
end BouncingBall;
```



Initial conditions

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

Application: Digital Control Systems



- Discrete-time controller + continuous-time plant = hybrid system or sampled-data system
- Typically periodic sampling, can be modeled with "**`when sample`**`(t0,td)` **`then`** …"

Sampled Data-Systems in Modelica

```
// time-discrete controller
when {initial(),sample(3,3)} then
    E*xd = A*pre(xd)+ B*y;
    ud = C*pre(xd) + D*y;
end when;
```

y

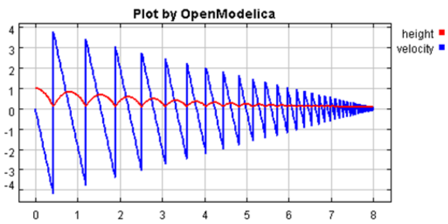
ud

```
// plant (continuous-time process)
0 = f(der(x), x, ud);
y = g(x);
```

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

Exercise 2.6 – BouncingBall

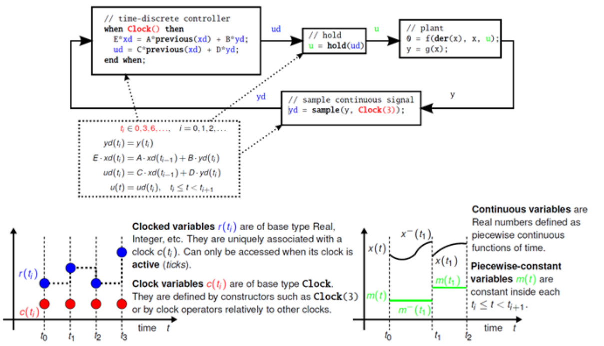
- Locate the BouncingBall model in one of the hybrid modeling sections of DrModelica (the When-Equations link in Section 2.9), run it, change it slightly, and re-run it.



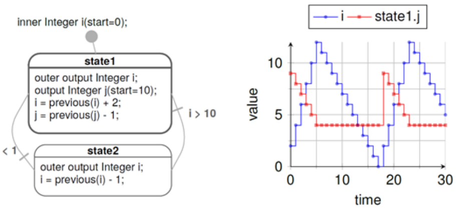
Part IIIc "Technology Preview"

Clocked Synchronous Models and State Machines

Clocked Synchronous Extension in Modelica 3.3



State Machines in Modelica 3.3: Simple Example

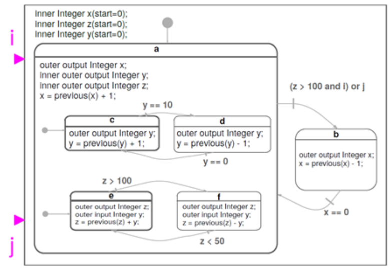


- Equations are active if corresponding *clock* ticks. Defaults to periodic clock with 1.0 s sampling period
- “i” is a shared variable, “j” is a local variable. Transitions are “*delayed*” and enter states by “*reset*”

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



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.

Technology Preview

- The clocked synchronous language extension not yet ready in OpenModelica (under development)
  - However some simple models can be simulated.
- No graphical editing support for state machine in OMEdit, yet.
- Full state machine extension requires that clocked synchronous support is available
- However, many state machines can already be simulated
  - By using a workaround that restricts the sampling period of a state machine to a fixed default value of 1s.

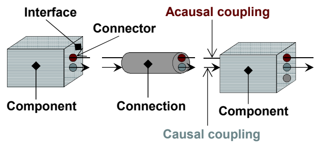
Preview Clocked Synchronous and State Machines

- The OMNotebook ebook “SynchronousAndStateMachinePreview.onb” provides one example featuring clocked synchronous language elements and two state machine examples.
- **Open** this and **simulate**. (If there is time)

Part IV

Components, Connectors and Connections – Modelica Libraries and Graphical Modeling

Software Component Model



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

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

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

Connectors and Connector Classes

Connectors are instances of *connector classes*

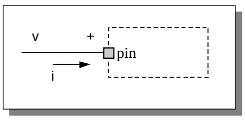
electrical connector

connector class

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

an instance `pin` of class `Pin`

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

Pin pin;
```

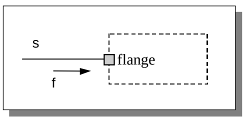


mechanical connector

connector class

an instance `flange` of class `Flange`

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

Flange flange;
```



The `flow` prefix

Two kinds of variables in connectors:

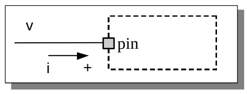
- *Non-flow variables potential* or energy level
- *Flow variables* represent some kind of flow

Coupling

- *Equality coupling*, for non-`flow` variables
- *Sum-to-zero coupling*, for `flow` variables

The value of a `flow` variable is *positive* when the current or the flow is *into* the component

positive flow direction:



Physical Connector

- Classes Based on Energy Flow

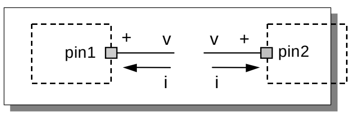
Domain Type	Potential	Flow	Carrier	Modelica Library
Electrical	Voltage	Current	Charge	`Electrical.Analog`
Translational	Position	Force	Linear momentum	`Mechanical.Translational`
Rotational	Angle	Torque	Angular momentum	`Mechanical.Rotational`
Magnetic	Magnetic potential	Magnetic flux rate	Magnetic flux	
Hydraulic	Pressure	Volume flow	Volume	`HyLibLight`
Heat	Temperature	Heat flow	Heat	`HeatFlow1D`
Chemical	Chemical potential	Particle flow	Particles	Under construction
Pneumatic	Pressure	Mass flow	Air	`PneuLibLight`

`connect`-equations

Connections between connectors are realized as *equations* in Modelica

```
connect(connector1,connector2)
```

The two arguments of a `connect`-equation must be references to *connectors*, either to be declared directly *within* the *same class* or be *members* of one of the declared variables in that class




```
Pin pin1,pin2;
//A connect equation
//in Modelica:
connect(pin1,pin2);
```

Corresponds to

```
pin1.v = pin2.v;
pin1.i + pin2.i =0;
```

Connection Equations

```
Pin pin1,pin2;
//A connect equation
//in Modelica
connect(pin1,pin2);
```

Corresponds to

```
pin1.v = pin2.v;
pin1.i + pin2.i =0;
```

Multiple connections are possible:

```
connect(pin1,pin2); connect(pin1,pin3); ... connect(pin1,pinN);
```

Each primitive connection set of nonflow variables is used to generate equations of the form:

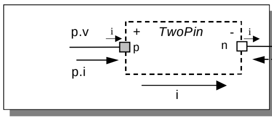
$$v_1 = v_2 = v_3 = \ldots v_n$$

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

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

Common Component Structure

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



partial class (cannot be instantiated)

positive pin

negative pin

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

electrical connector class

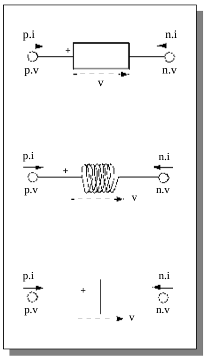
```
connector Pin
  Voltage      v;
  flow Current i;
end Pin;
```


Electrical Components

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

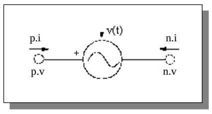
```
model Inductor "Ideal electrical inductor"
  extends TwoPin;
  parameter Real L "Inductance";
equation
  L*der(i) = v;
end Inductor;
```

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



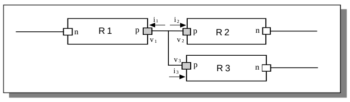
Electrical Components cont'

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




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

Resistor Circuit




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

Corresponds to

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


Modelica Standard Library - Graphical Modeling

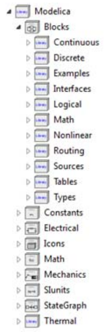
- *Modelica Standard Library* (called Modelica) is a standardized predefined package developed by Modelica Association
- It can be used freely for both commercial and noncommercial purposes under the conditions of *The Modelica License*.
- Modelica libraries are available online including documentation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

Modelica Standard Library cont'

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

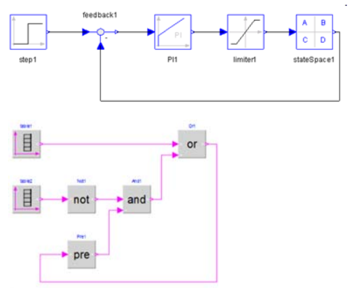
- Blocks — Library for basic input/output control blocks
- Constants — Mathematical constants and constants of nature
- Electrical — Library for electrical models
- Icons — Icon definitions
- Fluid — 1-dim Flow in networks of vessels, pipes, fluid machines, valves, etc.
- Math — Mathematical functions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

Modelica.Blocks



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

Examples:

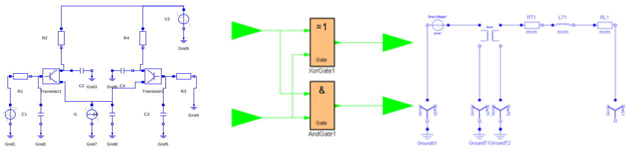


Modelica.Electrical

Electrical components for building analog, digital, and multiphase circuits

Analog Digital Machines MultiPhase

Examples:

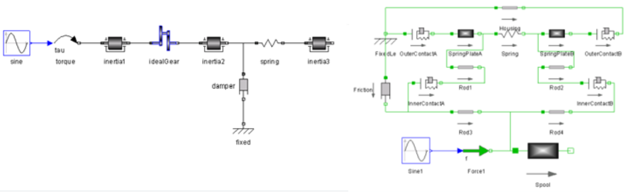


Modelica.Mechanics

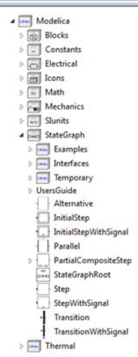
Package containing components for mechanical systems

Subpackages:

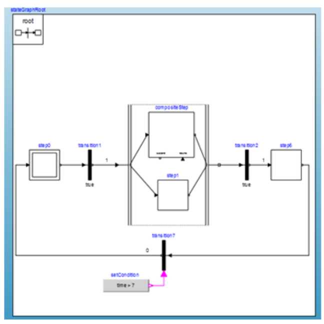
- Rotational — 1-dimensional rotational mechanical components
- Translational — 1-dimensional translational mechanical components
- MultiBody — 3-dimensional mechanical components



Modelica.Stategraph



Hierarchical state machines (similar to Statecharts)



Other Free Libraries

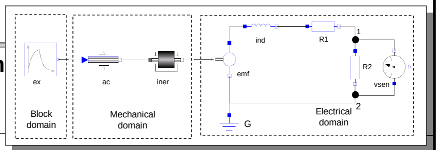
- WasteWater — Wastewater treatment plants, 2003
- ATPlus — Building simulation and control (fuzzy control included), 2005
- MotorCycleDymanics — Dynamics and control of motorcycles, 2009
- NeuralNetwork — Neural network mathematical models, 2006
- VehicleDynamics — Dynamics of vehicle chassis (obsolete),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

Some Commercial Libraries

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

Connecting Components from Multiple Domains

- **Block domain**
- **Mechanical domain**
- **Electrical domain**

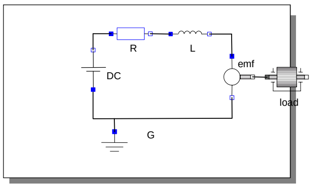



```
model Generator
  Modelica.Mechanics.Rotational.Accelerate ac;
  Modelica.Mechanics.Rotational.Inertia iner;
  Modelica.Electrical.Analog.Basic.EMF emf(k=-1);
  Modelica.Electrical.Analog.Basic.Inductor ind(L=0.1);
  Modelica.Electrical.Analog.Basic.Resistor R1,R2;
  Modelica.Electrical.Analog.Basic.Ground G;
  Modelica.Electrical.Analog.Sensors.VoltageSensor vsens;
  Modelica.Blocks.Sources.Exponentials ex(riseTime={2},riseTimeConst={1});
equation
  connect(ac.flange_b, iner.flange_a); connect(iner.flange_b, emf.flange_b);
  connect(emf.p, ind.p); connect(ind.n, R1.p); connect(emf.n, G.p);
  connect(emf.n, R2.n); connect(R1.n, R2.p); connect(R2.p, vsens.n);
  connect(R2.n, vsens.p); connect(ex.outPort, ac.inPort);
end Generator;
```


DCMotor Model Multi-Domain (Electro-Mechanical)

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

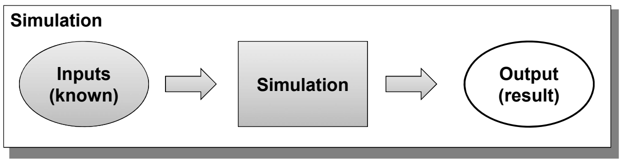
```
model DCMotor
   Resistor R(R=100);
   Inductor L(L=100);
   VsourceDC DC(f=10);
   Ground G;
   EMF emf(k=10,J=10, b=2);
   Inertia load;
equation
   connect(DC.p,R.n);
   connect(R.p,L.n);
   connect(L.p, emf.n);
   connect(emf.p, DC.n);
   connect(DC.n,G.p);
   connect(emf.flange,load.flange);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

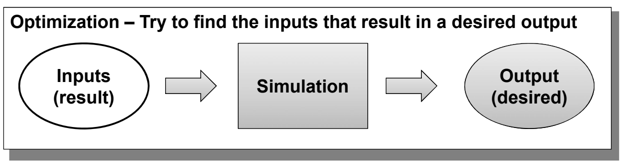


Part V
Dynamic Optimization
Theory and Exercises

using
OpenModelica

Built-in Dynamic Optimization - Motivation



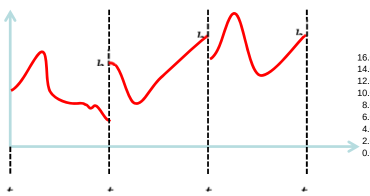


Optimization of Dynamic Trajectories Using Multiple-Shooting and Collocation

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

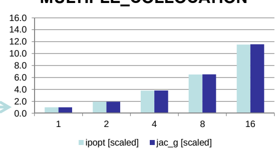
This approach uses a single optimization run and is different from classical parameter sweep optimization typically using a large number of simulations

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



Example speedup, 16 cores:

MULTIPLE_COLLOCATION



Optimal Control Problem (OCP)

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

Subject to

Initial conditions	$x(t_0) = x_0$	(2)
Nonlinear dynamic model	$\dot{x} = f(x(t),u(t),t)$	(3)
Path constraints	$\hat{g}(x(t),u(t),t) \leq 0$	(4)
Terminal constraints	$r\big(x(t_f)\big) = 0$	(5)

where

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

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

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

OCP Formulation in OpenModelica

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

$$x_{\min} \leq x(t) \leq x_{\max}$$
$$u_{\min} \leq u(t) \leq u_{\max}$$

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

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

Predator-Prey Example – The Forest Model

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

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

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

where

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

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

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

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

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

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

Predator-Prey Example – Modelica model

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

Control variables

Predator-Prey Example – Optimal Control Problem

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

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

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

Modelica model:

Extension of the system model

constraint

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

Cost function Mayer-term

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


Predator-Prey Example – Using OMNotebook

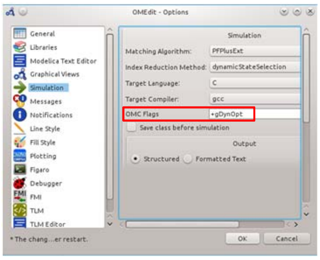
Start the optimization from OMNotebook using a time interval $[t_0, t_f] = [0{,}400]$seconds

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

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

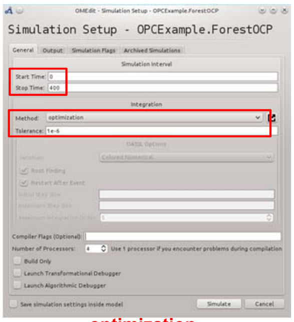
Predator-Prey Example – Using OMEdit

Tools→Options→Simulation



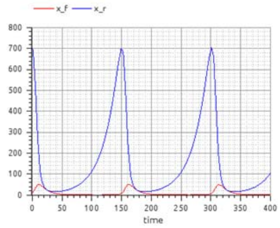
+gDynOpt

Simulation→Simulation Setup

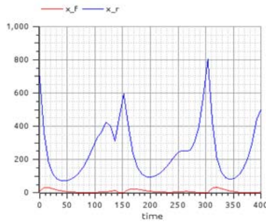


optimization

Predator-Prey Example – Plots



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



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

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

Exercise – Optimal Control

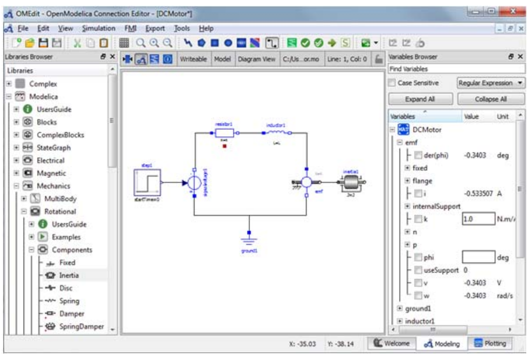
Load the `OPCExample.onb` ebook into OMNotebook and modify the optimization problem in the following ways:

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

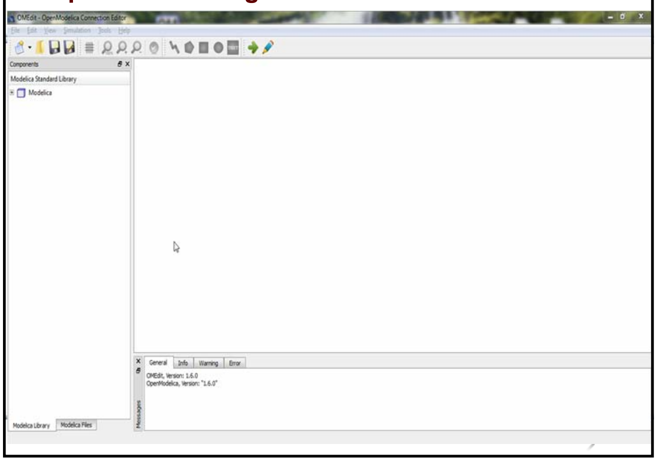
Part Vb
More
Graphical Modeling Exercises

using
OpenModelica

Graphical Modeling - Using Drag and Drop Composition



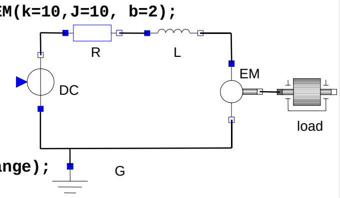
Graphical Modeling Animation – DCMotor



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                 (load component not included)
DC.n.v == G.p.v
```

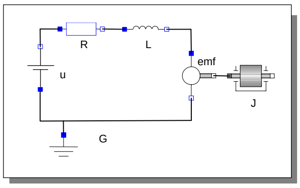
Automatic transformation to ODE or DAE for simulation:

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

Exercise 3.1

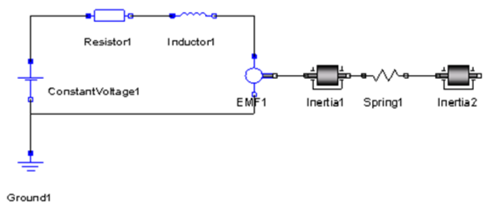
- Draw the `DCMotor` model using the graphic connection editor using models from the following Modelica libraries:
  `Mechanics.Rotational.Components`,
  `Electrical.Analog.Basic`,
  `Electrical.Analog.Sources`

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



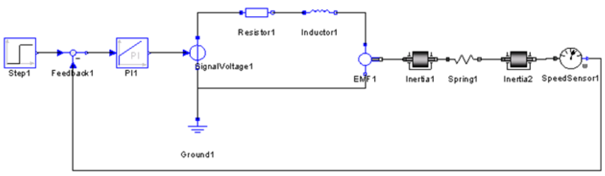
Exercise 3.2

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



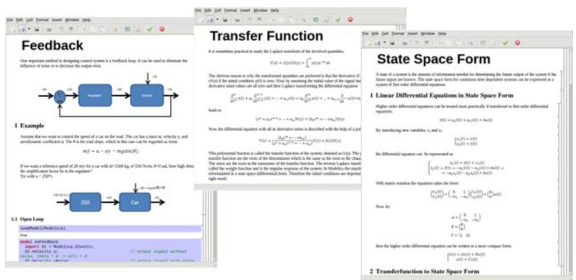
Exercise 3.3

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

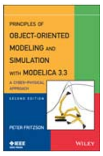


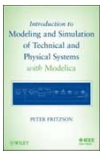
Exercise 3.4 – DrControl

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



Learn more…





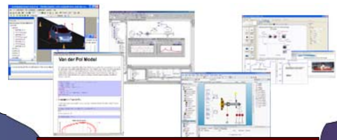
- OpenModelica
  - www.openmodelica.org
- Modelica Association
  - www.modelica.org
- Books
  - Principles of Object Oriented Modeling and Simulation with Modelica 3.3: A Cyber-Physical Approach, Peter Fritzson 2015.
  - Modeling and Simulation of Technical and Physical Systems with Modelica. Peter Fritzson.,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

Summary

Multi-Domain Modeling

Visual Acausal Component Modeling

MODELICA



Typed Declarative Textual Language

Thanks for listening!

Hybrid Modeling