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

Tutorial 2020-02-04

Peter Fritzson
Professor em. at Linköping University, peter.fritzson@liu.se
Research Director at Programming Environments Lab
Vice Director of the Open Source Modelica Consortium
Vice Chairman of Modelica Association

Adrian Pop
Linköping University, adrian.pop@liu.se
Technical Coordinator of the Open Source Modelica Consortium

Slides
Based on book and lecture notes by Peter Fritzson
Contributions 2004-2005 by Emma Larsdotter Nilsson, Peter Bunus
Contributions 2006-2018 by Adrian Pop and Peter Fritzson
Contributions 2009 by David Broman, Peter Fritzson, Jan Brugård, and
Mohsen Torabzadeh-Tari
Contributions 2010 by Peter Fritzson
Contributions 2011 by Peter F., Mohsen T., Adeel Asghar,
Contributions 2012-2018 by Peter Fritzson, Lena Buffoni, Mahder
Gebremedhin, Bernhard Thiele, Lennart Ochel
Contributions 2019-2020 by Peter Fritzson, Arunkumar Palanisamy, Bernt
Lie, Adrian Pop
Tutorial Based on Book, December 2014
Download OpenModelica Software

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

Can be ordered from Wiley or Amazon


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

September 2011
232 pages

Translations available in Chinese, Japanese, Spanish

Wiley
IEEE Press

For Introductory Short Courses on Object Oriented Mathematical Modeling
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 3.3”, copyright Wiley-IEEE Press
- Some examples and figures reproduced with permission from Modelica Association, Martin Otter, Hilding Elmqvist, Wolfram MathCore, Siemens
- Modelica Association: www.modelica.org
- OpenModelica: www.openmodelica.org
Outline

Part I
Introduction to Modelica and a demo example

Part II
Modelica environments

Part III
Modelica language concepts and textual modeling

Part IV
Graphical modeling and the Modelica standard library
Software Installation - Windows

• Start the software installation

• Install OpenModelica-1.16.0 nightly build Download or from the USB Stick (takes about 20min)
Software Installation – Linux (requires internet connection)

• Go to https://openmodelica.org/index.php/download/download-linux and follow the instructions.
Software Installation – MAC (requires internet connection)

• Go to https://openmodelica.org/index.php/download/download-mac and follow the instructions 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`
Software Installation for OMSens – Python Anaconda

USB INSTALLATION

This is **only needed for Sensitivity analysis feature**

Install **Anaconda3.exe** from USB stick (**Windows 64 bit**) (takes about 20 min)

For other platforms, download from Anaconda : https://www.anaconda.com/distribution/
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
Modelica Background: The Form – Equations

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

\[ 14.2e - 1.15.9 = 71.9 \]

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”:
\[
\text{variable} = \text{expression} \\
v = \text{INTEG}(F)/m
\]

Programming languages usually do not allow equations!
What is Modelica?

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

- Robotics
- Automotive
- Aircrafts
- Satellites
- Power plants
- Systems biology
What is Modelica?

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

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

A language for modeling of complex cyber-physical systems

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

Free, open language specification:

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

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

Available at: www.modelica.org

*Developed and standardized by Modelica Association*
Declarative statically typed language
   Equations and mathematical functions allow acausal modeling,
   high level specification and static type checking for increased correctness

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

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

Visual component programming
   Hierarchical system architecture capabilities

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

\[ R \cdot i = v; \]

can be used in three ways:

- \( i := v/R; \)
- \( v := R \cdot i; \)
- \( R := v/i; \)
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

Physical
- Mechanics

Electric

Reference

Cyber

PID

Control System
What is Special about Modelica?

Multi-Domain Modeling

Acausal model (Modelica)

Causal block-based model (Simulink)

Visual Acausal Hierarchical Component Modeling

Keeps the physical structure
What is Special about Modelica?

Multi-Domain Modeling
Hierarchical system modeling
Visual Acausal Hierarchical Component Modeling

Srel = n*transpose(n)+(identity(3)- n*transpose(n))*cos(q)- skew(n)*sin(q);
we = n*qd;
zela = n*qdd;
Sb = Sa*transpose(Srel);
r0b = r0a;
v = Srel*va;
w = Srel*(wa + wela);
a = Srel*aa;
z = Srel*(za + zela + cross(wa, wela));
What is Special about Modelica?

Multi-Domain 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)

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

Typed

Declarative

Equation-based

Textual Language

Visual Acausal

Hierarchical

Component Modeling
What is Special about Modelica?

- Hybrid modeling = continuous-time + discrete-time modeling
- Multi-Domain Modeling
- Visual Acausal Component Modeling
- Typed Declarative Equation-based Textual Language
- Continuous-time
- Discrete-time
- Clocked discrete-time

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

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

**Keeps the physical structure**

---

Modelica:

| R1=10 | C=0.01 | L=0.1 | R2=100 |

Simulink:

![Diagram](image)
Graphical Modeling - Using Drag and Drop Composition
Graphical Modeling with OpenModelica Environment
Multi-Domain (Electro-Mechanical) Modelica Model

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

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

Multi-Domain (Electro-Mechanical) Modelica Model
Corresponding DCMotor Model Equations

The following equations are automatically derived from the Modelica model:

\[
\begin{align*}
0 &= \text{DC.p.i} + \text{R.n.i} \\
\text{DC.p.v} &= \text{R.n.v} \\
0 &= \text{EM.p.i} + \text{EM.n.i} \\
\text{EM.i} &= \text{EM.p.i} \\
0 &= \text{R.p.i} + \text{L.n.i} \\
\text{R.p.v} &= \text{L.n.v} \\
\text{EM.i} &= \text{EM.M} / \text{EM.k} \\
\text{EM.J} \times \text{EM.}\omega &= \text{EM.M} - \text{EM.b} \times \text{EM.}\omega \\
0 &= \text{L.p.i} + \text{EM.n.i} \\
\text{L.p.v} &= \text{EM.n.v} \\
\text{DC.u} &= \text{DC.p.v} - \text{DC.n.v} \\
0 &= \text{DC.p.i} + \text{DC.n.i} \\
\text{EM.p.v} &= \text{DC.n.v} \\
\text{DC.i} &= \text{DC.p.i} \\
\text{DC.u} &= \text{DC.Amp} \times \sin[2 \pi \text{DC.f} \times t] \\
0 &= \text{DC.n.i} + \text{G.p.i} \\
\text{DC.n.v} &= \text{G.p.v}
\end{align*}
\]

Automatic transformation to ODE or DAE for simulation:

\[
\frac{dx}{dt} = f[x, u, t] \\
g\left[\frac{dx}{dt}, x, u, t\right] = 0
\]
Model Translation Process to Hybrid DAE to Code

Modelica Graphical Editor

Modelica Model

Modelica Textual Editor

Modelica Model

Modelica Source code

Translator

Flat model Hybrid DAE

Analyzer

Sorted equations

Optimizer

Optimized sorted equations

Code generator

C Code

Code generator

C Compiler

Executable

Simulation

C Compiler

Simulation

"Middle-end"

Backend

"Middle-end"

Frontend

Modeling Environment

Modeling Environment
Modelica in Automotive Industry
Modelica in Biomechanics
Application of Modelica in Robotics Models
Real-time Training Simulator for Flight, Driving

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

(Movie demo next page)

Courtesy of Tobias Bellmann, DLR, Oberphaffenhofen, Germany
DLR Real-time Training Simulator Movie Demo
Combined-Cycle Power Plant
Plant model – system level

• GT unit, ST unit, Drum boilers unit and HRSG units, connected by thermo-fluid ports and by signal buses.

• Low-temperature parts (condenser, feedwater system, LP circuits) are represented by trivial boundary conditions.

• GT model: simple law relating the electrical load request with the exhaust gas temperature and flow rate.

Courtesy Francesco Casella, Politecnico di Milano – Italy and Francesco Pretolani, CESI SpA - Italy
Formation flying on elliptical orbits

Control the relative motion of two or more spacecraft

Attitude control for satellites using magnetic coils as actuators

Torque generation mechanism: interaction between coils and geomagnetic field

Courtesy of Francesco Casella, Politecnico di Milano, Italy
Large-scale ABB OpenModelica Application
Generate code for controlling 7.5 to 10% of German Power Production

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

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

Manage 7.5% - 10% of German Power
- 2015, Aug: OpenModelica Exports FMUs for real-time optimizing control (seconds) of about **5,000 MW (7.5%) of power in Germany**
Industrial Product with OEM Usage of OpenModelica – MIKE by DHI, WEST Water Quality, Water Treatment and Sludge

- **MIKE by DHI**, www.mikebydhi.com, **WEST Water Quality** modeling and simulation environment
- Includes a large part of the OpenModelica compiler using the OEM license.
- Here a water treatment effluent and sludge simulation.
Most important challenge for humanity - Develop a sustainable society!

Use Modelica in to model and optimize sustainable technical innovations, and a sustainable circular economy
System Dynamics – World Society Simulation
Limits to Material Growth; Population, Energy and Material flows

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

CO2 Emissions per person:
- USA 17 ton/yr
- Sweden 7 ton/yr
- India 1.4 ton/yr
- Bangladesh 0.3 ton/yr

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

Not yet evident!

Humans on a finite Earth vs Bacteria on a finite substrate

Bacterial growth curve /kinetic curve (Wikipedia)
World3 Simulations with Different Start Years for Sustainable Policies – Collapse if starting too late

- **World Population (billion people)**
  - Time (Years): 1900 to 2100
  - World3 Simulations with Scenario 9 (sustainable)
  - Collapse starts year 2050 if sustainable policies are started too late

- Lines for Start Years:
  - Start 2012
  - Start 2022
  - Start 2032
  - Start 2042
How the world could be in 80-100 years at a global warming of 4 degrees

Business-as-usual scenario, IPCC

MASSIVE MIGRATION TO
TO NORTHERN EUROPE,
RUSSIA, AND CANADA

Example Emissions
CO₂e / person
- Earth can handle 2 ton/yr
- Flight Spain – 1 ton
- Flight Canaryisl – 2 ton
- Flight Thailand – 4 ton

References
New Scientist, 28 February 2009
IPCC, business as usual scenario
www.climate-lab-book.ac.uk
www.atmosfair.de
What Can You Do?
Need Global Sustainability Mass Movement

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

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

Expanded to 93 sqm, 12 kW, March 2013
House produced 11600 kwh, used 9500 kwh
Avoids 10 ton CO2 emission per year
Example Electric Cars
Can be charged by electricity from own solar panels

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

DLR ROboMObil
- experimental electric car
- Modelica models

2018, Tesla Model 3 LR, range 560 km
Tesla Model S, range about 550 km
What Can You Do?
More Train Travel – Less Air Travel

• Air travel by Swedish Citizens – about the same emissions as all personal car traffic in Sweden!

• By train from Linköping to Munich and back – saves almost 1 ton of CO2e emissions compared to flight

• Leave Linköping 07.00 in Munich 23.14

More Examples, PF travel 2016:
• Train Linköping-Paris, Dec 3-6, EU project meeting
• Train Linköping-Dresden, Dec 10-16, 1 week workshop
Small rectangles – surface needed for 100% solar energy for humanity
Solar Energy PhotoVoltaics Growth Trends

Almost Exponential worldwide Growth of Photovoltaics 2006 – 2018

IEA PVPS TRENDS IN PHOTOVOLTAIC APPLICATIONS 2019

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

2018 2.5% solar
Sustainable Society Necessary for Human Survival

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

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

Gandhi – role model for future less materialistic life style
Brief Modelica History

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

• Modelica Versions
  • 1.0 released September 1997
  • 2.0 released March 2002
  • 2.2 released March 2005
  • 3.0 released September 2007
  • 3.1 released May 2009
  • 3.2 released March 2010
  • 3.3 released May 2012
  • 3.2 rev 2 released November 2013
  • 3.3 rev 1 released July 2014
  • 3.4 released April 2017

• Modelica Association established 2000 in Linköping
  • Open, non-profit organization
Modelica Conferences

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

\[
\begin{align*}
R &= 100 \\
L &= 1
\end{align*}
\]
Exercises Part I – OMEdit Instructions (Part I)

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

Under the Modelica Library:
• Contains The standard Modelica library components
• The Modelica files contains the list of models you have created.
Exercises Part I – OMEdit Instructions (Part II)

• For the RLCircuit model, **browse** the Modelica standard library and **add** the following component models:
  • Add SineVoltage component model from Modelica.Electrical.Analog.Sources package.
• Make the corresponding **connections** between the component models as shown in the previous slide.
• To **draw a connection line**: first single-click on a connector box; then start drawing while keeping the mouse button down; after drawing a little you can release the mouse button and continue drawing.
• **Simulate** the model
  • Go to the Simulation menu and choose simulate or click on the simulate button in the toolbar.
• **Plot** the instance variables
  • Once the simulation is completed, a plot variables list will appear on the right side. Select the variable that you want to plot.
Part II

Modelica environments and OpenModelica
Dymola

- Dassault Systemes Sweden
- Sweden
- First Modelica tool on the market
- Initial main focus on automotive industry
- www.dymola.com
Wolfram System Modeler – Wolfram MathCore

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

Car model graphical view

Mathematica

Simulation and analysis
Simulation X

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

- Maplesoft
- Canada
- Integrated with Maple
- www.maplesoft.com
Modelon

- Modelon
- Sweden and International
- Library Suite
- Creator Suite with Optimica Compiler Toolbox and WAMS model editor
- www.modelon.com
The OpenModelica Environment

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

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

EngineV6 11116 equation model
The OpenModelica Open Source Environment
www.openmodelica.org

• Advanced Interactive Modelica compiler (OMC)
  • Supports most of the Modelica Language
  • Modelica, Python, Julia scripting
• OMSimulator – FMI Simulation/Co-simulation
• Basic environment for creating models
  • OMShell – an interactive command handler
  • OMNotebook – a literate programming notebook
  • MDT – an advanced textual environment in Eclipse

• OMEdit graphic Editor
• OMDebugger for equations
• OMOptim optimization tool
• OM Dynamic optimizer collocation
• ModelicaML UML Profile
• MetaModelica extension
• ParModelica extension
OSMC – International Consortium for Open Source Model-based Development Tools, 51 members Dec 2019

Founded Dec 4, 2007

Open-source community services

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

Industrial members

- ABB AB, Sweden
- Bosch Rexroth AG, Germany
- CDAC Centre, Kerala, India
- Creative Connections, Prague
- DHI, Aarhus, Denmark
- Dynamica s.r.l., Cremona, Italy
- EDF, Paris, France
- Equa Simulation AB, Sweden
- Fraunhofer IWES, Bremerhaven
- INRIA, Rennes, France
- ISID Dentsu, Tokyo, Japan
- Maplesoft, Canada
- RTE France, Paris, France
- Saab AB, Linköping, Sweden
- SKF, Göteborg, Sweden
- SmartFluidPower, Italy,
- TLK Thermo, Germany
- Siemens Turbo, Sweden
- Sozhou Tongyuan, China
- SRON Space Ins Netherlands
- Talent Swarm, Spain
- VTI, Linköping, Sweden
- VTT, Finland

University members

- Augsburg University, Germany
- FH Bielefeld, Bielefeld, Germany
- University of Bolivar, Colombia
- TU Braunschweig, Germany
- Chalmers Univ, Control, Sweden
- Chalmers Univ, Machine, Sweden
- TU Darmstadt, Germany
- TU Delft, The Netherlands
- TU Dresden, Germany
- Université Laval, Canada
- Georgia Inst of Technology, USA
- Ghent University, Belgium
- Halmstad University, Sweden
- Heidelberg University, Germany
- TU Hamburg/Harburg Germany
- IIT Bombay, Mumbai, India
- K.L. Univ, Waddeswaram, India
- KTH, Stockholm, Sweden
- Linköping University, Sweden
- Univ of Maryland, Syst Eng USA
- Univ of Maryland, CEE, USA
- Politecnico di Milano, Italy
- Ecoles des Mines, CEP, France
- Mälardalen University, Sweden
- RPI, Troy, USA
- Univ Pisa, Italy
- Univ College SouthEast Norway
- Tsinghua Univ, Beijing, China
- Vanderbilt Univ, USA

Code Statistics

/trunk: Lines of Code

Lines

<table>
<thead>
<tr>
<th>Date</th>
<th>1,000,000</th>
<th>2,000,000</th>
<th>3,000,000</th>
<th>4,000,000</th>
<th>5,000,000</th>
<th>6,000,000</th>
<th>7,000,000</th>
<th>8,000,000</th>
</tr>
</thead>
<tbody>
<tr>
<td>1996</td>
<td>100,000</td>
<td>200,000</td>
<td>300,000</td>
<td>400,000</td>
<td>500,000</td>
<td>600,000</td>
<td>700,000</td>
<td>800,000</td>
</tr>
<tr>
<td>1997</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1998</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1999</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2001</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2002</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2003</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2004</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2005</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2007</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2008</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2009</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2010</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2011</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2012</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2013</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2014</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Build System with Regression Testing

- Automatic Nightly build system (using Jenkins), and several multi-core computers
- Regression testing of libraries
- Verification testing comparing results to references
To learn about Modelica, read a book or a tutorial about Modelica®.

Interactive step-by-step beginners Modelica on-line spoken tutorials

Interactive OMWebbook with examples of Modelica textual modeling

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

About 12 results found.

1. Introduction to OMEdit
   Foss : OpenModelica - English
   Outline: Introduction to OpenModelica Introduction to OMEdit Perspectives in OMEdit Browsers in OMEdit View Icons in OMEdit Open a Class from Libraries Browser Checking for correctness.

2. Examples through OMEdit
   Foss : OpenModelica - English
   Outline: Expand Modelica library Expand Electrical library Expand Analog library Open Redcifier Class Compare the values of IDC & Losses time vs Losses plot Expand Mechanics library ..

3. Developing an equation-based model
   Foss : OpenModelica - English
   Outline: Introduction to OMEdit Declaration of variables and equations Simulation of a model in
OMNotebook Electronic Notebook with DrModelica

- Primarily for teaching
- Interactive electronic book
- Platform independent

Commands:
- Shift-return (evaluates a cell)
- File Menu (open, close, etc.)
- Text Cursor (vertical), Cell cursor (horizontal)
- Cell types: text cells & executable code cells
- Copy, paste, group cells
- Copy, paste, group text
- Command Completion (shift-tab)
OMnotebook Interactive Electronic Notebook
Here Used for Teaching Control Theory

1 Kalman Filter

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

Another difficulty is that the measured quantities of the input signal can be evaluated by the difference

\[
\dot{x} = \begin{bmatrix} 0 & 1 \\ -2 & -1 \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u(t)
\]

Here \( e \) denoting a disturbance in the input signal that can be evaluated by the difference

\[
K(y(t))
\]

By using this quantity as feedback we obtain the observer

\[
\hat{x} = A \hat{x}(t) + Bu(t) + \sum_{s} K(s) (y(t) - \hat{y}(t)).
\]

Now form the error as

\[
\delta(t) = x(t) - \hat{x}(t)
\]

The differential error is

\[
\delta'(t) + 2 \delta(t)
\]
OMNotebook supports Latex formatting for mathematics.

Contents in OMWebbook
Generated from OMNotebook

Latex instructions can be hidden by double clicking the Cell in tree view.
OpenModelica Environment Demo
OpenModelica MDT – Eclipse Plugin

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

- Code Outline for easy navigation within Modelica files
- Identifier Info on Hovering
OpenModelica Simulation in Web Browser Client

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

MultiBody RobotR3.FullRobot
OMPython – Python Scripting with OpenModelica

- Interpretation of Modelica commands and expressions
- Interactive Session handling
- Library / Tool
- Optimized Parser results
- Helper functions
- Deployable, Extensible and Distributable
OMJulia – Julia Scripting with OpenModelica

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

Control example with OMJulia in Jupyter notebooks

Use of Modelica + Julia in Process Systems Engineering Education

Complex models of "Seborg reactor"

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

*University of South-Eastern Norway, Norway
**University of Linköping, Sweden

Introducing packages

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

Out[9]:
OMMatlab – Matlab Scripting with OpenModelica

- Interpretation of Modelica commands and expressions from Matlab, transfer of data
- Interactive Session handling
- Library / Tool
- Separately downloadable, be run with OpenModelica
- Similar API functions as in OMJulia and OMPython
- Can be used for control design from Matlab
OMEdit 3D Visualization of Multi-Body Systems

- Built-in feature of OMEdit to animate MSL-Multi-Body shapes
- Visualization of simulation results
- Animation of geometric primitives and CAD-Files
OpenModelica 3D Animation Demo
(V6Engine and Excavator)
Visualization using Third-Party Libraries: DLR Visualization Library

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

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

Courtesy of Dr. Tobias Bellmann (DLR)
Exercise 1.2: Use 3D Visualization for Robot model

- Press Simulate with Animation
- Replay the animation
- Compare with the plot
Exercise 1.3: Visualization using the DLR Visualization Community Edition (1)

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

Please note: As of OpenModelica v1.14 support for the library is only partial and it is not yet as stable, fast and complete as for the Dymola tool (work in progress!)
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 (2)

- Problems
- Solved problems
- Result plot
- Export result data .csv

[Image of the OMOptim interface showing solved problems, result plot, and options to export result data]

[Graph showing optimization results with points plotted on a grid]
Multiple-Shooting and Collocation
Dynamic Trajectory Optimization

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

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

Example speedup, 16 cores:
OpenModelica Dynamic Optimization Collocation

DAE -> Cost function
  ODE -> Weight sum of the cost
      -> Residual equations
          -> Collocation technique
              -> Discrete NLP

OpenModelica
  Discrete goal function
  Discrete constraint function
  Gradient
  Jacobian
  Hessian
OMSens – Multi-Parameter Sensitivity Analysis

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

Heatmap visualization

For an exercise, see further in these slides

Tool architecture

1. Models and scripts handling
   - Python
2. Modelica
   - OpenModelica Models and Scripts
3. Optimization
   - CURVI (Fortran)
4. OS
   - Linux

Simultaneous Sensitivity Analysis

Individual Sensitivity Analysis

5. Sensitivity indices
   - Python
6. Parameter sweeping
   - Python
OMSysIdent – System Parameter Identification

- OMSysIdent is a module for parameter estimation of behavioral models (wrapped as FMUs) on top of the OMSimulator API.
- Identification of the parameter values is typically based on measurement data.
- It uses the Ceres solver (http://ceres-solver.org/) for the optimization task.
General Tool Interoperability & Model Exchange
Functional Mock-up Interface (FMI)

- 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)
  - > 120 tools supporting it (https://www.fmi-standard.org/tools)
Functional Mockup Units

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

Main Framework Aspects

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

Standalone open source simulation tool with rich interfaces
- C/Java
- Scripting languages Python, Lua

Co-simulation framework as a solid base for engineering tools
- Integration into OpenModelica/Papyrus
- Open for integration into third-party tools and specialized applications (e.g. flight simulators, optimization)

OMSimulator in OpenModelica 1.13.2
- Supports both FMI and TLM
- TLM connections are optional
- Co-simulation to multiple tools
- Composite model editor
- External API interface and scripting

OMEdit
OMC
Papyrus
Scripting ...

TLM component
OMSimulator
C-API interface
OMSimulator
Integrated TLM & FMI libOMSimulator
FMI component
FMI component
Composite FMI component
ADAMS wrapper
Beast wrapper
Simulink wrapper
FMI FMU
Modelica model
OMSimulator Composite Model Editor with 3D Viewer

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

Adding SSP bus connections

FMI Simulation results in OMEdit

FMI Simulation Tool Comparison

<table>
<thead>
<tr>
<th>Feature</th>
<th>OMSimulator</th>
<th>DACCOSIM</th>
<th>Simulink</th>
<th>PyFMI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commercial</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Open-source</td>
<td>OSMC-PL, GPL</td>
<td>AGPL2</td>
<td>No</td>
<td>LGPL</td>
</tr>
<tr>
<td>Lookup Table</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Alg. Loops</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Scripting</td>
<td>Python, Lua</td>
<td>proprietary</td>
<td>proprietary</td>
<td>Python</td>
</tr>
<tr>
<td>GUI</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>SSP</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>platform</td>
<td>Linux/Win/macOS</td>
<td>Linux/Win</td>
<td>Linux/Win/macOS</td>
<td>Linux/Win/macOS</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Feature</th>
<th>Dymola</th>
<th>PySimulator</th>
<th>FMI Go!</th>
<th>FMI Composer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commercial</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Open-source</td>
<td>No</td>
<td>BSD</td>
<td>MIT</td>
<td>No</td>
</tr>
<tr>
<td>Lookup Table</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Alg. Loops</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Scripting</td>
<td>proprietary</td>
<td>Python</td>
<td>Go</td>
<td>No</td>
</tr>
<tr>
<td>GUI</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>SSP</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>platform</td>
<td>Linux/Win</td>
<td>Linux/Win</td>
<td>Linux/Win/macOS</td>
<td>Linux/Win/macOS</td>
</tr>
</tbody>
</table>
OpenModelica Functional Mockup Interface (FMI)

**FMI Export**
- Modelica Code
  - OpenModelica Compiler
    - Code Generation
      - FMU

**FMI Import**
- FMU
  - OpenModelica Compiler
    - Code Generation
      - Modelica Code
    - FMU parsing, reading states & events
FMI in OpenModelica

- Model Exchange implemented (FMI 1.0 and FMI 2.0)
- FMI 2.0 Co-simulation implemented
- The FMI interface is accessible via the OpenModelica scripting environment, the OpenModelica connection editor and the OMSimulator tool in OpenModelica
OpenModelica Code Generators for Embedded Real-time Code

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

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

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

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

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

Supports only a limited Modelica subset

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

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

[Diagram showing various blocks and components related to communication and I/O devices]

[Link to GitHub repository: https://github.com/modelica/Modelica_DeviceDrivers/]
OpenModelica and Device Drivers Library
AVR Processor Support

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

Modelica Device Drivers Library - AVR processor sub-packages:

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

Single board heating system (IIT Bombay)

- Use for teaching basic control theory
- Usually controlled by serial port (set fan value, read temperature, etc)
- OpenModelica can generate code targeting the ATmega16 on the board (AVR-ISP programmer in the lower left).
  Program size is 4090 bytes including LCD driver and PID-controller (out of 16 kB flash memory available).

Movie Demo, see next page!
Example – Code Generation to SHBS
OpenModelica – ModelicaML UML Profile
SysML/UML to Modelica OMG Standardization

• ModelicaML is a UML Profile for SW/HW modeling
  • Applicable to “pure” UML or to other UML profiles, e.g. SysML

• Standardized Mapping UML/SysML to Modelica
  • Defines transformation/mapping for **executable** models
  • Being **standardized** by OMG

• ModelicaML
  • Defines graphical concrete syntax (graphical notation for diagram) for representing Modelica constructs integrated with UML
  • Includes graphical formalisms (e.g. State Machines, Activities, Requirements)
    • Which do not exist in Modelica language
    • Which are translated into executable Modelica code
  • Is defined towards generation of executable Modelica code
  • Current implementation based on the Papyrus UML tool + OpenModelica
Example: Simulation and Requirements Evaluation

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

tank-height is 0.6m

Req. 001 for the tank2 is violated

Req. 001 for the tank1 is not violated
### vVDR Method – virtual Verification of Designs vs Requirements

<table>
<thead>
<tr>
<th>Actor</th>
<th>Task</th>
<th>Created Artifact</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Formalize Requirements</td>
<td>Requirement Monitor Models (RMM)</td>
</tr>
<tr>
<td></td>
<td>Formalize Designs</td>
<td>Designs Alternative Models (DAM)</td>
</tr>
<tr>
<td></td>
<td>Formalize Scenarios</td>
<td>Scenario Models (SM)</td>
</tr>
<tr>
<td>AUTOMATED</td>
<td>Create Verification Models</td>
<td>Verification Models (VM)</td>
</tr>
<tr>
<td>AUTOMATED</td>
<td>Execute and Create Report</td>
<td>Reports</td>
</tr>
</tbody>
</table>

**Goal:** Enable on-demand verification of designs against requirements using automated model composition at any time during development.
Need for Debugging Tools
Map Low vs High Abstraction Level

• A **major part** of the total **cost** of software projects is due to testing and debugging

• US-Study 2002:
  Software errors cost the US economy **annually ~ 60 Billion $**

• **Problem:** Large Gap in Abstraction Level from **Equations** to **Executable Code**

• Example error message (hard to understand)
  Error solving nonlinear system 132
  
  time = 0.002  
  residual[0] = 0.288956  
  x[0] = 1.105149  
  residual[1] = 17.000400  
  x[1] = 1.248448  
  ...

...
OpenModelica MDT Algorithmic Code Debugger
The OpenModelica MDT Debugger (Eclipse-based) Using Japanese Characters
OpenModelica Equation Model Debugger

Showing equation transformations of a model:

1. substitution:
   \[ 0 = y + \text{der}(x \cdot \text{time} \cdot z); \quad z = 1.0; \]

2. simplify:
   \[ y + \text{der}(x \cdot \text{time}) \]

3. expand derivative (symbolic diff):
   \[ y + \text{der}(x \cdot \text{time}) \]

4. solve:
   \[ 0.0 = y + (x + \text{der}(x)) \cdot \text{time} \]

Mapping run-time error to source model position
Transformations Browser – EngineV6 Overview
(11 116 equations in model)
Equation Model Debugger on Siemens Model
(Siemens Evaporator test model, 1100 equations)

Pointing out the buggy equation
\[ y = \frac{u1}{u2}; \]
that gives division by zero.
Debugging Example – Detecting Source of Chattering (excessive event switching) causing bad performance

equation
\[ z = \text{if } x > 0 \text{ then } -1 \text{ else } 1; \]
\[ y = 2^*z; \]
Error Indication – Simulation Slows Down


```
OMEdit - Debugging.Chattering.ChatteringEvents1 Simulation Output

stdout | info | Chattering detected around time 0.500000005..0.500000995001 (100 state events in a row with a total time delta less than the step size 0.002). This can be a performance bottleneck. Use -lv LOG_EVENTS for more information. The zero-crossing was: x > 0.0. Debug more
```

Cancel Simulation
Performance Profiling for Faster Simulation
(Here: Profiling all equations in MSL 3.2.1 DoublePendulum)

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

Performance profiling DoublePendulum:
Performance Profiling of Siemens Drum Boiler Model with Evaporator

Conclusion from the evaluation:
“...the profiler makes the process of performance optimization radically shorter.”
ABB Industry Use of OpenModelica FMI 2.0 and Debugger

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

- ABB: “ABB uses several compatible Modelica tools, including OpenModelica, depending on specific application needs.”

- ABB: “OpenModelica provides outstanding debugging features that help to save a lot of time during model development.”
Exercise 1.2 – Equation-based Model Debugger

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

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

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

Uses 25% CPU
Exercise – FMU Export and Import (1)

- Open OMEdit and check FMI settings in Tools->Options
Exercise – FMU Export and Import (2)

- Find the FMIExercise.mo file in the tutorial folder and open it in OMEdit. Click on the + at the left to open and see components TestPIFMU, PI, etc.

- **Goal**: (1) Export this PI block as FMU, (2) import the exported FMU, (3) compare simulation results of imported PI FMU block vs. native use of the PI block

- Export the PI block by selecting the model and use right-click context menu indicated at the right

- The message browser shows where the FMU was generated on your system

```
Messages Browser
[2] 17:04:04 Scripting Notification
The FMU is generated at /tmp/OpenModelica_bernhard/OMEdit/FMIExercise_Components_PI.fmu.
```
Exercise – FMU Export and Import (3)

- Import FMU by selecting FMI->Import FMU from the menu
- Find and select the FMU in the directory where it was exported before as indicated at the right
- The FMU should now appear in the package browser
Exercise – FMU Export and Import (4)

• The imported FMU is wrapped inside a standard Modelica model and can be inserted by drag and drop into an existing model.

• The model TestPIFMU has been prepared so that the results of the imported FMU can be easily compared to the native block. Simulate it and compare results by plotting.
Part III

Modelica language concepts and textual modeling

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

class Color extends ColorData;

class ExpandedColor extends ColorData

  parameter Real red = 0.2;
  parameter Real blue = 0.6;
  Real green;

  equation
  red + blue + green = 1;
end ExpandedColor;
```
Acausal Modeling

The order of computations is not decided at modeling time

Acausal

Causal

Visual Component Level

Equation Level

A resistor equation: $R \times i = v$;

Causal possibilities:

\[ i := \frac{v}{R}; \]
\[ v := R \times i; \]
\[ R := \frac{v}{i}; \]
Typical Simulation Process

- **Modelica model**
  - Elaboration
- **Hybrid DAE**
  - Equation Transformation & Code generation
- **Executable**
  - Simulation
- **Simulation Result**
Simple model - Hello World!

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

Name of model

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

Initial condition

Continuous-time variable

Parameter, constant during simulation

Simulation in OpenModelica environment

Differential equation

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

Name of model
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`

```plaintext
constant Real PI=3.141592653589793;
constant String redcolor = "red";
constant Integer one = 1;
parameter Real mass = 22.5;
```
A Simple Rocket Model

\[
\begin{align*}
\text{acceleration} &= \frac{\text{thrust} - \text{mass} \cdot \text{gravity}}{\text{mass}} \\
\text{mass}' &= -\text{massLossRate} \cdot \text{abs}(\text{thrust}) \\
\text{altitude}' &= \text{velocity} \\
\text{velocity}' &= \text{acceleration}
\end{align*}
\]

new model
parameters (changeable before the simulation)

floating point type

differentiation with regards to time

declaration
comment

start value

mathematical equation (acausal)

name + default value

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
\[
\begin{align*}
(\text{thrust} - \text{mass} \cdot \text{gravity})/\text{mass} &= \text{acceleration}; \\
\text{der}(\text{mass}) &= -\text{massLossRate} \cdot \text{abs}(\text{thrust}); \\
\text{der}(\text{altitude}) &= \text{velocity}; \\
\text{der}(\text{velocity}) &= \text{acceleration};
\end{align*}
\]

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

```
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

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.

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

```modelica
function sum
    input Real arg1;
    input Real arg2;
    output Real result;
algorithm
    result := arg1+arg2;
end sum;
```
Example Modelica function call:

```modelica
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`.

Example Modelica function call:

```modelica
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`.
Inheritance

Data and behavior: field declarations, equations, and certain other contents are *copied* into the subclass
Multiple Inheritance is fine – inheriting both geometry and color

```modelica
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;

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

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

class VerticalLine extends Point;
    Real vlength;
end VerticalLine;

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:
  ```modelica
  class SameColor = Color;
  end SameColor;
  ```

Equivalent to:

```modelica
inheritance
  class SameColor extends Color;
end SameColor;
```

- Often used for introducing new names of types:
  ```modelica
  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:

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

```plaintext
  ...
  ...
```
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;
model MoonLanding

parameter Real force1 = 36350;
parameter Real force2 = 1308;
parameter Real thrustEndTime = 210;
parameter Real thrustDecreaseTime = 43.2;

Rocket apollo(name="apollo13", mass(start=1038.358));
CelestialBody moon(mass=7.382e22, radius=1.738e6, name="moon");

equation
  apollo.thrust = if (time<thrustDecreaseTime) then force1
                  else if (time<thrustEndTime) then force2
                  else 0;
  apollo.gravity = moon.g*moon.mass/(apollo.altitude+moon.radius)^2;
end Landing;


inherited parameters
Inheritance of Protected Elements

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

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

A protected element cannot be accessed via dot notation!

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

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

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

class ColoredPointWithoutInheritance
  Real x;
  Real y, z;
  protected Real red;
  protected Real blue;
  protected Real green;
equation
  red + blue + green = 1;
end ColoredPointWithoutInheritance;
```
Exercises Part III a
(15 minutes)
Exercises Part III a

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

• Open Exercises-ModelicaTutorial.pdf (also available in printed handouts)
Exercises 2.1 and 2.2 (See also next two pages)

- Open the Exercises-ModelicaTutorial.onb found in the Tutorial directory you copied at installation.

- **Exercise 2.1.** Simulate and plot the HelloWorld example. Do a slight change in the model, re-simulate and re-plot. Try command-completion, val( ), etc.

```modelica
class HelloWorld "A simple equation"
    Real x(start=1);

equation
    der(x) = -x;
end HelloWorld;
```

- Exercise 2.2: Simulate and plot VanDerPol. Do a slight change in the model, re-simulate and re-plot.

- 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: \( x(0) = 1 \)

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

Simulation in OpenModelica environment

```modelica
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.

Simulation in OpenModelica environment

```modelica
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;
```

simulate(DAEexample, stopTime = 1)
plot(x)
Exercise 2.4 – Model the system below

- Model this Simple System of Equations in Modelica

\[
\begin{align*}
\dot{x} &= 2 \cdot x \cdot y - 3 \cdot x \\
\dot{y} &= 5 \cdot y - 7 \cdot x \cdot y \\
x(0) &= 2 \\
y(0) &= 3
\end{align*}
\]
(extra) Exercise 2.5 – Functions

• a) Write a function, \texttt{sum2}, which calculates the sum of Real numbers, for a vector of arbitrary size.

• b) Write a function, \texttt{average}, which calculates the average of Real numbers, in a vector of arbitrary size. The function \texttt{average} should make use of a function call to \texttt{sum2}. 
Part III b
Discrete Events and Hybrid Systems

Picture: Courtesy Hilding Elmqvist
Modelica Hybrid Modeling

Hybrid modeling = continuous-time + discrete-time modeling

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

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

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

if \langle\text{condition}\rangle \text{ then}
  \langle\text{equations}\rangle
elsif \langle\text{condition}\rangle \text{ then}
  \langle\text{equations}\rangle
else
  \langle\text{equations}\rangle
end if;

false if \( s < 0 \)
If-equation choosing equation for \( v \)
If-expression
**Event Creation – when**

**when-equations**  (two kinds: unclocked and clocked)

- **When unclocked event**:
  ```modelica
  when <conditions> then
  <equations>
  end when;  // un-clocked version
  ```

- **When clocked event**:
  ```modelica
  when clock then
  <equations>
  end when;  // clocked version
  ```

---

**Time event**

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

Only dependent on time, can be scheduled in advance

---

**State event**

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

Related to a state. Check for zero-crossing
Generating Repeated Events by unclocked sample

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

```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;
```

Variables need to be discrete.

Creates an event after 2 s, then each 0.5 s.

`pre(...)` takes the previous value before the event.
Generating Clock Tick Events using Clock()
(clocked models, Modelica 3.3)

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

class ClockTicks

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

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

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

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

Initial conditions
Exercise 2.6 – BouncingBall

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

Clocked Synchronous Models and State Machines

and Applications for Digital Controllers
Control System Applications

Control System

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

Measurements → Control Computing → Controller Outputs

Sensors → Control Computing → Actuators
Control Theory Perspective
Feedback Control System

\[ r(t) + e(t) \quad \text{Controller} \quad u(t) \quad \text{Plant (Physical System)} \quad y(t) \]

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

Usual Objective

Plant output should follow the reference signal.
1. **Discrete-time** controller + **continuous-time** plant $\equiv$ *hybrid system* or *sampled-data system*

2. Interface between digital and analog world: Analog to Digital and Digital to Analog Converters (ADC and DAC).

3. ADC$\rightarrow$Algorithm$\rightarrow$DAC is synchronous (zero-delay model!)

4. A *clock* controls the *sampling instants*. Usually *periodic sampling*. 
Controller with Sampled Data-Systems
(unclocked models, using pre() and sample() )

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

// 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)
Controller with Clocked Synchronous Constructs
clocked models using Clock(), previous(), hold() in Modelica 3.3

```modelica

// time-discrete controller
when Clock() then
    E*xd = A*previous(xd) + B*yd;
    ud = C*previous(xd) + D*yd;
end when;

// plant
y = g(x);

// sample continuous signal
yd = sample(y, Clock(3));

// hold
u = hold(ud);

// plant
0 = f(der(x), x, u);

// sample continuous signal
yd = sample(y, Clock(3));
```

\[ t_i \in 0,3,6,\ldots, \quad i = 0,1,2,\ldots \]
\[ yd(t_i) = y(t_i) \]
\[ E \cdot xd(t_i) = A \cdot xd(t_{i-1}) + B \cdot yd(t_i) \]
\[ ud(t_i) = C \cdot xd(t_{i-1}) + D \cdot yd(t_i) \]
\[ u(t) = ud(t_i), \quad t_i \leq t < t_{i+1} \]
Unclocked Variables in Modelica 3.2

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

Piecewise-constant variables \( m(t) \) are constant inside each

\[ t_i \leq t < t_{i+1} \]
Clock variables \( r(t_i) \) are of base type Real, Integer, etc. They are uniquely associated with a clock \( c(t_i) \). Can only be accessed when its clock is active (ticks).

Clock variables \( c(t_i) \) are of base type Clock. They are defined by constructors such as \texttt{Clock(3)} or by clock operators relatively to other clocks.
Clocked Synchronous Extension in Modelica 3.3

// time-discrete controller
when Clock() then
  E * xd = A * previous(xd) + B * yd;
  ud = C * previous(xd) + D * yd;
end when;

// sample continuous signal
yd = sample(y, Clock(3));

// plant
θ = f(der(x), x, u);
y = g(x);

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

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

Continuous variables are Real numbers defined as piecewise continuous functions of time. Piecewise-constant variables \( m(t) \) are constant inside each interval \( t_i \leq t < t_{i+1} \).
State Machines in Modelica 3.3: Simple Example

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

transition(state1, state2, i > 10, immediate=false);
transition(state2, state1, i < 1, immediate=false);
initialState(state1);
end Simple_NoAnnotations;
Hierarchical and Parallel Composition of Modelica State Machine Models

Semantics of Modelica state machines (and example above) inspired by Florence Maraninchi & Yann Rémond’s “Mode-Automata” and by Marc Pouzet’s Lucid Synchrone 3.0.
Hierarchical and Parallel Composition

Semantics of Modelica state machines (and example above) inspired by Florence Maraninchi & Yann Rémond’s “Mode-Automata” and by Marc Pouzet’s Lucid Synchrone 3.0.
Part IV

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

**connector class**

- **connector Pin**

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

  - keyword `flow` indicates that currents of connected pins sum to zero.
  - an instance `pin` of class `Pin`

**connector class**

- **connector Flange**

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

  - an instance `flange` of class `Flange`

**connector class**
electrical connector

- **connector class**

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

  - an instance `pin` of class `Pin`

**connector class**
mechanical connector
The flow prefix

Three possible kinds of variables in connectors:

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

Coupling

- Equality coupling, for potential variables
- Sum-to-zero coupling, for flow variables

The value of a flow variable is positive when the current or the flow is into the component
# Physical Connector Classes Based on Energy Flow

<table>
<thead>
<tr>
<th>Domain Type</th>
<th>Potential</th>
<th>Flow</th>
<th>Carrier</th>
<th>Modelica Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>Electrical</td>
<td>Voltage</td>
<td>Current</td>
<td>Charge</td>
<td>Electrical. Analog</td>
</tr>
<tr>
<td>Translational</td>
<td>Position</td>
<td>Force</td>
<td>Linear momentum</td>
<td>Mechanical. Translational</td>
</tr>
<tr>
<td>Rotational</td>
<td>Angle</td>
<td>Torque</td>
<td>Angular momentum</td>
<td>Mechanical. Rotational</td>
</tr>
<tr>
<td>Magnetic</td>
<td>Magnetic potential</td>
<td>Magnetic flux rate</td>
<td>Magnetic flux</td>
<td>Magnetic</td>
</tr>
<tr>
<td>Hydraulic</td>
<td>Pressure</td>
<td>Volume flow</td>
<td>Volume</td>
<td>OpenHydraulics</td>
</tr>
<tr>
<td>Heat</td>
<td>Temperature</td>
<td>Heat flow</td>
<td>Heat</td>
<td>HeatFlow1D</td>
</tr>
<tr>
<td>Chemical</td>
<td>Chemical potential</td>
<td>Particle flow</td>
<td>Particles</td>
<td>Chemical</td>
</tr>
<tr>
<td>Pneumatic</td>
<td>Pressure</td>
<td>Mass flow</td>
<td>Air</td>
<td>PneuLibLight</td>
</tr>
</tbody>
</table>
Connections between connectors are realized as *equations* in Modelica:

```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

```modelica
pin1.v = pin2.v;
pin1.i + pin2.i = 0;
```
Connection Equations

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

Corresponds to

\[
\begin{align*}
\text{pin1.v} &= \text{pin2.v}; \\
\text{pin1.i} + \text{pin2.i} &= 0;
\end{align*}
\]

Multiple connections are possible:

\[
\text{connect}(\text{pin1}, \text{pin2}); \quad \text{connect}(\text{pin1}, \text{pin3}); \quad \ldots \quad \text{connect}(\text{pin1}, \text{pinN});
\]

Each primitive connection set of potential 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:

\[
i_1 + i_2 + \ldots + (-i_k) + \ldots i_n = 0
\]
The base class `TwoPin` has two connectors `p` and `n` for positive and negative pins respectively.

```
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
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;
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);
  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](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 – for magnetic applications
- **Mechanics**  
  Library for mechanical systems
- **Media**  
  Media models for liquids and gases
- **Slunits**  
  Type definitions based on SI units according to ISO 31-1992
- **Stategraph**  
  Hierarchical state machines (analogous to Statecharts)
- **Thermal**  
  Components for thermal systems
- **Utilities**  
  Utility functions especially for scripting
Modelica.Blocks

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

Examples:
Modelica.Electrical

Electrical components for building analog, digital, and multiphase circuits

Examples:
Modelica.Mechanics

Package containing components for mechanical systems

Subpackages:

- Rotational 1-dimensional rotational mechanical components
- Translational 1-dimensional translational mechanical components
- MultiBody 3-dimensional mechanical components
PNlib - An Advanced Petri Net Library for Hybrid Process Modeling

xHPN: Extended Hybrid Petri Nets

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

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

Arcs
- "normal" arc
- inhibitor arc
- test arc
- read arc
Other Free Libraries
Up to date list at:  https://www.modelica.org/libraries

- WasteWater       Wastewater treatment plants, 2003
- ATPlus           Building simulation and control (fuzzy control included), 2005
- 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
- VirtualLabBuilder Implementation of virtual labs, 2007
- PowerSystems     Power systems in transient and steady-state mode
- ...
Some Commercial Libraries
Up to date list at: https://www.modelica.org/libraries

- Air Conditioning
- Electric Power
- Fuel Cell
- Heat Exchanger
- Hydro Power
- Liquid Cooling
- Thermal Power
- Vapor Cycle
- Battery
- Belts
- Engine
- ...

- Powertrain
- SmartElectricDrives
- VehicleDynamics
- Hydraulics
- Pneumatics
- Engine Dynamics
- Environmental Control
- CombiPlant
- …
- (there are many more)
Connecting Components from Multiple Domains

- Block domain
- Mechanical domain
- Electrical domain

model Generator
Modelica.Mechanics.Rotational.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.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;
A DC motor can be thought of as an electrical circuit which also contains an electromechanical component.

```model DCMotor
    Resistor R(R=100);
    Inductor L(L=100);
    VsourceDC DC(f=10);
    Ground G;
    EMF emf(k=10, J=10, b=2);
    Inertia load;

    equation
        connect(DC.p, R.n);
        connect(R.p, L.n);
        connect(L.p, emf.n);
        connect(emf.p, DC.n);
        connect(DC.n, G.p);
        connect(emf.flange, load.flange);

end DCMotor;
```
Part IV
Sensitivity Analysis

using OpenModelica
OMSens – Multi-Parameter Sensitivity Analysis

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

Heatmap visualization

1. Models and scripts handling
   - Python

2. Modelica
   - OpenModelica Models and Scripts

3. Optimization
   - CURVI (Fortran)

4. OS
   - Linux

5. Sensitivity indices
   - Python

6. Parameter sweeping
   - Python

Simultaneous Sensitivity Analysis
Individual Sensitivity Analysis
Introduction to Sensitivity Analysis

- Sensitivity of nonlinear systems in the form of ODEs
  - Undergo noticeable *dynamic changes* in response to *small perturbations* in the parameters.

- OO-languages (Modelica)
  - Systematic treatment of the problem
  - Clear, unambiguous access to parameters, variables and simulation configuration.
  - Reusable frameworks to manipulate models as black boxes.

- Varied options to use internal knowledge about model structure
Approaches to Sensitivity Analysis

- **Individual** analysis:
  - One parameter perturbed at a time
  - Ignores combinations of perturbations

- **Simultaneous** analysis:
  - All possible combinations not feasible
    - Would give combinatorial explosion of parameter settings
  - Find “optimal” combinations of perturbations
    - “Smallest simultaneous perturbations that produce largest deviations”
  - Typically: optimization-based strategies
CURVIF: robust derivative-free optimization algorithm

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

model LotkaVolterra "This is the typical equation-oriented model"

  parameter Real alpha=0.1 "Reproduction rate of prey";
  parameter Real beta=0.02 "Mortality rate of predator per prey";
  parameter Real gamma=0.4 "Mortality rate of predator";
  parameter Real delta=0.02 "Reproduction rate of predator per prey";
  parameter Real prey_pop_init=10 "Initial prey population";
  parameter Real pred_pop_init=10 "Initial predator population";

  Real prey_pop(start=prey_pop_init) "Prey population";
  Real pred_pop(start=pred_pop_init) "Predator population";

initial equation
  prey_pop = prey_pop_init;
  pred_pop = pred_pop_init;

equation
  der(prey_pop) = prey_pop*(alpha-beta*pred_pop);
  der(pred_pop) = pred_pop*(delta*prey_pop-gamma);

end LotkaVolterra ;
Preparations for Sensitivity Analysis Exercises

Assuming that you already have latest OpenModelica nightly installed. (e.g.1.16.0)

NOTE: Change the paths according to your installation whenever needed in the following commands.

Step1. Install Anaconda (most easily in c:\anaconda3)

===================
For 64-bit use https://repo.anaconda.com/archive/Anaconda3-2019.10-Windows-x86_64.exe
For 32-bit use https://repo.anaconda.com/archive/Anaconda3-2019.10-Windows-x86.exe

Follow the installer and install using the default options i.e., don’t change anything (we recommend installing in C:\Anaconda3).

Step 2. Install libpython

====================
Open windows command prompt CMD and type/copy the following commands,
(this is one line😊

set PATH=C:\Anaconda3;C:\Anaconda3\Library\mingw-w64\bin;C:\Anaconda3\Library\bin;C:\Anaconda3\Scripts;
C:\Program Files\OpenModelica1.16.0-dev-64bit\tools\msys\mingw64\bin;%PATH%
conda install libpython

Step 3. Compile OMSens Backend

=========================
Use the same windows command prompt from last step and copy/run the commands (one at a time)

cd "C:\Program Files\OpenModelica1.16.0-dev-64bit\OMSens\fortran_interface"
gfortran -c Rutf.for Rut.for Curvif.for
f2py.exe -c -I. Curvif.o Rutf.o Rut.o -m curvif_simplified curvif_simplified.pyf Curvif_simplified.f90 --compiler=mingw32
"C:\Program Files\OpenModelica1.16.0-dev-64bit\bin\OMEdit.exe"

Step 4: Run OMEdit with OMSens

Now use the OMEdit we just started from the CMD command window (just push enter)
Load the LotkaVolterra model, open it, run it e.g. 100 seconds
and select the OMSens functionality from the top menu item i.e., "Sensitivity Optimization"

More info in the file:
OMSens Example_Exercise_ Lotka-Volterra.pdf
OMSens Exercise – Locate Python
Select Analysis type

Individual Parameter Based Sensitivity Analysis
Multi-parameter Sweep
Vectorial Parameter Based Sensitivity Analysis

Individual Sensitivity Analysis Results

Relative (REL)    Root Mean Square (RMS)

Description: The REL index calculates the change of a state variable (at the end of a simulation) with and without a parameter perturbation (at the beginning of the simulation). It can be used to rank parameters according to their impact on a state variable at a target final time.

Results: Matrix    Heatmap    State Variable IDs    Parameter IDs

Results can be found in:
C:/Users/petr27/AppData/Local/Temp/OpenModelica/OMEdit/omsens_results/indiv_results/2020-02-03/15_4_42/results

Open    Ok
OMSens Exercise – results from individual analysis

More info in the file:
OMSens Example_Exercise_Lotka-Volterra.pdf
Part Vb
More
Graphical Modeling Exercises

using
OpenModelica
Graphical Modeling - Using Drag and Drop Composition

OMEdit - OpenModelica Connection Editor - [DCMotor]

Libraries
- Complex
- Modelica
- UsersGuide
- Blocks
- ComplexBlocks
- StateGraph
- Electrical
- Magnetic
- Mechanics
- MultiBody
- Rotational
- UsersGuide
- Examples
- Components
- Fixed
- Inertia

Diagram View
- DCMotor
- Parameters:
  - \( \text{emf} \)
  - \( \text{deg} \): -0.3403
  - \( \text{fixed} \)
  - \( \text{flange} \)
  - \( \text{i} \)
  - \( \text{A} \): -0.533507
  - \( \text{internalSupport} \)
  - \( \text{k} \)
  - \( \text{N.m/} \)
  - \( \text{n} \)
  - \( \text{N.m/} \)
  - \( \text{p} \)
  - \( \text{deg} \)
  - \( \text{useSupport} \)
  - \( \text{v} \)
  - \( \text{V} \): -0.3403
  - \( \text{w} \)
  - \( \text{rad/s} \): -0.3403

Variables
- \( \text{ground1} \)
- \( \text{inductor1} \)
- \( \text{resistor1} \)
- \( \text{shunt1} \)

Welcome
- Modeling
- Plotting

X: -35.03
Y: -38.14
Graphical Modeling Animation – DCMotor
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```

Multi-Domain (Electro-Mechanical) Modelica Model
## Corresponding DCMotor Model Equations

The following equations are automatically derived from the Modelica model:

<table>
<thead>
<tr>
<th>Equation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 == DC.p.i + R.n.i</td>
<td>EM.u == EM.p.v - EM.n.v</td>
</tr>
<tr>
<td>DC.p.v == R.n.v</td>
<td>0 == EM.p.i + EM.n.i</td>
</tr>
<tr>
<td>0 == R.p.i + L.n.i</td>
<td>EM.u == EM.k * EM.omega</td>
</tr>
<tr>
<td>R.p.v == L.n.v</td>
<td>EM.i == EM.M / EM.k</td>
</tr>
<tr>
<td>0 == L.p.i + EM.n.i</td>
<td>EM.J * EM.omega == EM.M - EM.b * EM.omega</td>
</tr>
<tr>
<td>L.p.v == EM.n.v</td>
<td>DC.u == DC.p.v - DC.n.v</td>
</tr>
<tr>
<td>0 == DC.p.i + DC.n.i</td>
<td>EM.p.v == DC.n.v</td>
</tr>
<tr>
<td>0 == EM.p.i + DC.n.i</td>
<td>DC.i == DC.p.i</td>
</tr>
<tr>
<td>0 == DC.n.i + G.p.i</td>
<td>DC.u == DC.Amp * Sin[2 * pi * DC.f * t]</td>
</tr>
<tr>
<td>DC.n.v == G.p.v</td>
<td>(load component not included)</td>
</tr>
</tbody>
</table>

### Automatic transformation to ODE or DAE for simulation:

\[
\frac{dx}{dt} = f(x, u, t) \quad g\left[\frac{dx}{dt}, x, u, t\right] = 0
\]
Exercise 3.1

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

- Simulate it for 15s and plot the variables for the outgoing rotational speed on the inertia axis and the voltage on the voltage source (denoted `u` in the figure) in the same plot.
Exercise 3.2

- If there is enough time: Add a torsional spring to the outgoing shaft and another inertia element. Simulate again and see the results. Adjust some parameters to make a rather stiff spring.
Exercise 3.3

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

- **OpenModelica**
  - [www.openmodelica.org](http://www.openmodelica.org)

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

- **Books**
  - Introduction to Modelica, Michael Tiller
Summary

Multi-Domain Modeling

Visual Acausal Component Modeling

Typed Declarative Textual Language

Hybrid Modeling

Thanks for listening!