Principles of Equation-Based Object-Oriented Modeling and Languages

Module A: EOO Languages and Modelica Fundamentals

Mini-course, Scuola Superiore Sant'Anna, Pisa, Italy.
December 9-10, 2014

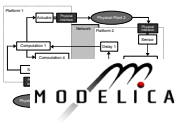
David Broman

Associate Professor, KTH Royal Institute of Technology

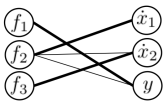
Assistant Research Engineer, University of California, Berkeley

Some of the slides are based a OSMC tutorials and contributed by Peter Fritzson (Based on book and lecture nodes), David Broman, Emma Larsdotter Nilsson, Peter Bunus, Adrian Pop, Jan Brugård, Mohsen Torabzadeh-Tari, and Adeel Asghar.

Course Structure



Module A
EOO Languages and Modelica Fundamentals



Module B
DAEs and Algorithms in EOO Languages

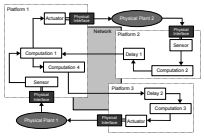
Module C
Modelyze – Defining Equation-Based DSLs

Module D
Co-simulation and the Functional Mock-up Interface

Agenda

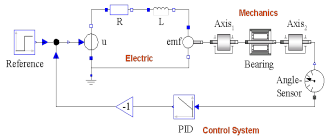
Part I

EOO Languages for CPS



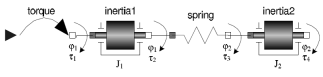
Part II

Modelica Overview



Part III

Acausal Connection Semantics in Modelica

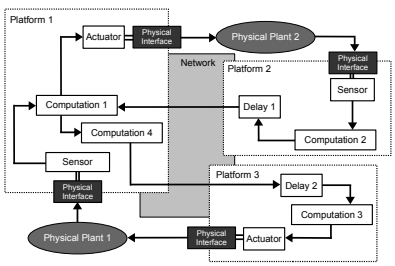


Part IV

Modeling in OpenModelica

Part I

EOO Languages for CPS



Cyber-Physical Systems

Automotive

Process Industry and Industrial Automation

Aircraft

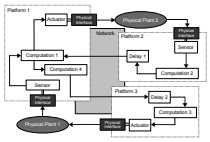
Cyber-Physical Systems Design

Model

$$\begin{aligned} J_1\dot{\omega}_1 &= M_v - M_1 \\ J_2\dot{\omega}_2 &= M_h - M_2 \\ \omega_1 &= -r\omega_2 \\ M_1 &= -r^{-1}M_2 \end{aligned}$$

Equation-based model

Simulation with timing properties



Various models of computation (MoC)

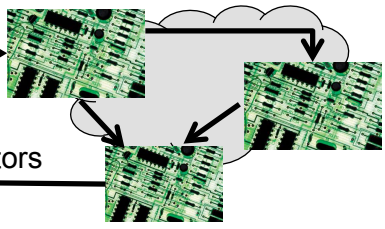
Modeling

Modeling

System

Sensors

Actuators



Physical system (the plant)

Cyber system: Computation (embedded) + Networking

Equation-Based Object-Oriented (EOO) Languages

Domain-Specific Language (DSL)

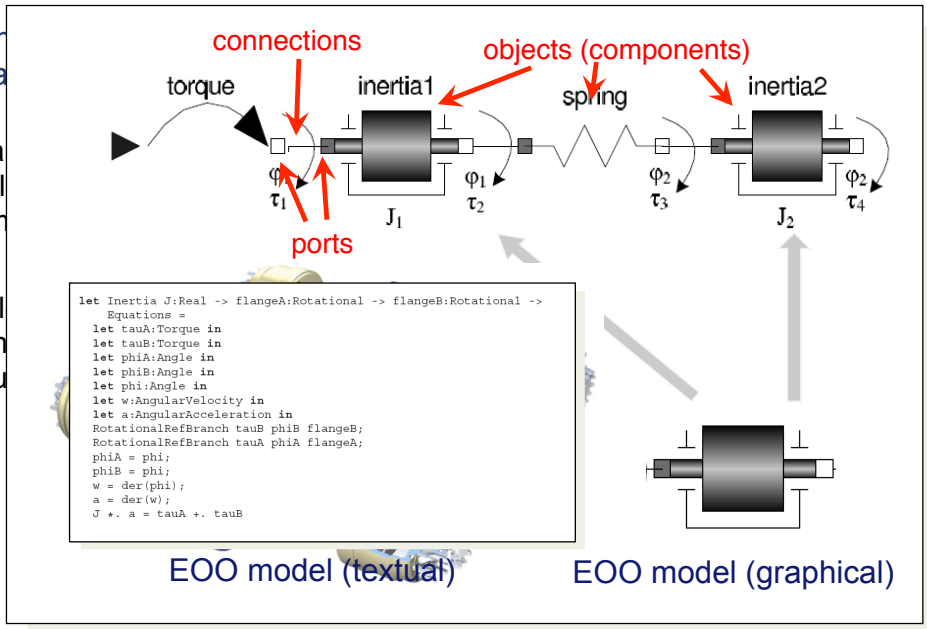
- Primarily domain: Modeling of physical systems
- Multiple physical domains: e.g., mechanical, electrical, hydraulic

Equation-Based Object-Oriented (EOO)

Models and Objects

- Object in e.g., Java, C++: object = data + methods
- Objects in EOO languages: object = data + equations

Equation-Based Object-Oriented (EOO) Languages



Equation-Based Object-Oriented (EOO) Languages

Domain-Specific Language (DSL)

- Primarily domain: Modeling of physical systems
- Multiple physical domains: e.g., mechanical, electrical, hydraulic

Equation-Based Object-Oriented (EOO)

Models and Objects

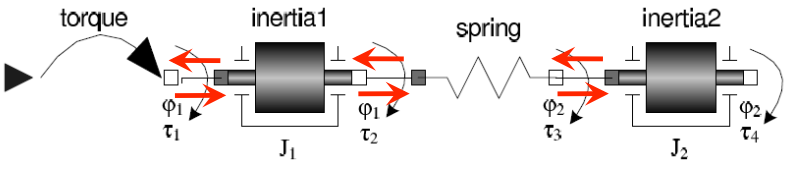
- Object in e.g., Java, C++: object = data + methods
- Objects in EOO languages: object = data + equations

Acausality

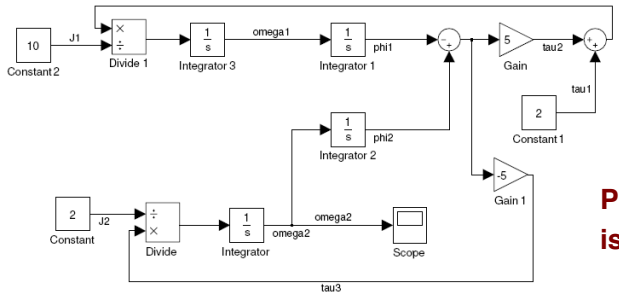
- At the equation-level
  u = R * i
- At the object connection level

Equation-Based Object-Oriented (EOO) Languages

Direction not determined at modeling time



acausal (non-causal)



Physical topology is lost

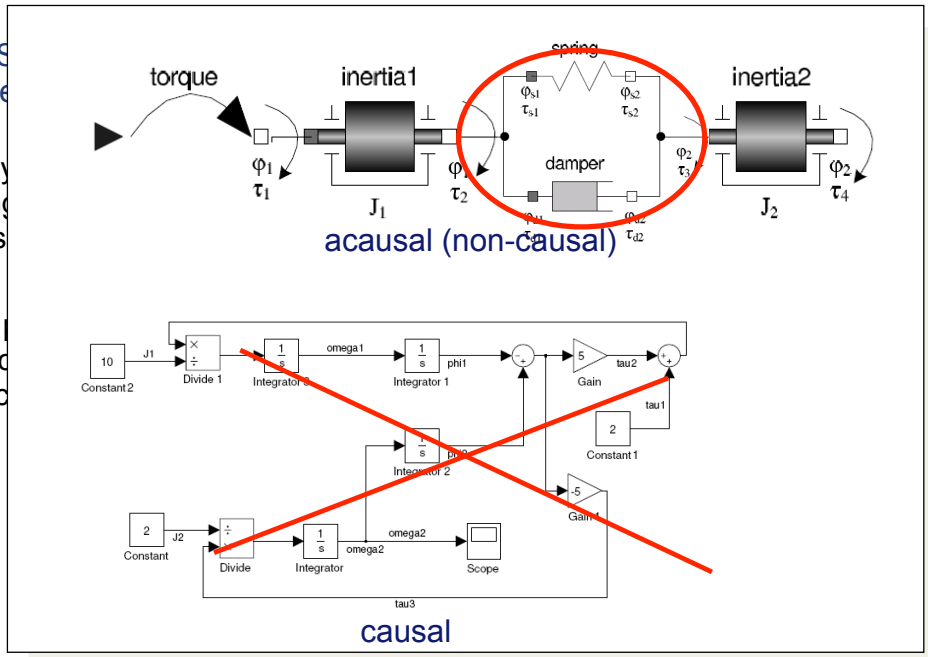
causal

Equation-Based Object-Oriented (EOO) Languages

Domain-S
Language

• Primarily
Modelin
systems

• Multiple
e.g., mec
hydraulic



s

a, C++:
thods

nguages:
uations

el

Equation-Based Object-Oriented (EOO) Languages

Domain-Specific Language (DSL)

• Primarily domain: Modeling of physical systems

• Multiple physical domains: e.g., mechanical, electrical, hydraulic

• Modelica
• VHDL-AMS
• gPROMS
• Modelyze
• …

Equation-Based Object-Oriented (EOO)

Models and Objects

• Object in e.g., Java, C++: object = data + methods

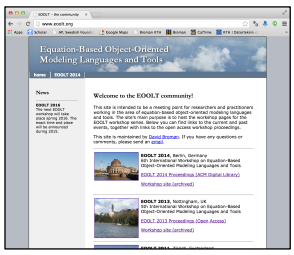
• Objects in EOO languages: object = data + equations

Acausality

• At the equation-level
u = R * i

• At the object connection level

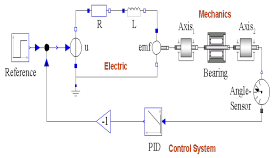
Conferences and Workshops



www.eoolt.org

The International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools (EOOLT)
The main workshop concerning EOO language and tool research.

- First workshop in year 2007
- 6 workshops so far.
- Proceedings published by ACM (year 2014)



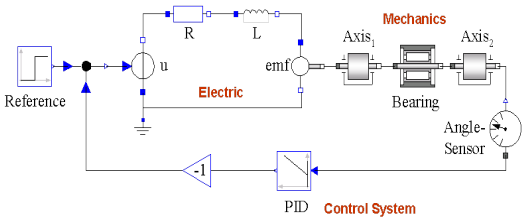
www.modelica.org

The Modelica Conference)
The main conference concerning Modelica modeling and Modelica tools.

- 10 international conferences (2000-2014)
- Proceedings published by Linköping Electronic Press.

Part II

Modelica Overview



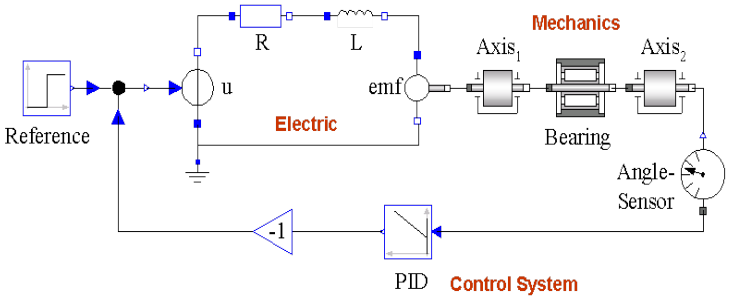
What is Modelica?

A language for modeling of complex physical systems

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

What is Modelica?

A language for modeling of complex physical systems



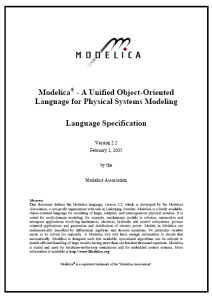
Primary designed for simulation, but there are also other usages of models, e.g. optimization.

What is Modelica?

A language for modeling of complex physical systems

i.e., Modelica is not a tool

Free, open language specification:


MODELICA

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

Language Specification

Available at: www.modelica.org

Modelica Tools

Commercial Environments

Dymola by Dassault Systemes

SimulationX by ITI GmbH

LMS Imagine.Lab AMESim by LMS

MapleSim by Maplesoft

MOSILAB by Fraunhofer FIRST

CyModelica by CyDesign Labs

OPTIMICA Studio by Modelon AB

MWorks by Suzhou Tongyuan

Wolfram SystemModeler by Wolfram

Free Environments

OpenModelica supported by OSMC

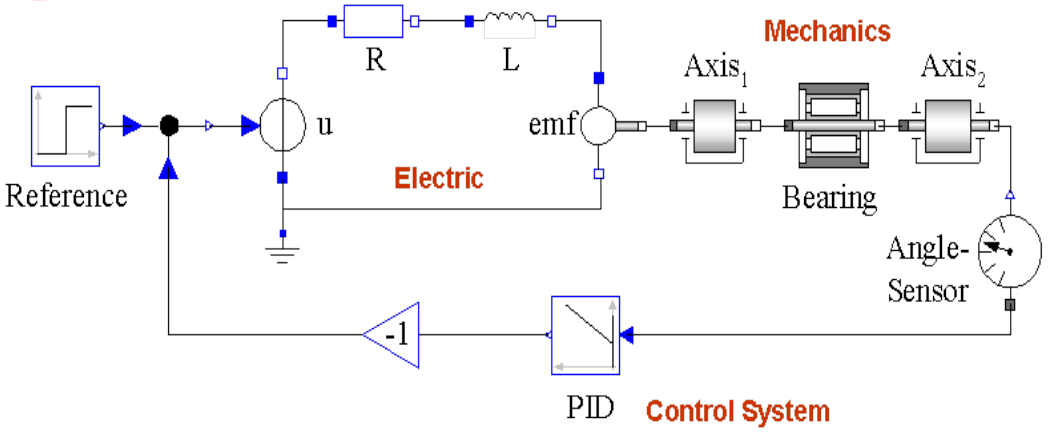
Jmodelica.org supported by Modelon

Modelicac (part of Scilab)

SimForge

What is special about Modelica?

Multi-Domain Modeling



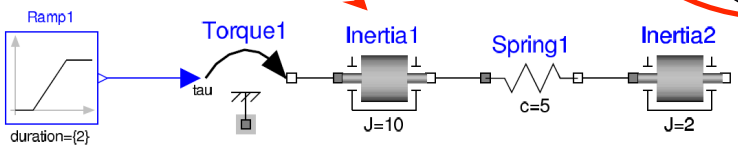
What is special about Modelica?

Multi-Domain Modeling

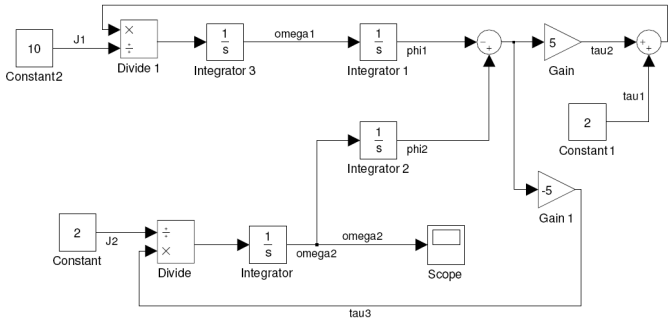
Visual Acausal Component Modeling

Keeps the physical structure

Acausal model (Modelica)



Causal block-based model (Simulink)

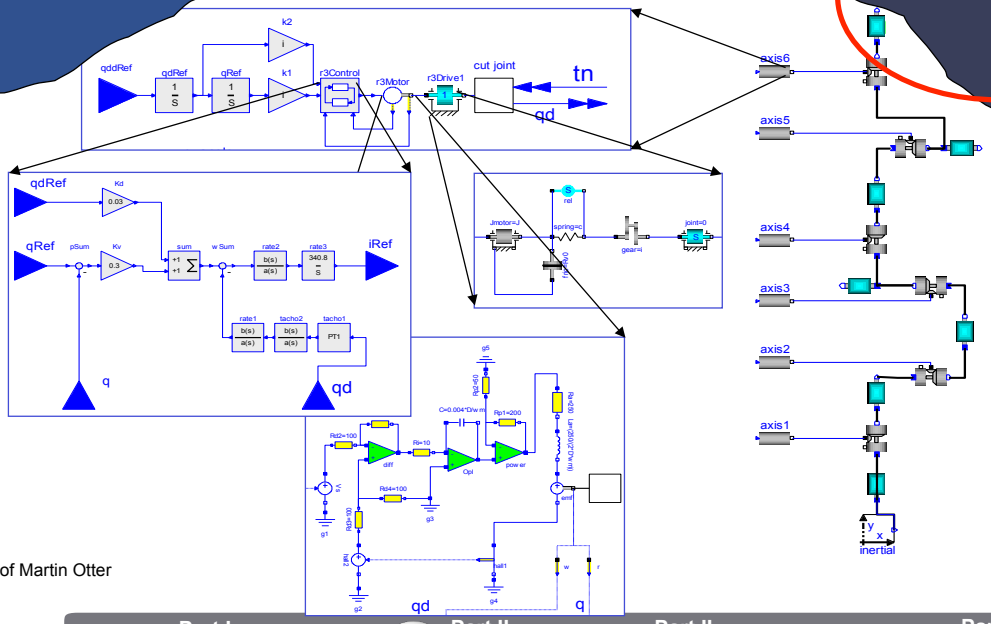


What is special about Modelica?

Multi-Domain Modeling

Hierarchical system modeling

Visual Acausal Component Modeling



Courtesy of Martin Otter

What is special about Modelica?

Multi-Domain Modeling

Visual Acausal 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)

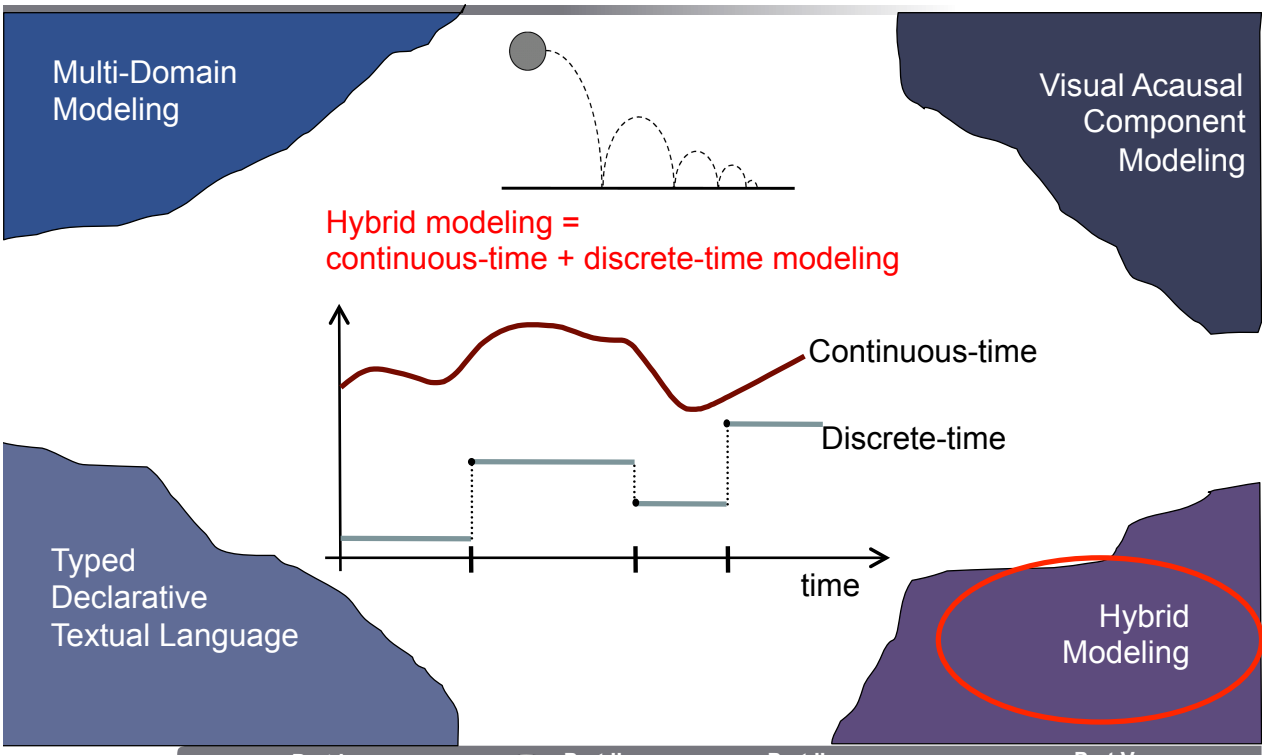
Typed Declarative Textual Language

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

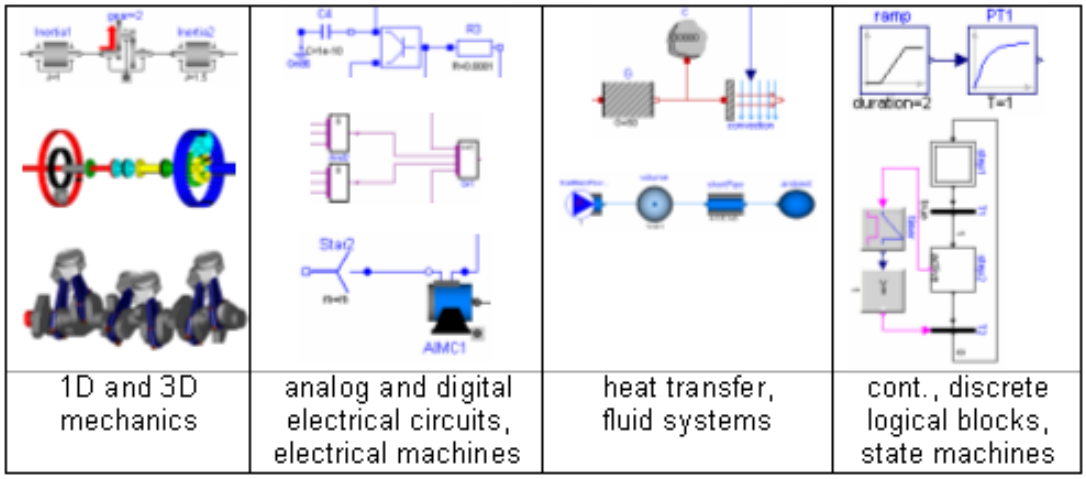
What is special about Modelica?



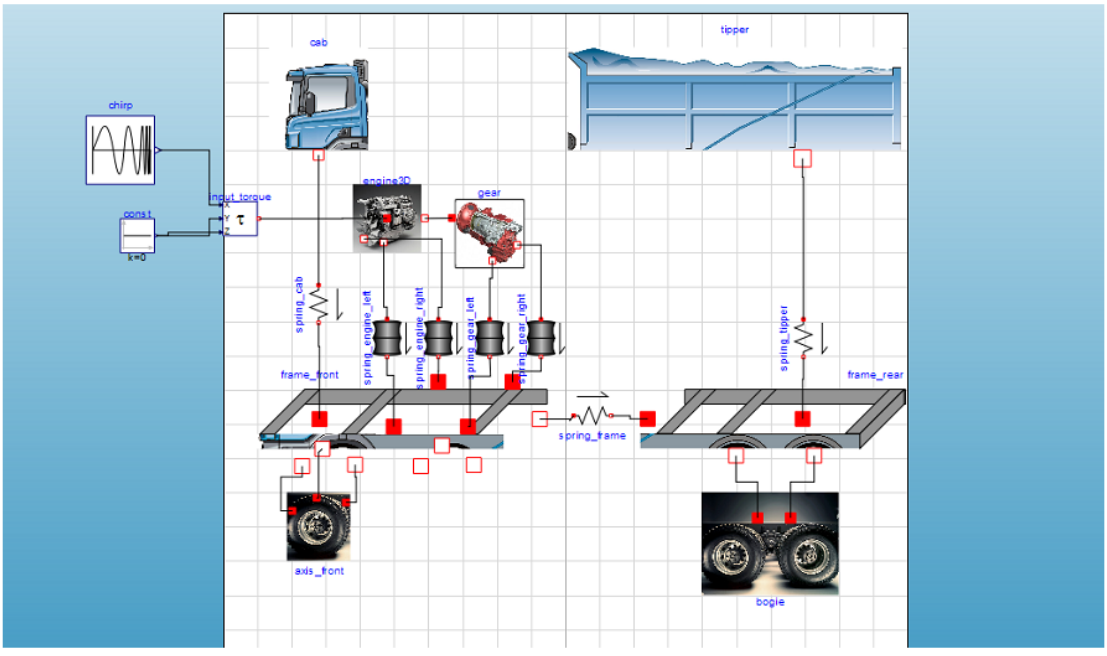
Some Domains

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	Presure	Mass flow	Air	`PneuLibLight`

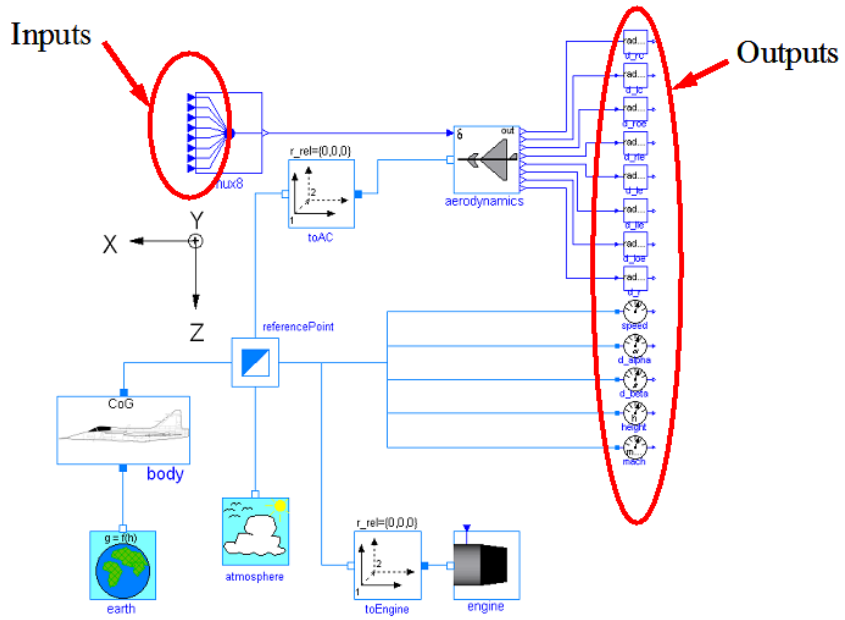
Modelica Standard Library



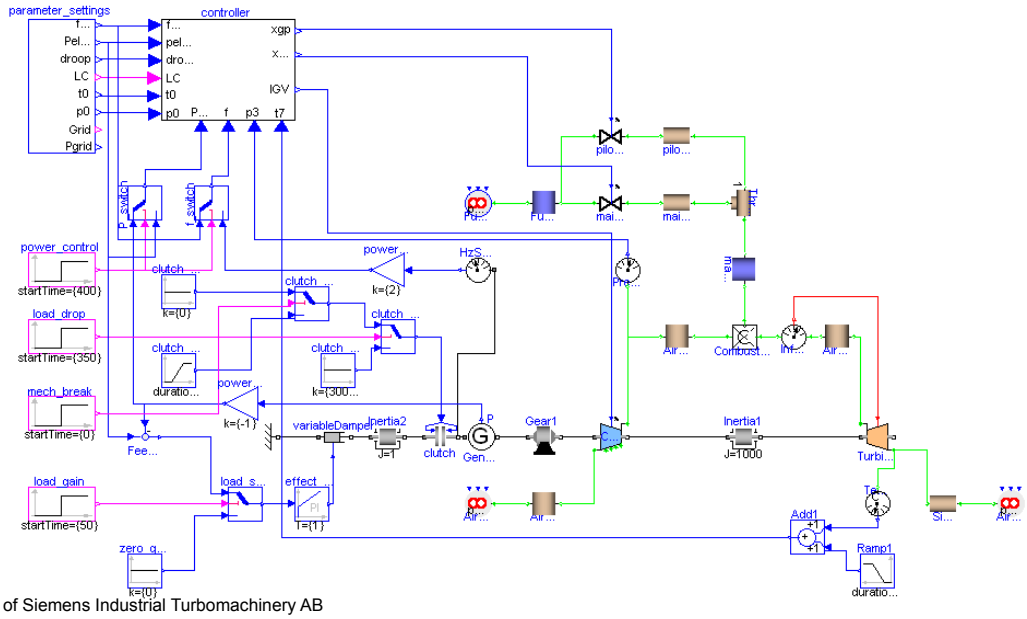
Modelica in Autmotive Industry



Modelica in Avionics



Modelica in Power Generation GTX Gas Turbine



Courtesy of Siemens Industrial Turbomachinery AB

Developed by MathCore for Siemens

Brief Modelica History

Modelica design group meetings

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

Modelica Language Versions

- v1.0 (1997), v2.0 (2002) v.2.2 (2005) v.3.0 (2007) 3.1 (2009) 3.2 (2010), 3.2 revision 1 (2012), 3.3 revision 1 (2014)

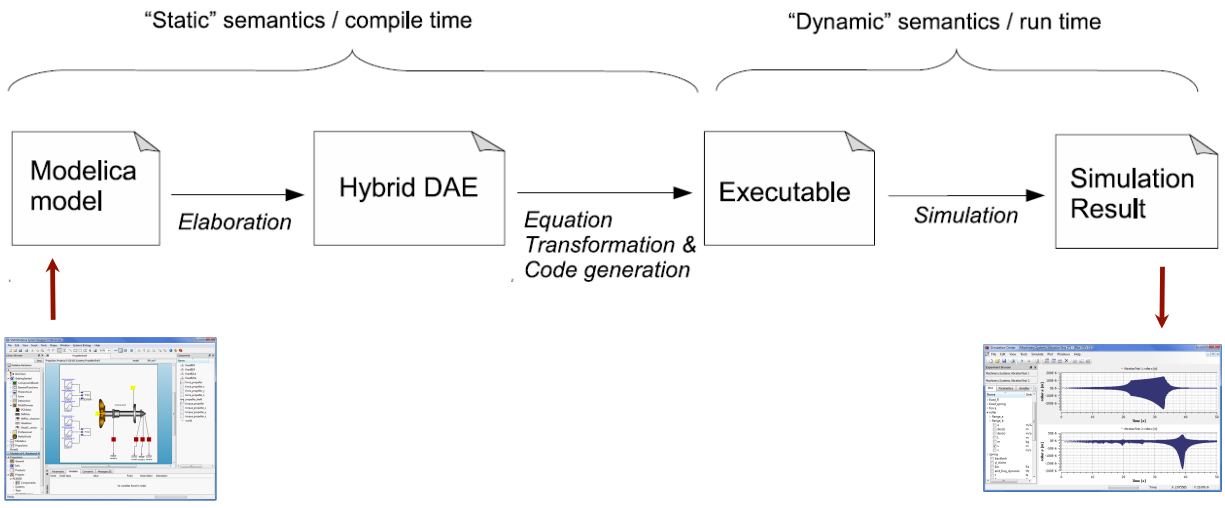
Modelica Association established 2000

- Open, non-profit organization

Modelica Conferences

- 10 international conferences (2000-2014)

Typical Simulation Process



Simple model - Hello World!

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

Name of model

Initial condition

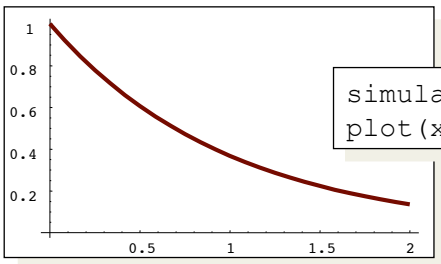
Continuous-time variable

Parameter, constant during simulation

Differential equation

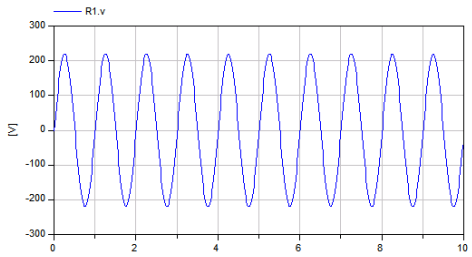
```
model HelloWorld "A simple equation"
  Real x(start=1);
  parameter Real a = -1;
equation
  der(x)= a*x;
end HelloWorld;
```

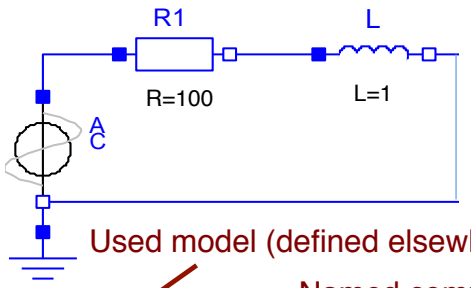
Simulation in OpenModelica environment




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


Textual and Graphical Models





Used model (defined elsewhere)

Named component = model instance

Modification of parameter value

Connect equations

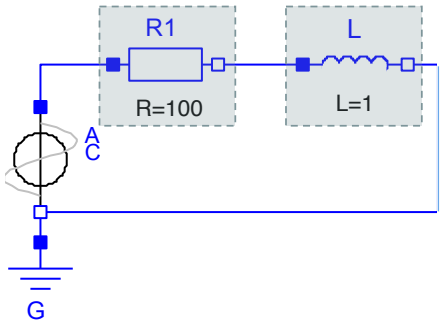
```
model Circuit
protected
  replaceable Resistor R1(R=10);
  replaceable Inductor L(L=0.1);
  VsourceAC AC;
  Ground G;
equation
  connect(AC.p, R1.p);
  connect(R1.n, L.p);
  connect(L.n, AC.n);
  connect(AC.n, G.p);
end Circuit;
```

Equations and Inheritance

```
model TwoPin
  Pin p;
  Pin n;
  Real v;
  Real i;
equation
  v = p.v - n.v;
  0 = p.i + n.i;
  i = p.i;
end TwoPin;
```

Pin **p**, **n** and Reals **v** and **i** are copied to the subclass

Equations are copied as well.



Inherits equations and components from `TwoPin`

```
model Resistor
  extends TwoPin;
  Real R = 100;
equation
  R*i = v;
end Resistor;
```

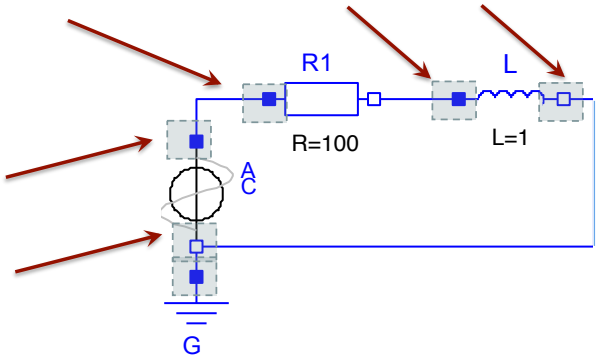
Differential equation

Algebraic equation

```
model Inductor
  extends TwoPin;
  Real L = 1;
equation
  L*der(i) = v;
end Inductor;
```


Connectors (Ports)

```
model TwoPin
  Pin p;
  Pin n;
  Real v;
  Real i;
equation
  v = p.v - n.v;
  0 = p.i + n.i;
  i = p.i;
end TwoPin;
```

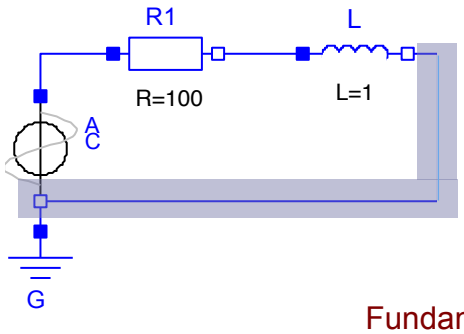


Connectors are instances of a `connector` class.

```
connector Pin
  Real v;
  flow Real i;
end Pin;
```

Connections and Flow Variables

```
model Circuit
protected
  replaceable Resistor R1(R=10);
  replaceable Inductor L(L=0.1);
  VsourceAC AC;
  Ground G;
equation
  connect(AC.p, R1.p);
  connect(R1.n, L.p);
  connect(L.n, AC.n);
  connect(AC.n, G.p);
end Circuit;
```




```
connector Pin
  Real v;
  flow Real i;
end Pin;
```

Equations from potential variables:

```
L.n.v = AC.n.v
AC.n.v = G.p.v
```

Equation from flow variables:

```
L.n.i + AC.n.i + G.p.i = 0
```

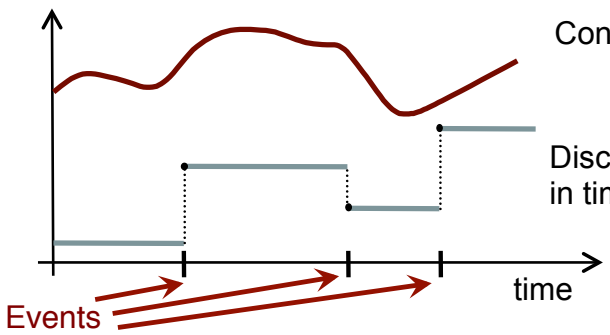
Fundamental concept making acausal modeling work (simplified)

The resulting equation system is an DAE.

(differential-algebraic equations).

Hybrid Modeling

Hybrid modeling = continuous-time + discrete changes (events)
(Using Modelica terminology)




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

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

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

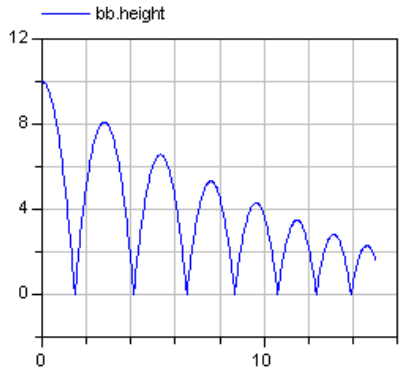
```
when sin(x) > 0.5 then
  ...
end when;
```

Related to a state. Check for zero-crossing

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

Modelica – large and complex

We have just “scratched on the surface of the language”

Examples of the features which has not been covered

- Functions and algorithm sections
- Arrays and matrices
- Inner / outer variables (lookup in instance hierarchy)
- Annotations
- Loop constructs
- Partial classes
- Packages, blocks...

And much more...

Part III

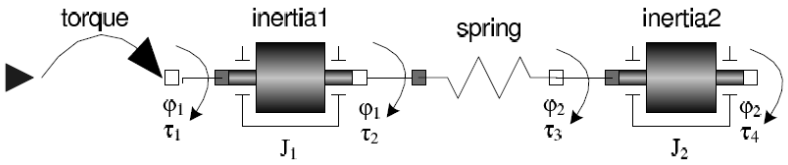
Modeling in OpenModelica

Examples are based on Michael Tiller’s new book:
http://beta.book.xogeny.com/

OpenModelica Website
http://www.openmodelica.org/

Part III

Acausal Connection Semantics in Modelica



Formalization of Connection Semantics

The following formalizes *port-based* connection semantics (as in Modelica).
An alternative is a *node-based* approach, used in e.g. Modelyze
[Broman&Nilsson, PADL 2012]

Structure

(M, P, σ, C, ρ)

Set of model instances	M
Set of ports	P
Port to model mapping	$\sigma : P \rightarrow M$
Set of port connection sets	$C \subseteq \mathcal{P}(P)$
Parent model instance mapping	$\rho : M \rightarrow M \cup \{\top\}$

Init and Behavior Equations

$(X, pvar, fvar, g, iq, bq)$

Set of variables	$X \subseteq \mathbb{X}$
Set of equations	$\mathbb{E}$
Port to potential variable mapping	$pvar : P \rightarrow X$
Port to flow variable mapping	$fvar : P \rightarrow X$
Initial guess value	$g : X \rightarrow \mathbb{R}$
Initial equations	$iq : M \rightarrow \mathcal{P}(\mathbb{E})$
Behavior equations	$bq : M \rightarrow \mathcal{P}(\mathbb{E})$

Generation of Connect Equations

Not itself

Model instance's parent instance?

Outside port?

Sum-to-zero equations

$$sign(p, c) = \begin{cases} -1 & \text{if } \exists q \in c.\ p \neq q \wedge \rho(\sigma(q)) = \sigma(p) \\ 1 & \text{otherwise.} \end{cases}$$

$$sumtozero : C \rightarrow \mathbb{E}$$

$$\begin{aligned} &sumtozero(c) = \\ &\quad \text{let } \{p_1, \ldots, p_n\} = c \text{ in} \\ &\quad sign(p_1, c) \cdot fvar(p_1) + \ldots + sign(p_n, c) \cdot fvar(p_n) = 0 \end{aligned}$$

Potential equations

$$potential : C \rightarrow \mathcal{P}(\mathbb{E})$$

$$\begin{aligned} &potential(c) = \\ &\quad \text{let } \{p_1, \ldots, p_n\} = c \text{ in} \\ &\quad \{pvar(p_1){=}pvar(p_2), pvar(p_2){=}pvar(p_3), \ldots, \\ &\quad\ pvar(p_{n-1}){=}pvar(p_n)\} \end{aligned}$$

Summary and Conclusions

Summary and Conclusions

Some key take away points:

- Equation-Based Object-Oriented (EOO) Languages have reusable components because of acausal connections and equations.
- Modelica is the state-of-the-art EOO language, used for modeling of complex systems.
- OpenModelica is free Modelica tool that can be used for advanced physical modeling.

Thanks for listening!