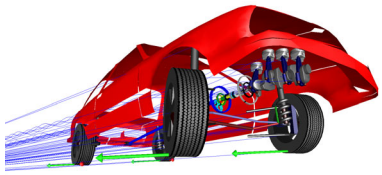
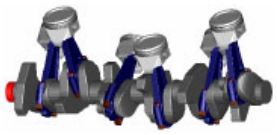
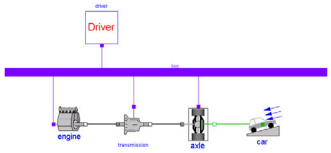
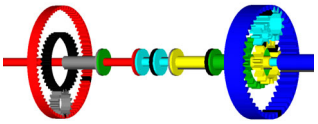
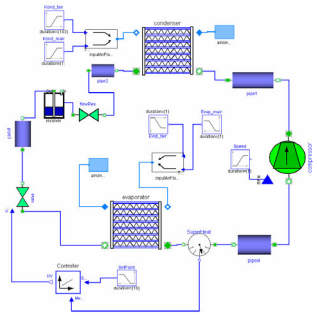
# Tutorial 1
# Modelica and Dymola for System Design
# Part 1 - Introduction

**Martin Otter and Hilding Elmqvist**
DLR Institute of Robotics and Mechatronics, and Dynasim AB
Modelica Conference, March 7-8, 2005



## Contents

# 1. Modelica and Dymola

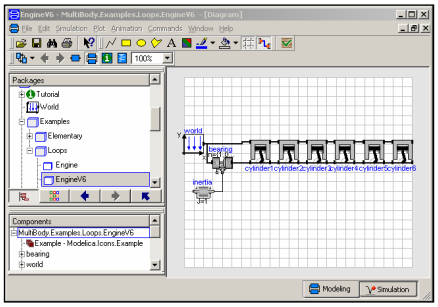
Goal: Modeling and Simulation of **physical** systems, consisting of components from **different engineering domains**

**Modelica**:
free textual format to specify
model + graphics + animation of devices

```
model IF97_ph
  "Steam properties as defined by IAPWS/IF97 standar

  extends Interfaces.PartialMedium(
    mediumName="WaterIF97",
    substanceNames=fill("", 0),
    MassFlowRate(quantity="MassFlowRate.WaterIF97"),
    p(stateSelect=if preferedMediumStates then State
          StateSelect.default),
    h(stateSelect=if preferedMediumStates then State
          StateSelect.default),
    final eta    =if eta_needed then IF97.dynamicVisc
    final lambda=if lambda_needed then IF97.thermal
    final sigma =if sigma_needed then IF97.surfaceTe
equation
  T = IF97.T_ph(p, h);
  d = IF97.rho_ph(p, h);
  u = h - p/d;
```

**Dymola**:
commercial Modelica modeling and simulation environment from Dynasim
(graphical editor/library browser,
translation to C, simulation, plotting,
animation)



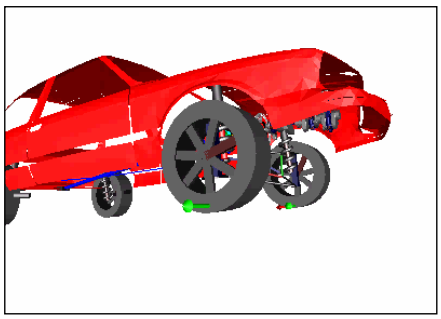
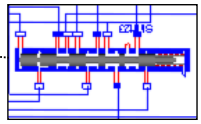
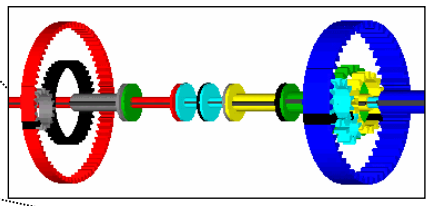
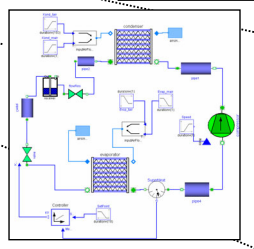
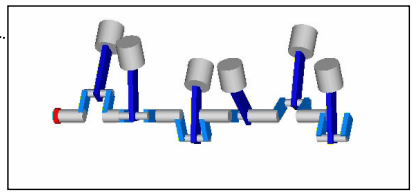
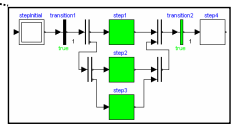
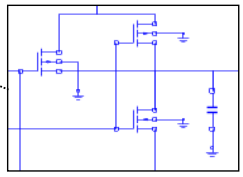



MODELICA http://www.Modelica.org

- **Modelica** is a **free modeling language** developed by the non-profit **Modelica Association** since 1996 in 42 design meetings à 3 days (about 4 m.y.)
  Chairman of Modelica Association: Martin Otter, DLR, since 2000.
- Modelica® is a registered trademark (Europe and U.S.A)
- It is based on **differential, algebraic** and **discrete** equations
- It is used in industrial applications since 2000.
  previous version: Modelica 2.1 (January 2004)
  New version : Modelica 2.2 (February 2005)
- **Modelica tools**: Dymola, MathModelica, OpenModelica (not yet useful)
- **Books**:
  - Introduction to Physical Modeling with Modelica.
    by Michael Tiller, Kluwer Academic Publisher, 2001
  - Principles of Object Oriented Modeling and Simulation with Modelica
    by Peter Fritzson, Wiley-IEEE press, 940 pages, 2003

Example: **detailed Modelica vehicle model**

- **Vehicle dynamics** (3-dim. mechanics)
- **Power train** (1-dim. mechanics)
- **Hydraulics**
- **Combustion**
- **Air-conditioning systems** (thermo-fluid systems)
- **Electrical/electronic systems**
- **Hierarchical State machines**
- **Control** (Input/output blocks, ...)
- ...



## Dynasim AB

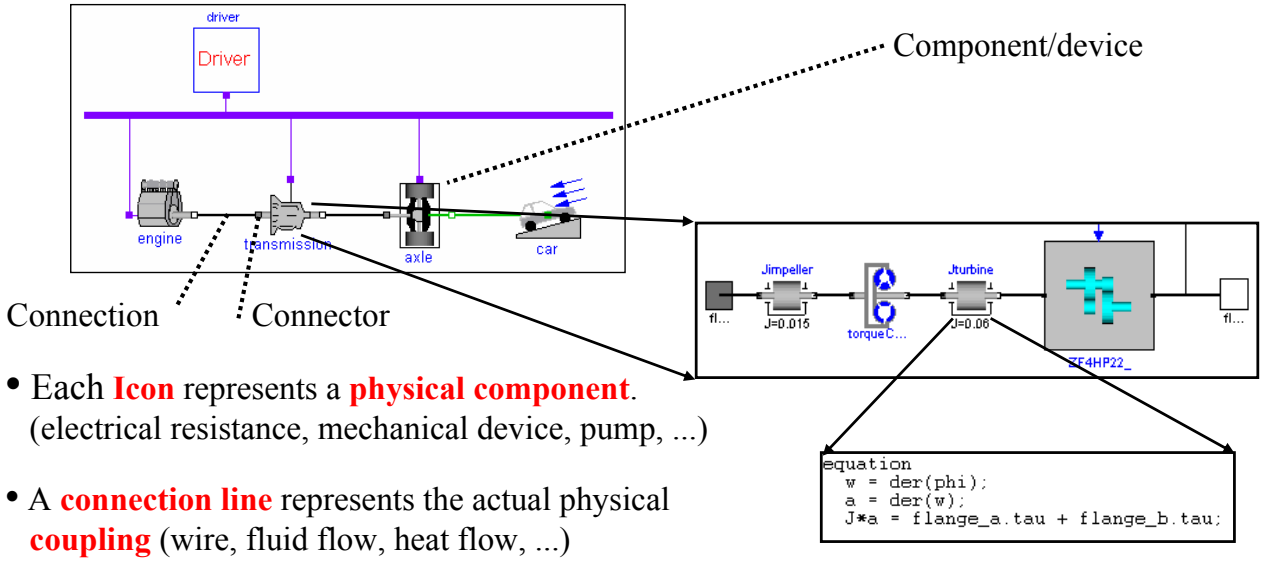
Ideon, Lund, Sweden

http://www.dynsim.de

- Founded 1992
- Privately owned by Hilding Elmqvist
- Product: **Dymola**
- Support of **Modelica** since **1999**
- Major customers:
  **Toyota, United Technologies, Ford, Aisin Seiki, BMW**
- About **1000 Dymola licenses** sold
- Distributors in
  Germany, Japan, U.S.A, Canada, U.K., Italy

# 2. Users View - Schematics

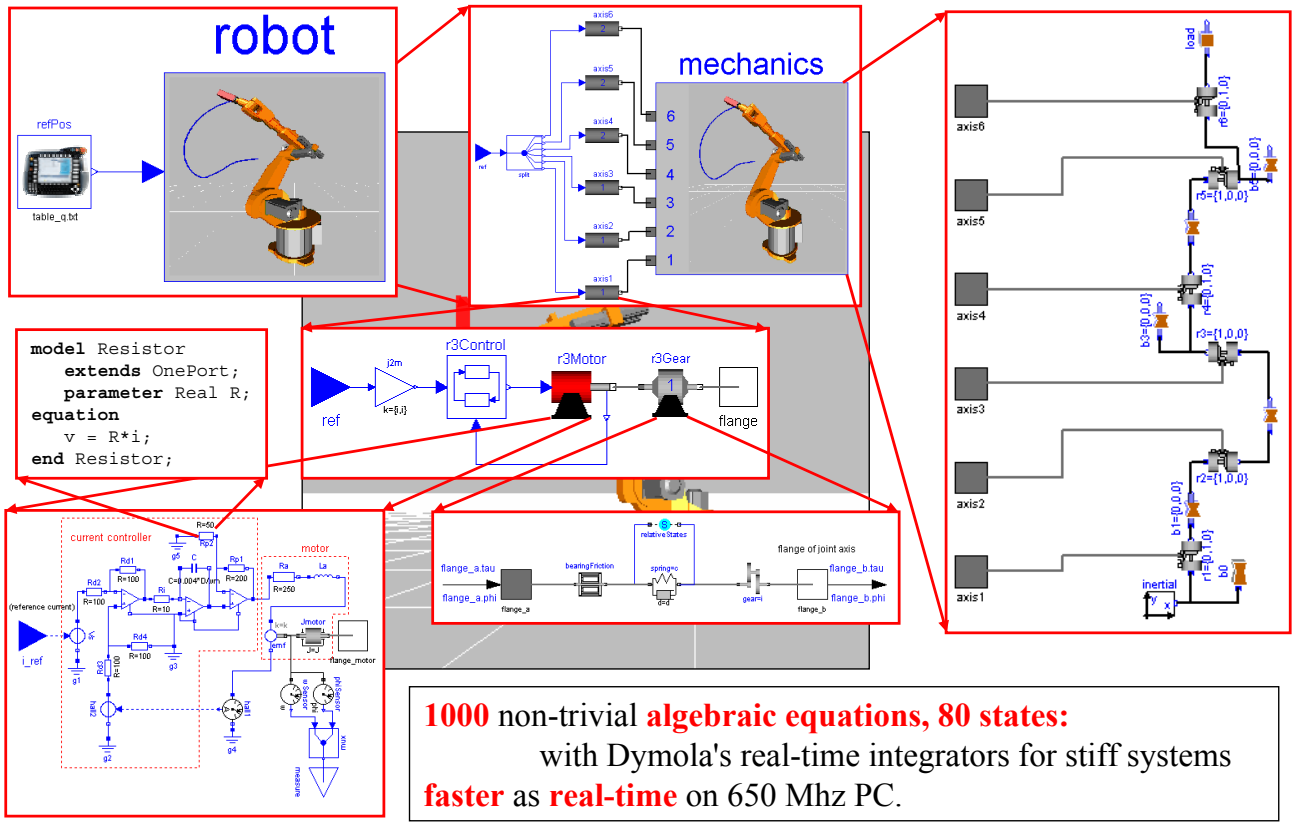
(screenshots from Dymola's graphical editor)




- Each **Icon** represents a **physical component**. (electrical resistance, mechanical device, pump, ...)
- A **connection line** represents the actual physical **coupling** (wire, fluid flow, heat flow, ...)
- A component consists of **connected** sub-components (= hierarchical structure) and/or is described by **equations**.
- By **symbolic** algorithms, the high level Modelica description is transformed into state space form (d**x**/dt = **f**(**x**, t))



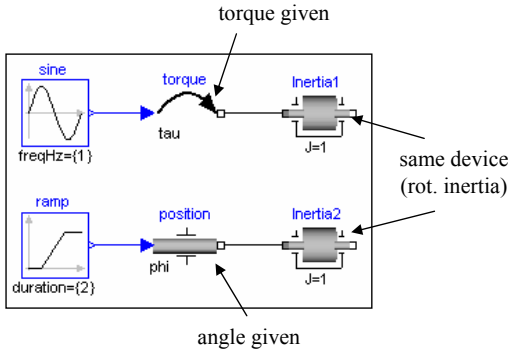
## Example: Industrial Robot (DLR, Dynasim, KUKA)



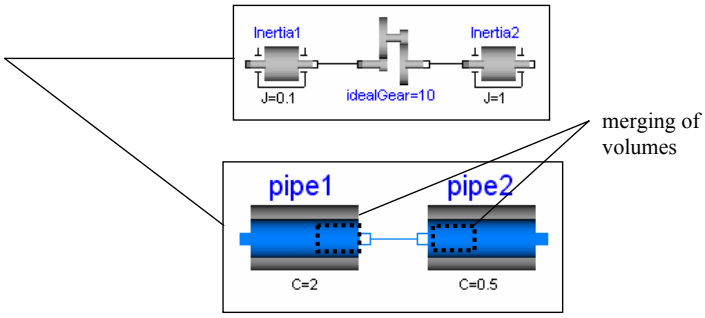

**1000** non-trivial **algebraic equations, 80 states:**
with Dymola's real-time integrators for stiff systems
**faster** as **real-time** on 650 Mhz PC.

# Design principles of Modelica

- Software components represent **physical devices** (computational causality is **not** fixed in model)




- Same **connection possibilities** as physical devices

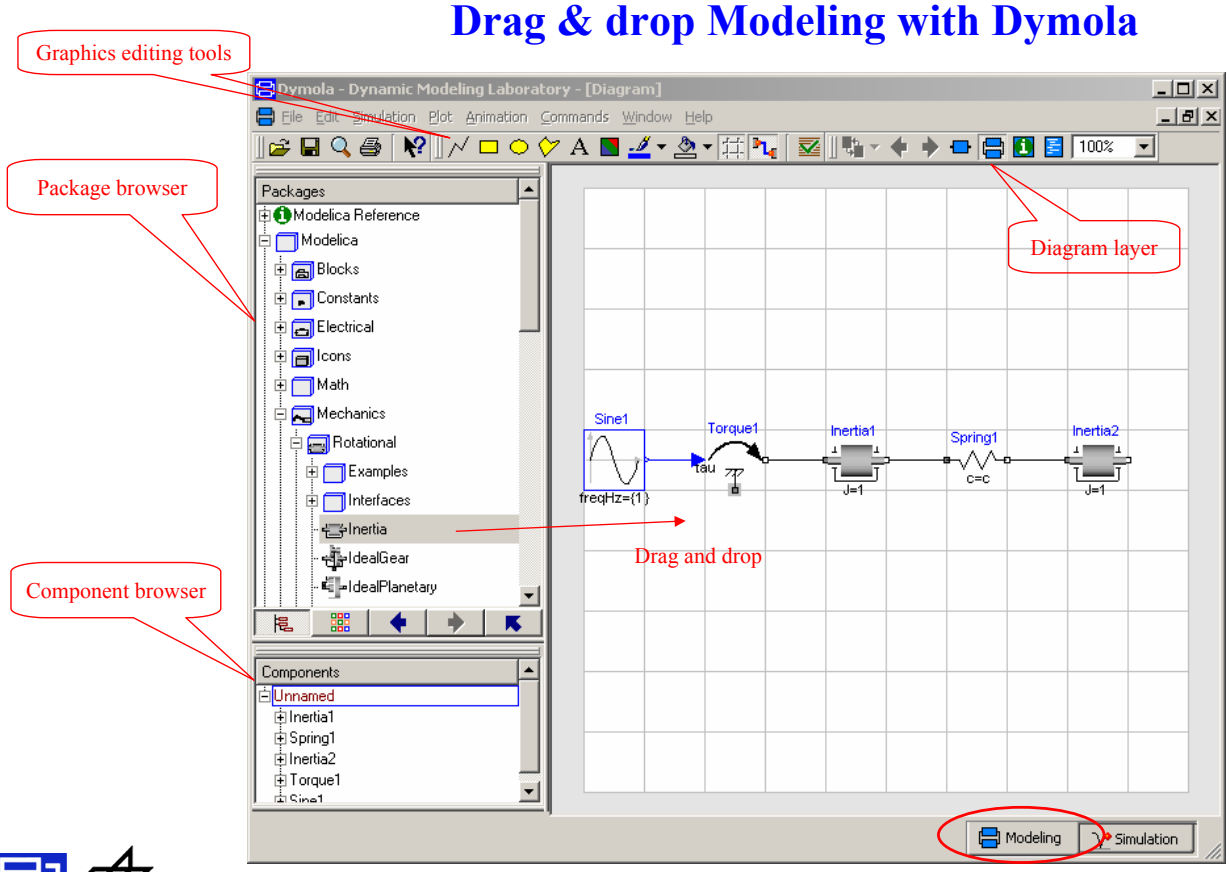



- Same or better computational **efficiency** as "standard" software

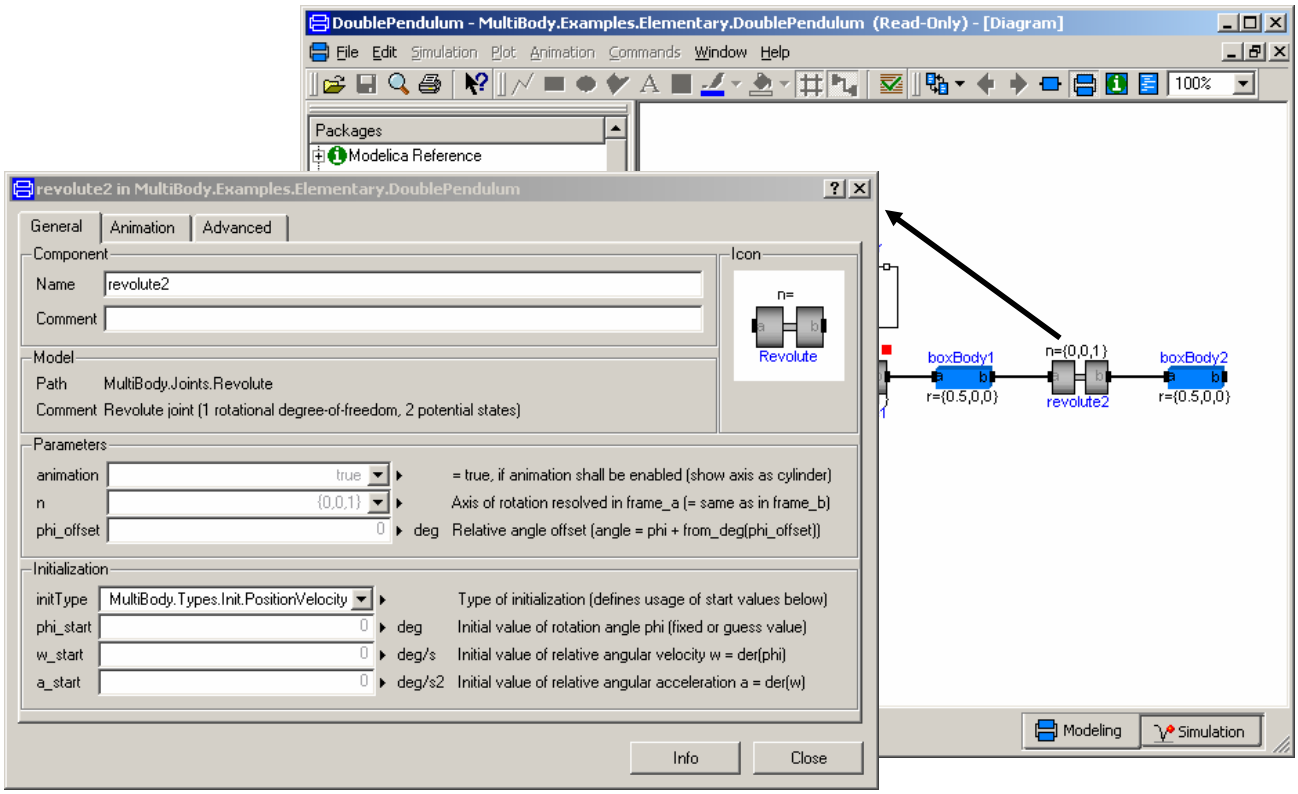
Only possible, due to specialized **symbolic** pre-processing (Modelica is designed such that known algorithms can be used)



# Drag & drop Modeling with Dymola

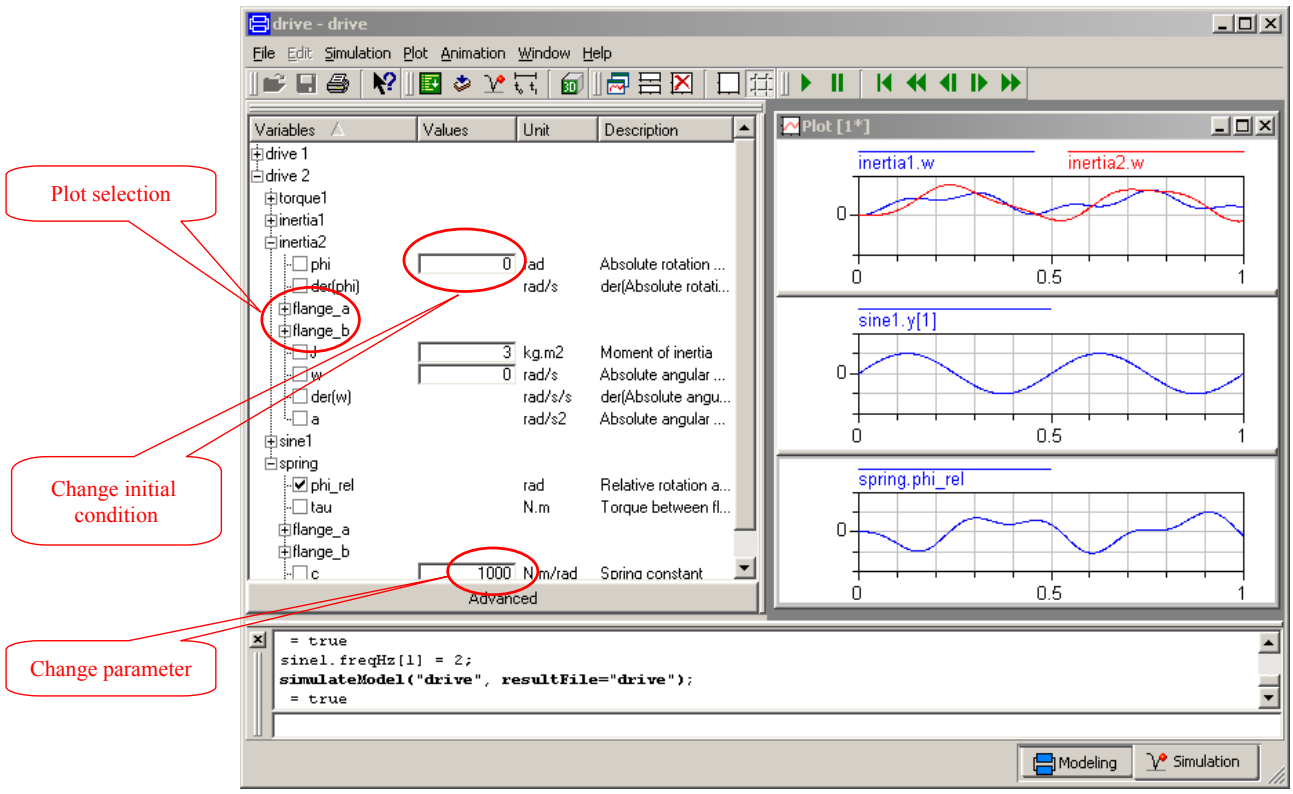


# **Component Data Definition** (double click on component)

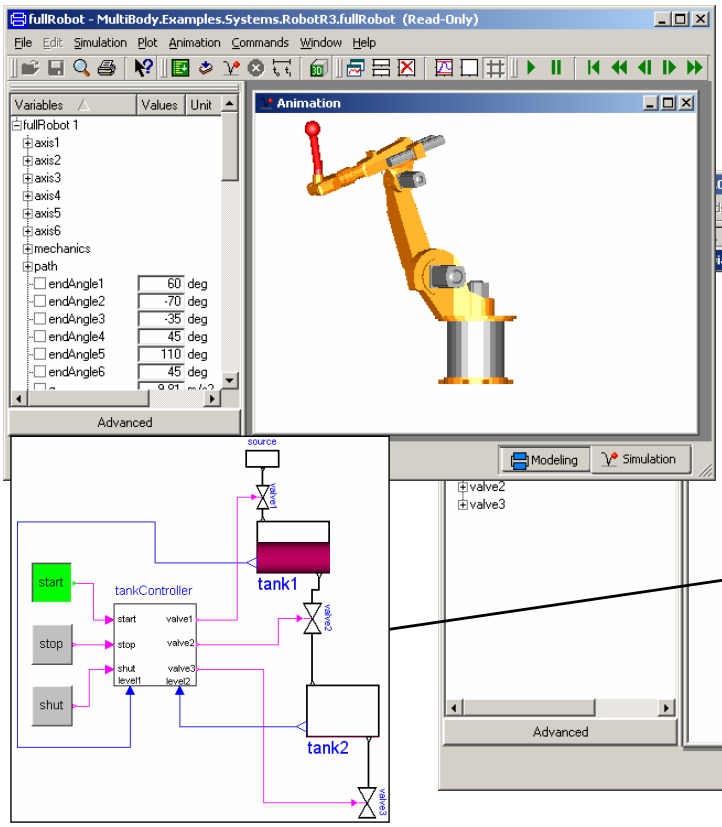




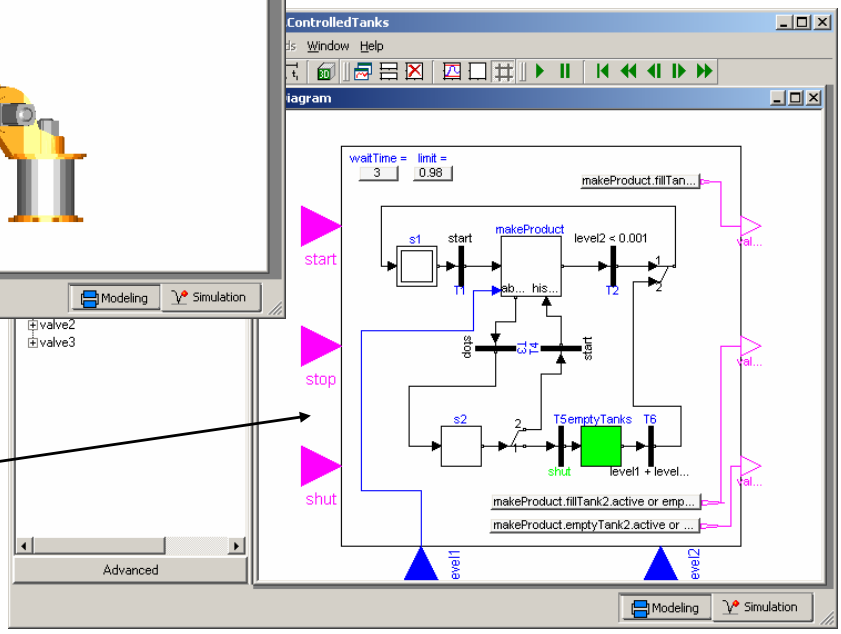

# **Translation** (Modelica → C → Object-code) + **Simulation** + **Plots**



# 3-dim. Animation



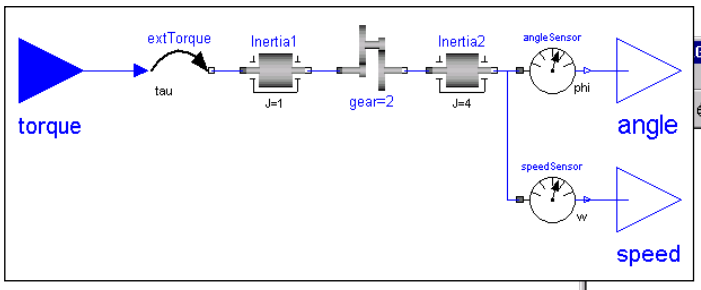

# Schematic Animation (Beta)





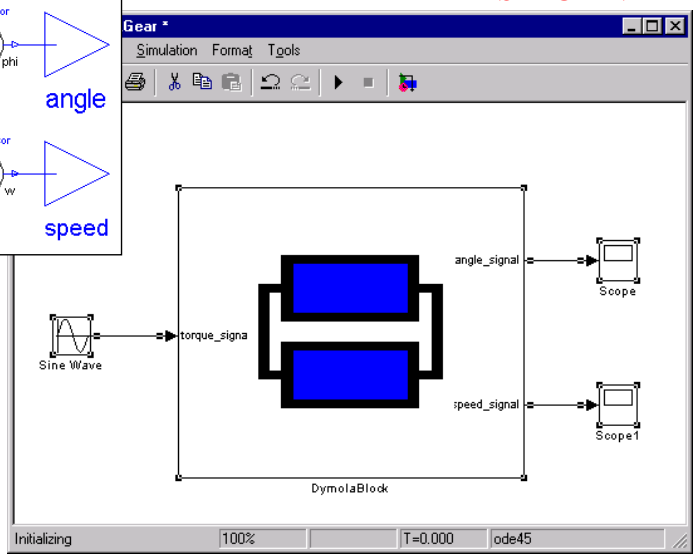

# Export of Modelica models to SIMULINK

Modelica models (with inputs, outputs) can be translated by **Dymola** to **Simulink** in the form of a C-Mex S-Function:

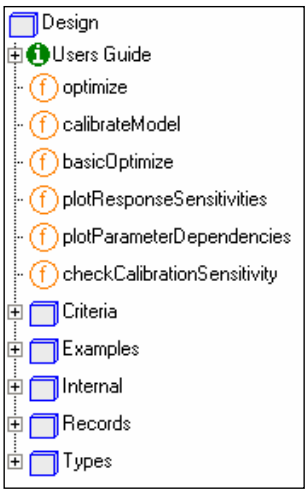



in Dymola

in SIMULINK



# Design of Modelica models (Dymola 6)




with **parameter optimization** that is based on

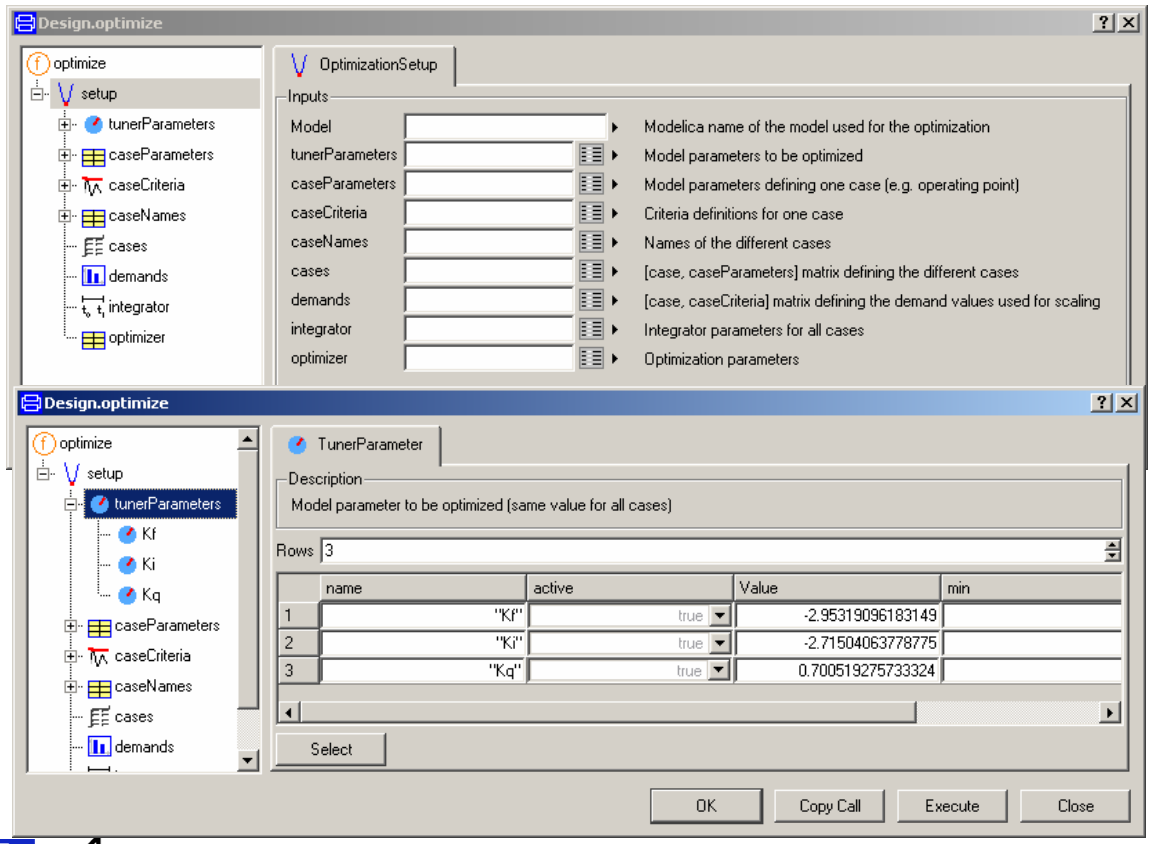
- **multiple criteria** (e.g. overshoot, rise time, settling time)
- **multiple simulation runs** for one iteration (e.g. different operating points)

Starting point: Modelica model where some **parameters** are not yet fixed and shall be **determined**.
Set of functions (expanded for final release), e.g.:

- **calibrateModel()**
  make simulation results close to measurements
- **optimize()**
  tune, e.g., controller parameters, that the controller works well in all operating points.
- **plotResponseSensitivities()**
  evaluate sensitivity of model against variations of parameters.



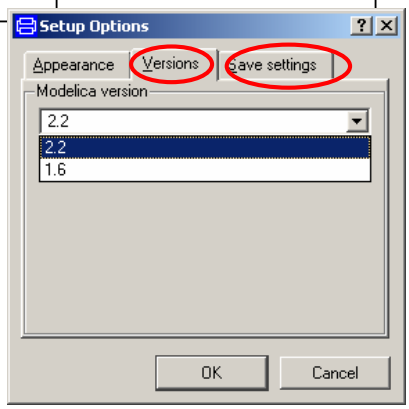
**Design.optimize()** minimize (maximum of citeria_i/demand_i)



# 3. Modelica Libraries

## Modelica Standard Library (free)

| | *number of generic models* | *number of functions* | *number of packages* |
|---|---|---|---|
| Modelica **1.6** (June 2004) | 290 | 40 | 50 |
| Modelica **2.1** (Nov. 2004) | 580 | 200 | 130 |
| Modelica **2.2** (March 2005) | 640 | 470 | 1450 (Media) |




**Size** is **doubled** from 1.6 to 2.1.

The 2.2 library is delivered with the newest Dymola version 5.3b++ used in the exercises.

The user has the choice between 1.6, 2.1 and 2.2 since not backward compatible for the first time (1.6 -> 2.1)

Conversion is automatic, but not possible to go back.



**Electrical**

| | |
|---|---|
| Analog | R, L, C, Diode, PMOS, NMOS, ... |
| Digital | 2-, 3-, 4-, 9-valued VHDL logic elements |
| Machines | asynchronous-, synchronous-, DC-motors/generators |
| MultiPhase | R, L, C, Diode etc. for multiple phases |

**Mechanics**

| | |
|---|---|
| MultiBody | 3D-mechanics (body, joint, force, sensor, ...) |
| Rotational | 1D-rotational (inertia, clutch, brake, bearingFriction, ..) |
| Translational | 1D-translational (mass, stop, spring, ...) |

**Thermal**

| | |
|---|---|
| HeatTransfer | Lumped heat transfer (heat capacitor, thermal conductor, ...) |
| FluidHeatFlow | 1-dim., incompressible thermo-fluid flow, e.g., for cooling. |

**Media** 1250 media (single/multiple substances, one/multiple phases)

**Blocks** (input/output blocks)
- Continuous
- Discrete
- Logical
- Nonlinear
- Routing
- Sources
- Tables

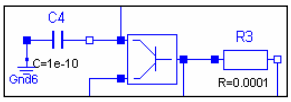
**StateGraph** (hierarchical state machines)

**Constants, Icons, SIunits**

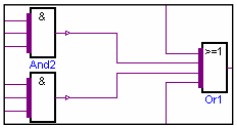
**Math**
- Matrices (eigen values, LU, QR, ...)

**Utilities**
- Files (copy, move, rename files, ...)
- Streams (print, readFile, readLine, ...)
- Strings (find, replace, sort, scan, ...)
- System (get/set work directory, ...)

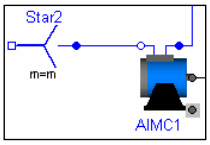
Modelica.Electrical.Analog



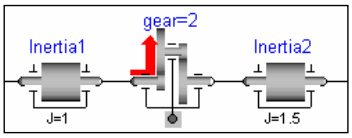

Modelica.Electrical.Digital



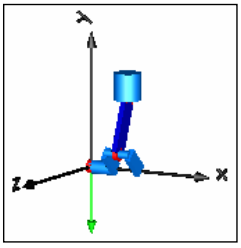

Modelica.Electrical.Machines

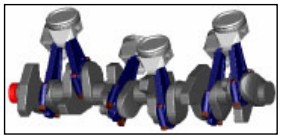



Modelica.Mechanics.Rotational

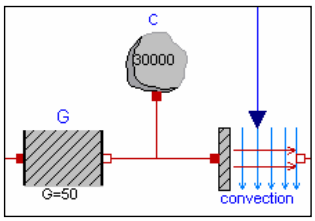



Modelica.Mechanics.MultiBody

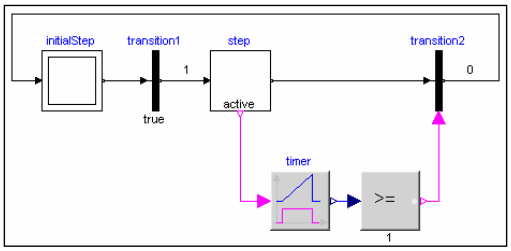


Modelica.Thermal.HeatTransfer



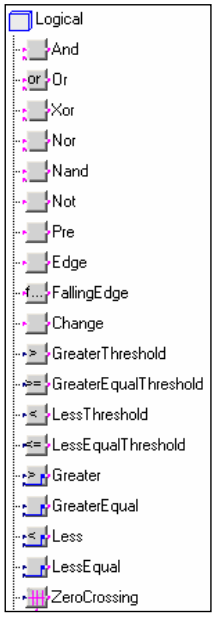

Modelica.StateGraph, Modelica.Blocks.Logical



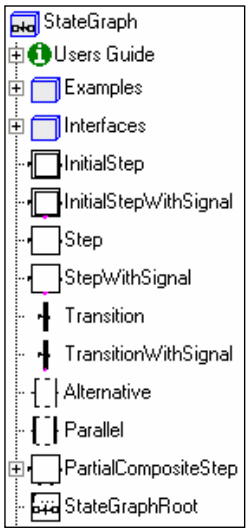



Sublibraries of Modelica for **sampled data** and **discrete event systems**

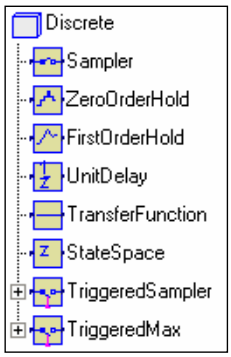
**Modelica.Blocks.Logical**
(logical networks)

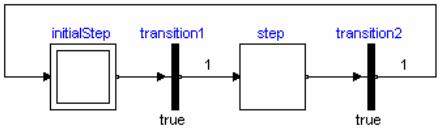



**Modelica.StateGraph**
(discrete event systems)

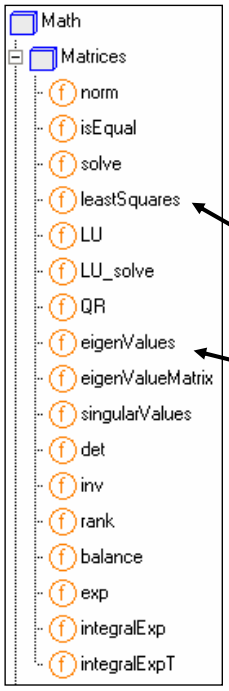



**Modelica.Blocks.Discrete**
(sampled data systems)



## Modelica.Math.Matrices




Dymola and Modelica have been mainly used for simulation of non-linear systems. The Modelica Association develops currently also libraries for **scripting**, as needed for pre- and postprocessing of simulations, e.g., for **linear algebra**. This reduces the need to switch to Matlab, since direct and convenient usage in Modelica/Dymola. Furthermore, much easier to get a stable/robust interface due to the strongly typed language:

"least squares" solution of linear system

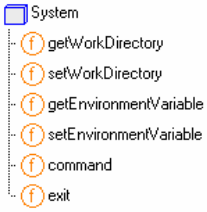
Eigenvalues and optional eigenvectors

```
ew = eigenValues(A);
(ew, ev) = eigenValues(A);
```



## Modelica.Utilities




very **convenient** for **String** handling (much better as in C, Fortran, Matlab).

e.g. Streams.**print**("Output on terminal")
Streams.**print**("Output on file", "test.txt")
Files.**remove**("test.txt")

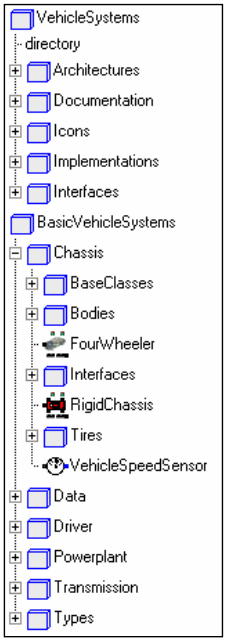
Modelica.Utilities.Examples.readRealParameterModel:

```
parameter String file = classDirectory() + "data/test.txt"
parameter SI.Inertia  J  = readRealParameter(file, "J");
parameter SI.Velocity v0 = readRealParameter(file, "v0");
```

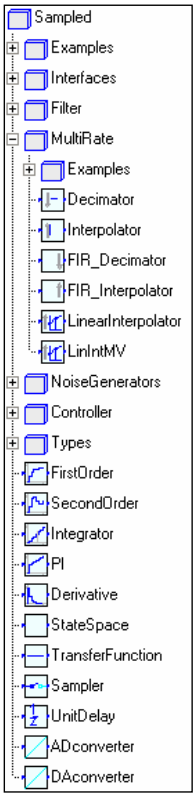
```
test.txt:
   J  = 2.0 // comment
   v0 = sin(pi/3);
```

# Free Libraries under development (Beta available)

## VehicleSystems




Complete vehicle models (driver, transmission, battery, 1D models, simple 3D chassis models)

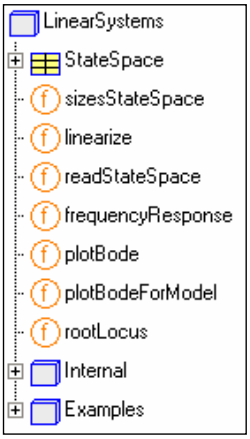



## Sampled

- continuous parameterization
- switch between continuous and discrete approximations

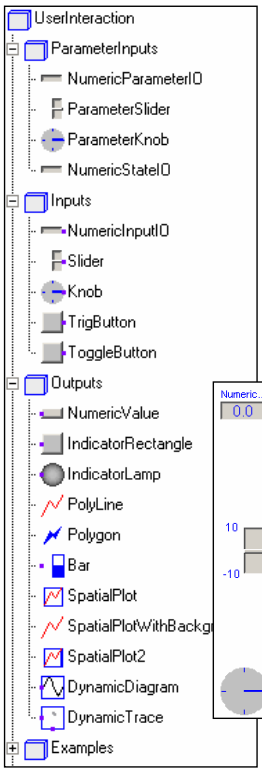


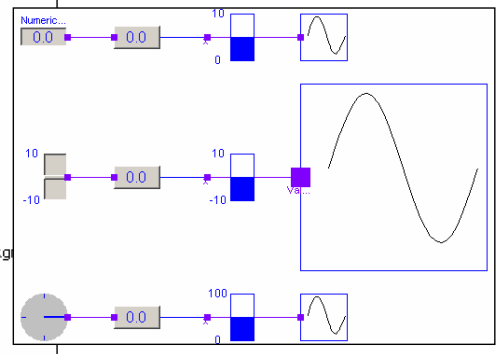
## LinearSystems




operations on linear systems,
e.g. model linearization,
eigen value plot,
Bode plot

## UserInteraction

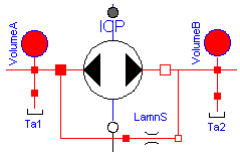


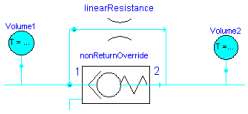

Components for dynamic and interactive interaction with the model

# Commercial Libraries

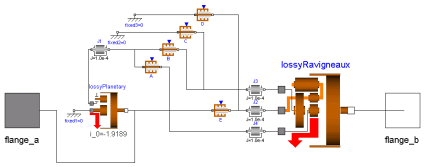
**HyLib** (hydraulic systems)



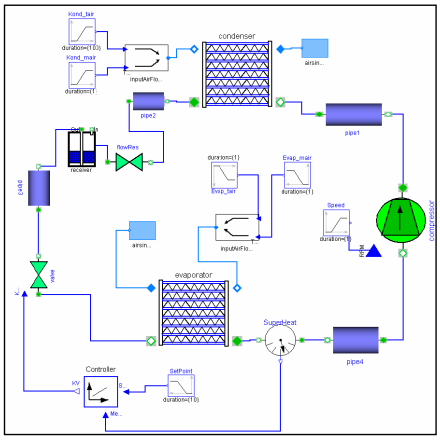

**PneuLib** (pneumatic systems)




**PowerTrain**




**AirConditioning**




selected as standard tool for modeling of vehicle air conditioning systems by the 6 German car manufacturers

**Import/Export**

- Simulink → Modelica (Simelica)
- Modelica → Simulink (Dymola option)



# 4. Modelica Basics

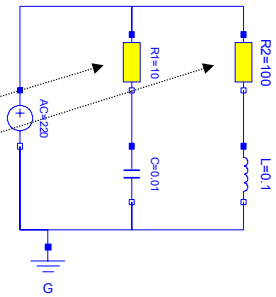
## Equation based modeling

The **general form** in Modelica:

*expression* = *expression*

e.g., J***der**(x) = t1 + t2 (Eulers law)

R*i = v (Ohm's law)




"**Unknown**" depends on model **connection structure**:

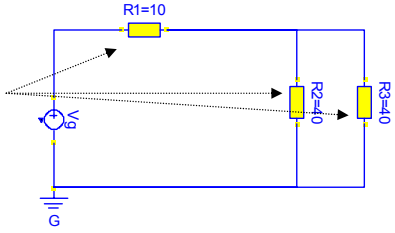
i := u/R

u := R*i

R := u/i

or **system of algebraic equations**

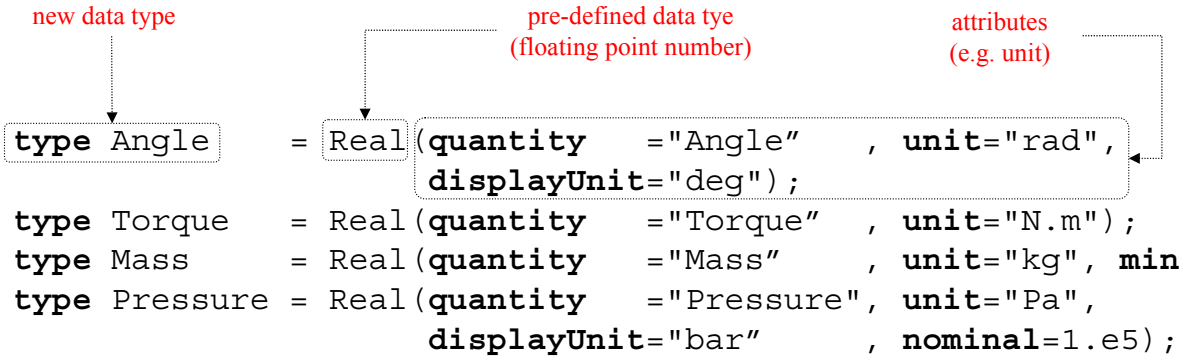
ε := R*i - u (residue)



# Variables and Attributes

**pre-defined data types**

Real, Integer, Boolean, String




```
type Angle    = Real(quantity    ="Angle"    , unit="rad",
                     displayUnit="deg");
type Torque   = Real(quantity    ="Torque"   , unit="N.m");
type Mass     = Real(quantity    ="Mass"     , unit="kg", min=0);
type Pressure = Real(quantity    ="Pressure", unit="Pa",
                     displayUnit="bar"      , nominal=1.e5);
```

**Attributes:**

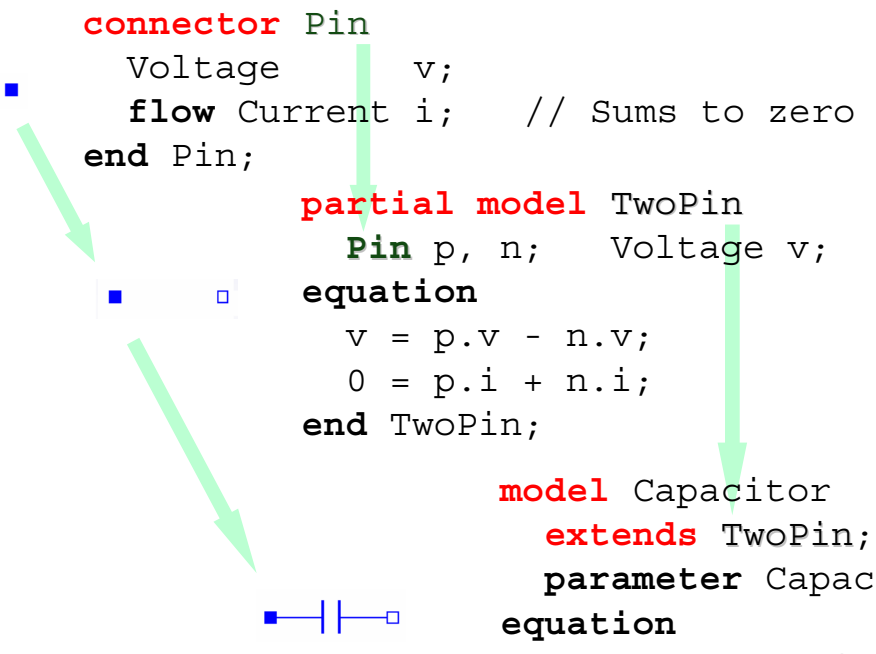
quantity, unit, displayUnit, min, max, start, fixed, nominal, stateSelect

Library **Modelica.SIunits:**

450 pre-defined types according to the ISO unit standard



# Modelica Component Models

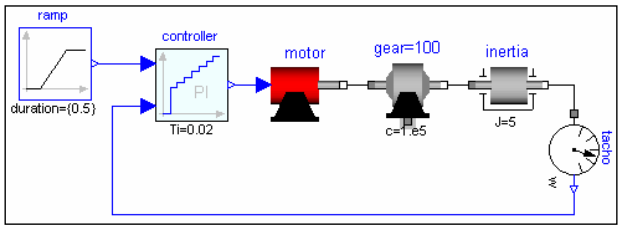


```
connector Pin
  Voltage      v;
  flow Current i;   // Sums to zero
end Pin;

partial model TwoPin
  Pin p, n;   Voltage v;
equation
  v = p.v - n.v;
  0 = p.i + n.i;
end TwoPin;

model Capacitor
  extends TwoPin;
  parameter Capacitance C;
equation
  C*der(v) = p.i;
end Capacitor;
```

# Hierarchical Modelica Models




```
model MotorDrive
  PI          controller;
  Ramp        ramp;
  Motor       motor;
  Gearbox     gear(ratio = 100);
  Inertia     inertia(J = 10);
  SpeedSensor tacho;
equation
  connect(controller.outPort, motor.inPort);
  connect(motor.flange      , gearbox.flange_a);
  connect(gearbox.flange_b  , inertia.flange_a);
  connect(inertia.flange_b  , tacho.flange);
  connect(tacho.outPort     , controller.inPort2);
  connect(ramp.outPort      , controller.inPort1);
end MotorDrive;
```

Class name

Instance name

Modifier

Connection

Connector

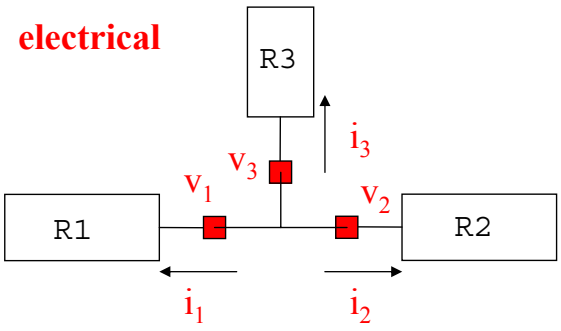


# Modelica Interface Definitions

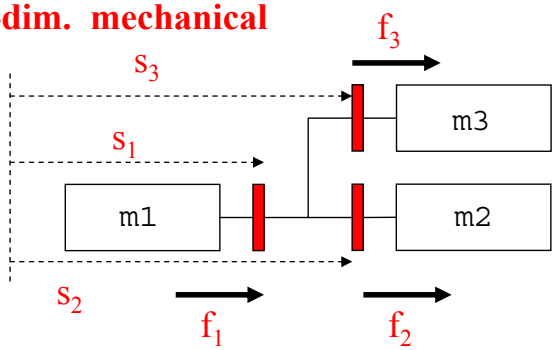
```
connector Pin
  Voltage      v;
  flow Current i;
end Pin;
```

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

**electrical**




**1-dim. mechanical**




```
connect(R1.p, R2.p);
connect(R1.p, R3.p);
```

$v_1 = v_2$

$v_1 = v_3$

$i_1 + i_2 + i_3 = 0$

($i_1$ = R1.p.i, $v_1$ = R1.p.v, etc.)

```
connect(m1.flange_a, m2.flange_a);
connect(m1.flange_a, m3.flange_a);
```

$s_1 = s_2$

$s_1 = s_3$

$f_1 + f_2 + f_3 = 0$

# Elementary Interfaces provided in Libraries

| Typ | Potential-V. | Fluss-Variable | Connector-Name (Modelica library) | Icon |
|---|---|---|---|---|
| **electrical** | electrical potential | current | Modelica.Electrical.Analog.Interfaces.**PositivePin** | PositivePin<br>NegativePin |
| **translational** | distance | force (scalar) | Modelica.Mechanics.Translational.Interfaces.**Flange_a** | Flange_a<br>Flange_b |
| **rotational** | angle | torque (scalar) | Modelica.Mechanics.Rotational.Interfaces.**Flange_a** | Flange_a<br>Flange_b |
| **heat transfer** | temperature | heat flow rate | Modelica.Thermal.HeatTransfer.Interfaces.**HeatPort_a** | HeatPort_a<br>HeatPort_b |
| **hydraulic** | pressure | volume flow rate | HyLibLight.Interfaces.**Port_A** | Port_A<br>Port_B |
| **pneumatic** | pressure | mass flow rate | PneuLibLight.Interfaces.**Port_1** | Port_1<br>Port_2 |
| **mechanical** | position vector, transformation matrix | cut-force (vector)<br>cut-torque (vector) | Modelica.Mechanics.MultiBody.Interfaces.**Frame_a** | Frame_a<br>Frame_b |
| **thermo-fluid** | pressure, specific enthalpy, mass fractions | mass flow rate, enthalpy flow rate, mass flow rate of substances | Modelica_Fluid.Interfaces.**FluidPort**_a | FluidPort_a<br>FluidPort_b |
| **signal** | Real, Boolean, Integer Vektor | | Modelica.Blocks.Interfaces.**RealInput** /Output<br>Modelica.Blocks.Interfaces.**BooleanInput**/Output<br>Modelica.Blocks.Interfaces.**IntegerInput** /Output | u<br>y |

(hydraulic : incompressible pipe flow
pneumatic : compressible pipe flow
thermo-fluid: incompressible/compressible multi-substances/phases pipe flow with heat transport)



# Modelica Libraries (package)

Modelica **libraries** are **hierarically** structured and are **mapped** on the **identical hierarchy** of the **file system**. This allows to find referenced Modelica classes automatically in the file system:

```
\Modelica
  \Blocks
  \Electrical
     \Analog
        package.mo
        Basic.mo
        Semiconductors.mo
        Sources.mo
           ...
  \Mechanics
     package.mo
     Rotational.mo
     Translational.mo
  Constants.mo
  Icons.mo
  Math.mo
  SIunits.mo
    ...
```

```
package Basic
   model Resistor
      ...
   end Resistor;

   model Capacitor
      ...
   end Capacitor;
   ...
end Basic;
```

Library

File: ...\Modelica\Electrical\Analog\Basic.mo

Library: Modelica.Electrical.Analog.Basic

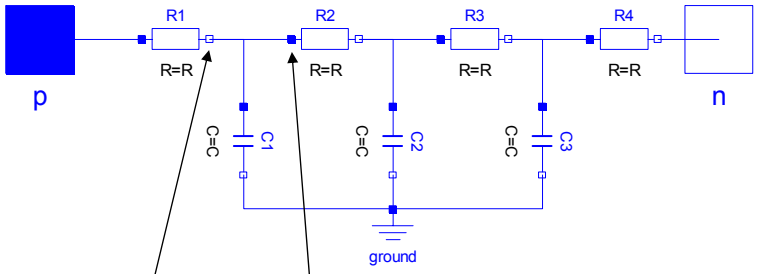
# Arrays and Component Arrays

**Arrays** are similar as in **Matlab**:

```
parameter Real A[:,:] = [11, 12, 13;
                         21, 22, 23];
```

But: **Arrays** can be defined from every model class.

**Example: electrical line with losses**




```
import Basic = Modelica.Electrical.Analog.Basic;
parameter Integer n=4;
Basic.Resistor Rvec[n]      // 4 resistors
equation
   for i in 1:n-1 loop
      connect(Rvec[i].n, Rvec[i+1].p);  // connector definition
   end for;
```

Allows convenient **discretization** of simple **partial differential equations**