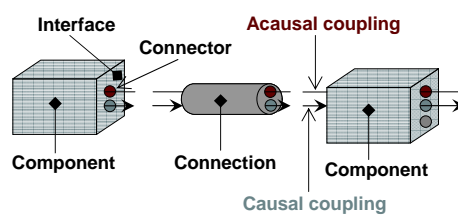


# Components, Connectors and Connections

## Software Component Model



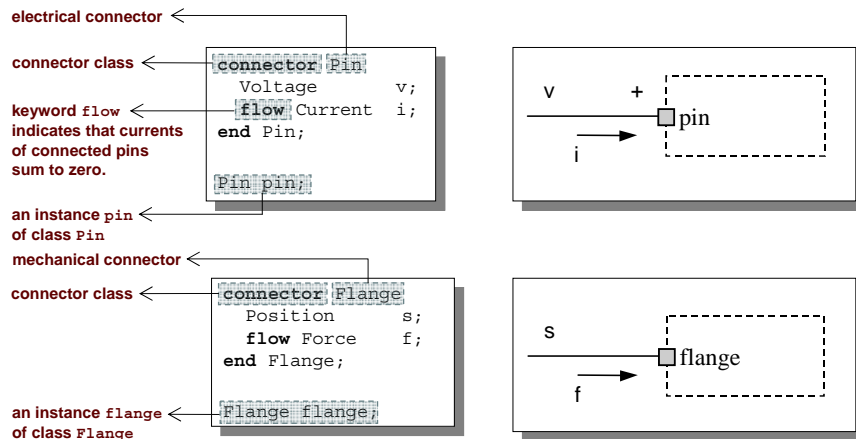
A component class should be defined *independently of the environment*, very essential for *reusability*

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

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

## Connectors and Connector Classes

Connectors are instances of *connector classes*



## The `flow` prefix

Two kinds of variables in connectors:

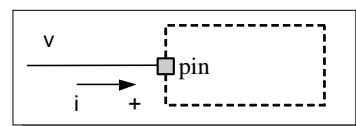
- *Non-flow variables* *potential* or energy level
- *Flow variables* represent some kind of flow

Coupling

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

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

positive flow direction:



## Physical Connector Classes Based on Energy Flow

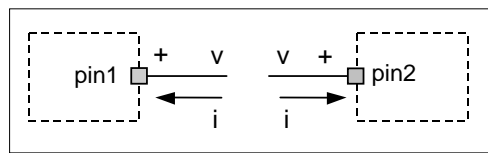
Domain Type	Potential	Flow	Carrier	Modelica Library
Electrical	Voltage	Current	Charge	Electrical. Analog
Translational	Position	Force	Linear momentum	Mechanical. Translational
Rotational	Angle	Torque	Angular momentum	Mechanical. Rotational
Magnetic	Magnetic potential	Magnetic flux rate	Magnetic flux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

## connect-equations

Connections between connectors are realized as *equations* in Modelica

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

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



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

Corresponds to

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

## Connection Equations

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

Corresponds to

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

Multiple connections are possible:

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

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

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

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

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

## Acausal, Causal, and Composite Connections

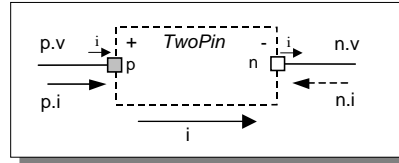
Two *basic* and one *composite* kind of connection in Modelica

- *Acausal connections*
- *Causal connections*, also called *signal connections*
- *Composite connections*, also called structured connections, composed of basic or composite connections

```
connector class ← connector OutPort  
fixed causality ← output Real signal;  
end OutPort
```

## Common Component Structure

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



partial class  
(cannot be  
instantiated)

positive pin

negative pin

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

connector Pin

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

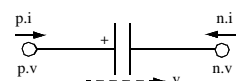
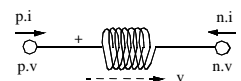
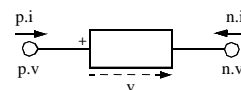
electrical connector class

## Electrical Components

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

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

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

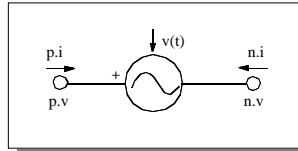


## Electrical Components cont'

```

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

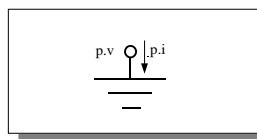
```



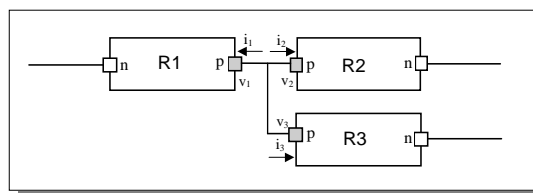
```

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

```



## Resistor Circuit



```

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

```

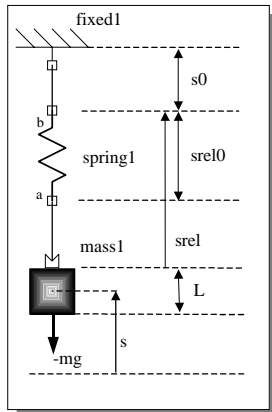
Corresponds to

```

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

```

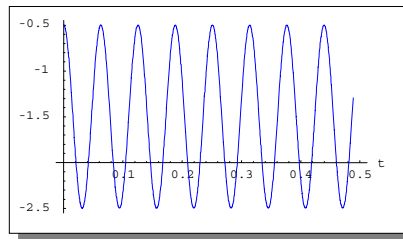
## An Oscillating Mass Connected to a Spring



```

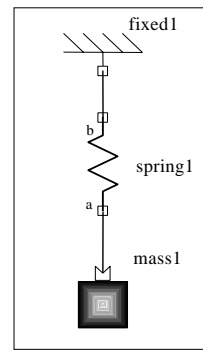
model Oscillator
  Mass    mass1(L=1, s(start=-0.5));
  Spring  spring1(srel0=2, c=10000);
  Fixed   fixed1(s0=1.0);
equation
  connect(spring1.flange_b, fixed1.flange_b);
  connect(mass1.flange_b, spring1.flange_a);
end Oscillator;

```



## Extra Exercise

- Locate the Oscillator model in DrModelica using OMNotebook!
- Simulate and plot the example. Do a slight change in the model e.g. different elasticity  $c$ , re-simulate and re-plot.
- Draw the Oscillator model using the graphic connection editor e.g. using the library Modelica.Mechanical.Translational
- Including components SlidingMass, Force, Blocks.Sources.Constant
- Simulate and plot!



## Signal Based Connector Classes

```

connector InPort "Connector with input signals of type Real"
  parameter Integer n=1 "Dimension of signal vector";
  input Real signal[n] "Real input signals";
end InPort;

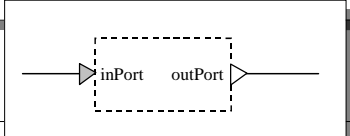
connector OutPort "Connector with output signals of type Real"
  parameter Integer n=1 "Dimension of signal vector";
  output Real signal[n] "Real output signals";
end OutPort;

partial block MISO
  "Multiple Input Single Output continuous control block"
  parameter Integer nin=1 "Number of inputs";
  InPort inPort(n=nin) "Connector of Real input signals";
  OutPort outPort(n=1) "Connector of Real output signal";
protected
  Real u[:] = inPort.signal "Input signals";
  Real y = outPort.signal[1] "Output signal";
end MISO; // From Modelica.Blocks.Interfaces
  
```

fixed causality ←

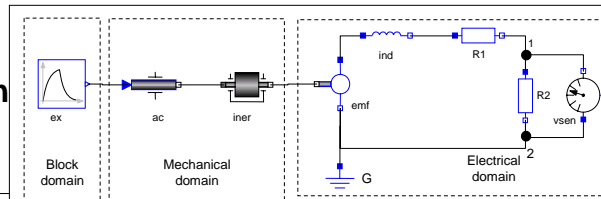
fixed causality ←

multiple input single output block ←



## Connecting Components from Multiple Domains

- Block domain
- Mechanical domain
- Electrical domain



```

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



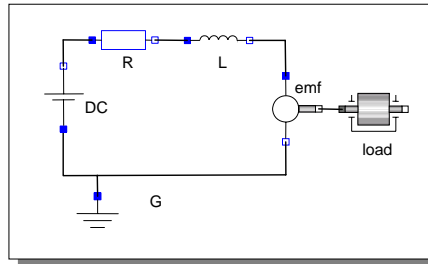
## Simple Modelica DCMotor Model Multi-Domain (Electro-Mechanical)

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

```

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

```



## Corresponding DCMotor Model Equations

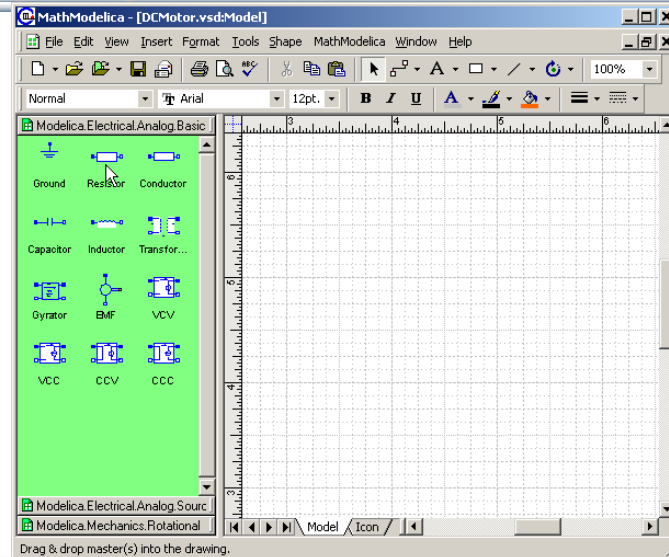
The following equations are automatically derived from the Modelica model:

0 == DC.p.i + R.n.i	EM.u == EM.p.v - EM.n.v	R.u == R.p.v - R.n.v
DC.p.v == R.n.v	0 == EM.p.i + EM.n.i	0 == R.p.i + R.n.i
	EM.i == EM.p.i	R.i == R.p.i
0 == R.p.i + L.n.i	EM.u == EM.k * EM.ω	R.u == R.R * R.i
R.p.v == L.n.v	EM.i == EM.M / EM.k	
	EM.J * EM.ω == EM.M - EM.b * EM.ω	L.u == L.p.v - L.n.v
0 == L.p.i + EM.n.i		0 == L.p.i + L.n.i
L.p.v == EM.n.v	DC.u == DC.p.v - DC.n.v	L.i == L.p.i
	0 == DC.p.i + DC.n.i	L.u == L.L * L.i'
0 == EM.p.i + DC.n.i	DC.i == DC.p.i	
EM.p.v == DC.n.v	DC.u == DC.Amp * Sin[2 * π * DC.f * t]	
0 == DC.n.i + G.p.i		
DC.n.v == G.p.v	(load component not included)	

Automatic transformation to ODE or DAE for simulation:

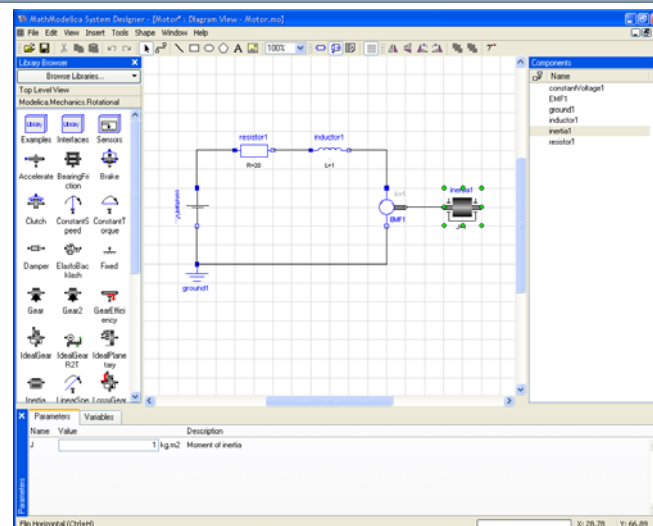
$$\frac{dx}{dt} == f[x, u, t] \quad g\left[\frac{dx}{dt}, x, u, t\right] == 0$$

## Graphical Modeling Using Drag and Drop Composition



Courtesy  
MathCore  
Engineering AB

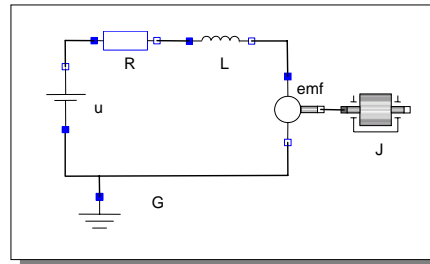
## Completed DCMotor using Graphical Composition



Courtesy MathCore  
Engineering AB

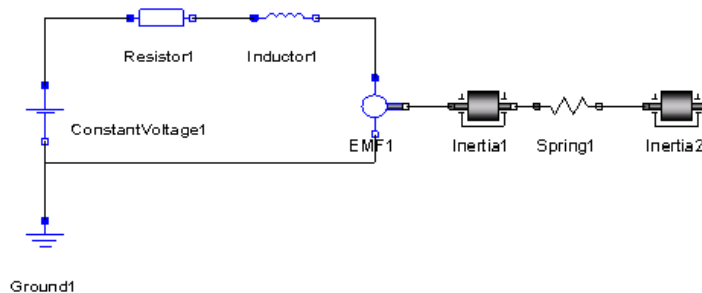
## Graphical Exercise 3.1

- Open `Exercise02-graphical-modeling.onb` and the corresponding `.pdf`
- Draw the `DCMotor` model using the graphic connection editor using models from the following Modelica libraries:  
`Mechanics.Rotational`,  
`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.



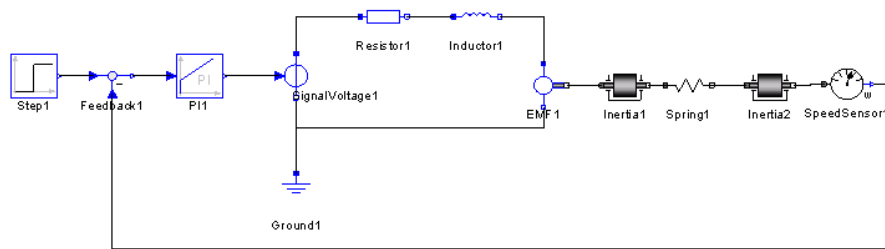
## Exercise 3.2

- 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 simForge.

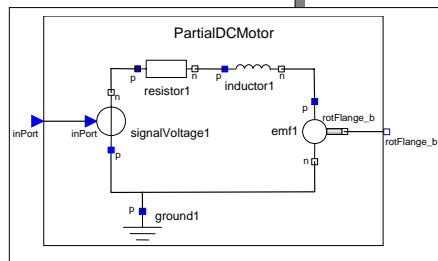


## Hierarchically Structured Components

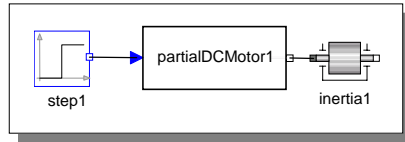
An *inside connector* is a connector belonging to an *internal component* of a structured component class.

An *outside connector* is a connector that is part of the *external interface* of a structured component class, is declared directly within that class

```
partial model PartialDCMotor
  InPort          inPort; // Outside signal connector
  RotFlange_b    rotFlange_b; // Outside rotational flange connector
  Inductor       inductor1;
  Resistor       resistor1;
  Ground         ground1;
  EMF            emf1;
  SignalVoltage  signalVoltage1;
equation
  connect(inPort, signalVoltage1.inPort);
  connect(signalVoltage1.n, resistor1.p);
  connect(resistor1.n, inductor1.p);
  connect(signalVoltage1.p, ground1.p);
  connect(ground1.p, emf1.n);
  connect(inductor1.n, emf1.p);
  connect(emf1.rotFlange_b, rotFlange_b);
end PartialDCMotor;
```



## Hierarchically Structured Components cont'



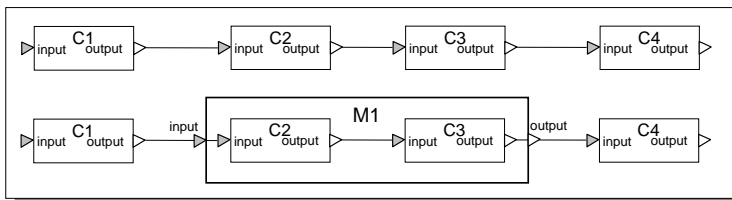
```

model DCMotorCircuit2
  Step      step1;
  PartialDCMotor partialDCMotor1;
  Inertia    inertia1;
equation
  connect(step1.outPort, partialDCMotor1.inPort);
  connect(partialDCMotor1.rotFlange_b, inertia1.rotFlange_a);
end DCMotorCircuit2;

```

## Connection Restrictions

- Two *acausal* connectors can be connected to each other
- An input connector can be connected to an output connector or vice versa
- An input or output connector can be connected to an *acausal* connector, i.e. a connector without input/output prefixes
- An *outside* input connector behaves approximately like an output connector internally
- An *outside* output connector behaves approximately like an input connector internally



## Connector Restrictions cont'

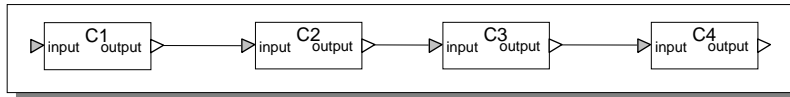
```
connector RealInput
  input Real signal;
end RealInput;
```

```
connector RealOutput
  output Real signal;
end RealOutput;
```

```
class C
  >RealInput u; // input connector
  >RealOutput y; // output connector
end C;
```

```
class CInst
  C C1, C2, C3, C4; // Instances of C
equation
  connect(C1.outPort, C2.inPort);
  connect(C2.outPort, C3.inPort);
  connect(C3.outPort, C4.inPort);
end CInst;
```

A circuit consisting of four connected components C1, C2, C3, and C4 which are instances of the class C

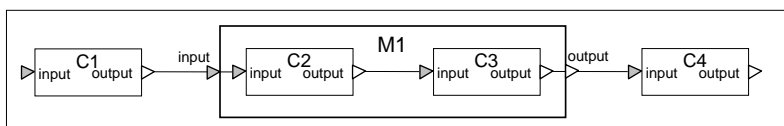


## Connector Restrictions cont'

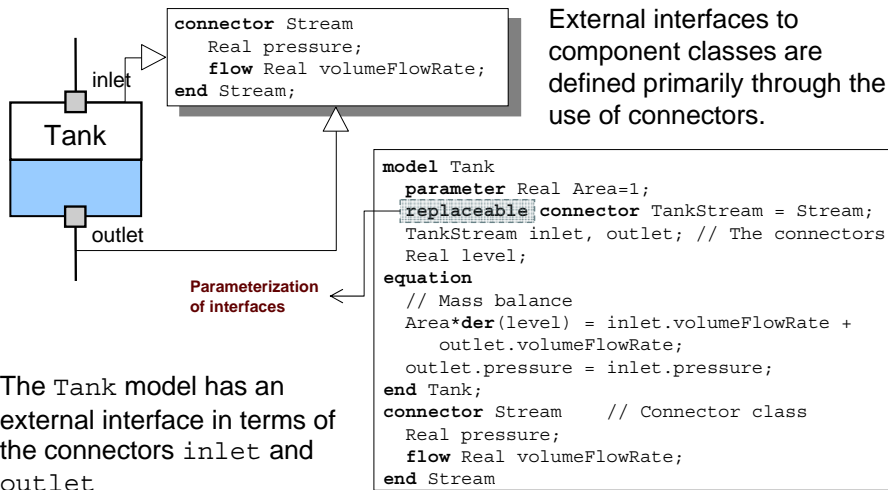
```
class M "Structured class M"
  RealInput u; // Outside input connector
  RealOutput y; // Outside output connector
  C C2;
  C C3;
end M;
```

A circuit in which the middle components C2 and C3 are placed inside a structured component M1 to which two outside connectors M1.u and M1.y have been attached.

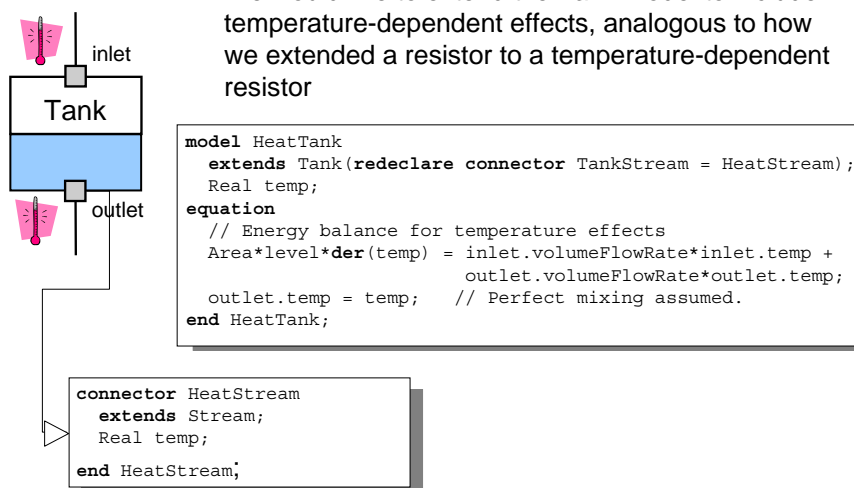
```
class MInst
  M M1; // Instance of M
equation
  connect(C1.y, M1.u); // Normal connection of outPort to inPort
  connect(M1.u, C2.u); // Outside inPort connected to inside inPort
  connect(C2.y, C3.u); // Inside outPort connected to inside inPort
  connect(C3.y, M1.y); // Inside outPort connected to outside outPort
  connect(M1.y, C4.u); // Normal connection of outPort to inPort
end MInst;
```



## Parameterization and Extension of Interfaces

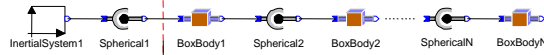


## Parameterization and Extension of Interfaces – cont'



## Arrays of Connectors

Part built up with a for-equation (see Lecture 4)



The model uses a for-equation to connect the different segments of the links

```

model ArrayOfLinks
  constant Integer n=10 "Number of segments (>0)";
  parameter Real[3,n] r={fill(1,n),zeros(n),zeros(n)};
  ModelicaAdditions.MultiBody.Parts.InertialSystem InertialSystem1;
  ModelicaAdditions.MultiBody.Parts.BoxBody[n]
    boxBody(r = r, Width=fill(0.4,n));
  ModelicaAdditions.MultiBody.Joints.Spherical spherical[n];
equation
  connect(InertialSystem1.frame_b, spherical[1].frame_a);
  connect(spherical[1].frame_b, boxBody[1].frame_a);
  for i in 1:n-1 loop
    connect(boxBody[i].frame_b, spherical[i+1].frame_a);
    connect(spherical[i+1].frame_b, boxBody[i+1].frame_a);
  end for;
end ArrayOfLinks;
  
```