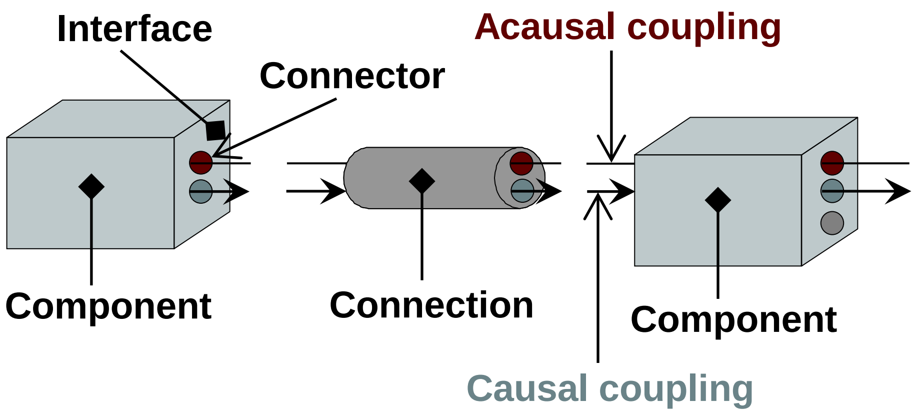
Components, Connectors and Connections

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*

electrical connector

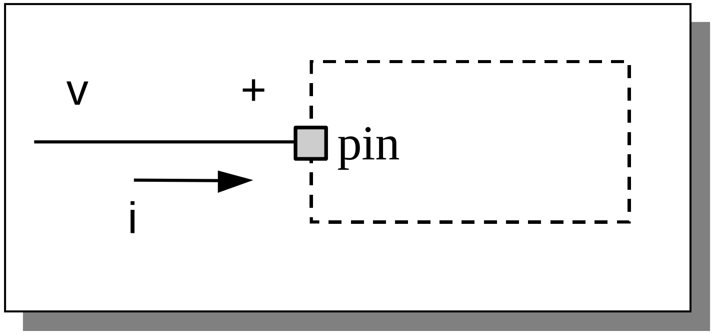
connector class

```
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`



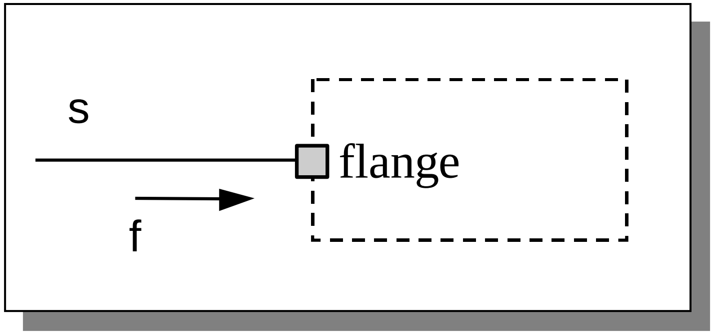
mechanical connector

connector class

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

Flange flange;
```

an instance `flange` of class `Flange`



The `flow` prefix

Three possible kinds of variables in connectors:

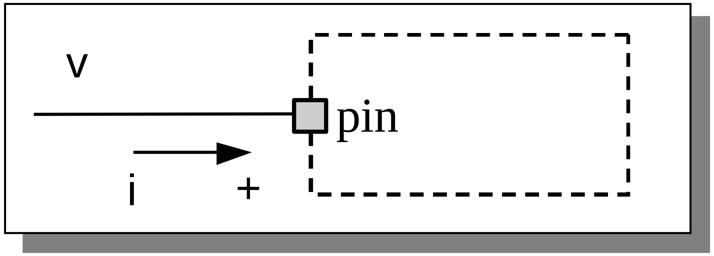
- *Potential variables potential* or energy level
- *Flow variables* represent some kind of flow
- *Stream variables* represent fluid flow in convective transport

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

positive flow direction:



Physical Connector Classes Based on Energy Flow

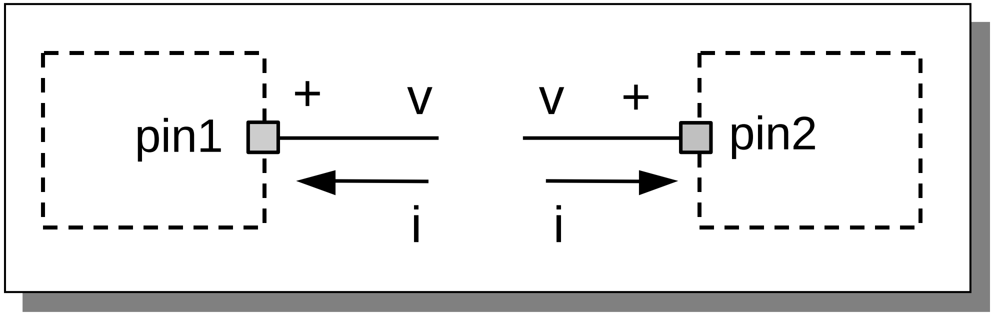
Domain Type	Potential	Flow	Carrier	Modelica Library
Electrical	Voltage	Current	Charge	`Electrical.Analog`
Translational	Position	Force	Linear momentum	`Mechanical.Translational`
Rotational	Angle	Torque	Angular momentum	`Mechanical.Rotational`
Magnetic	Magnetic potential	Magnetic flux rate	Magnetic flux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

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

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

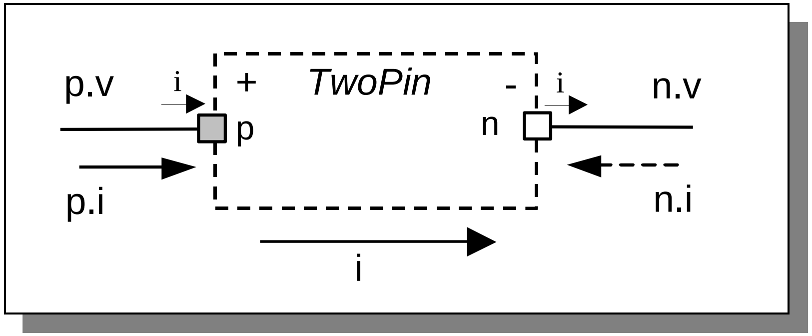
connector class

fixed causality

```
connector RealOutput =
  output Real;
```

Common Component Structure

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



partial class (cannot be instantiated)

positive pin

negative pin

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

electrical connector class

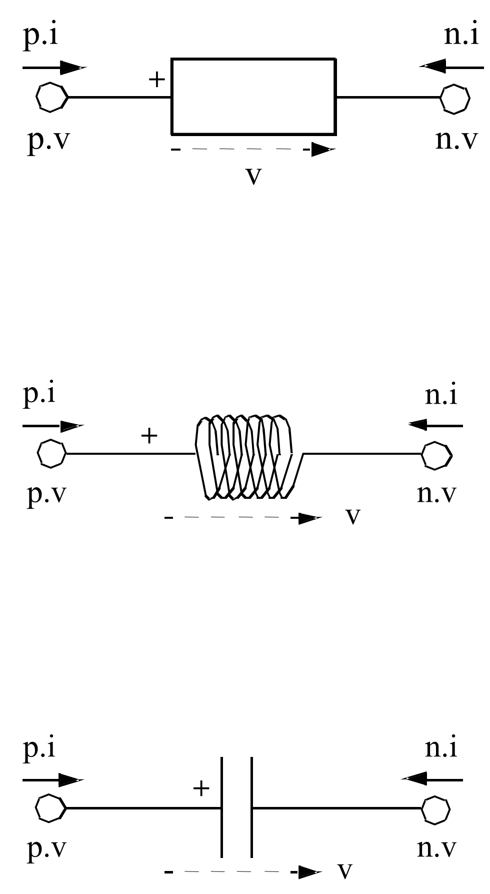
```
connector Pin
  Voltage       v;
  flow Current  i;
end Pin;
```

Electrical Components

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

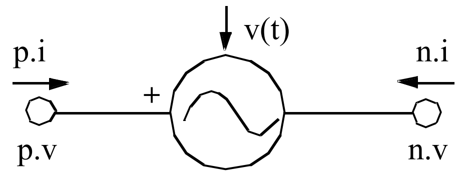
```
model Inductor "Ideal electrical inductor"
  extends TwoPin;
  parameter Real L "Inductance";
equation
  L*der(i) = v;
end Inductor;
```

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

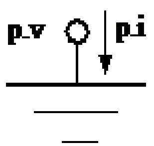


Electrical Components cont'

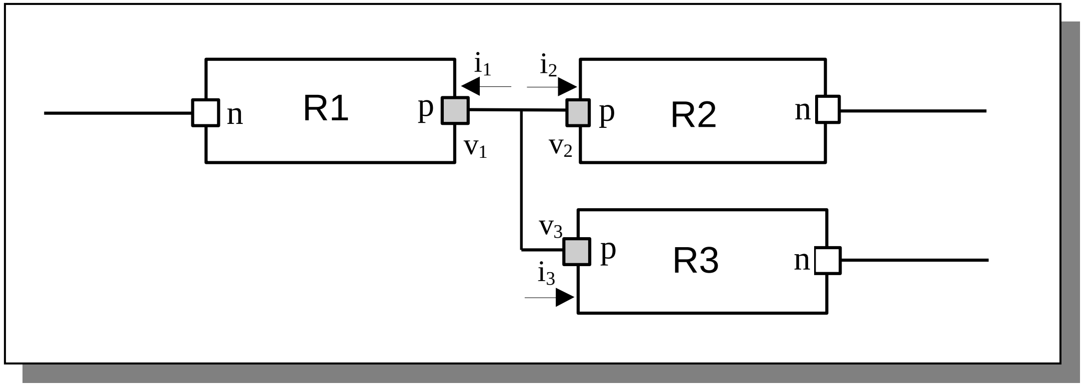
```
model Source
   extends TwoPin;
   parameter Real A,w;
equation
   v = A*sin(w*time);
end Resistor;
```




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



Resistor Circuit




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

Corresponds to

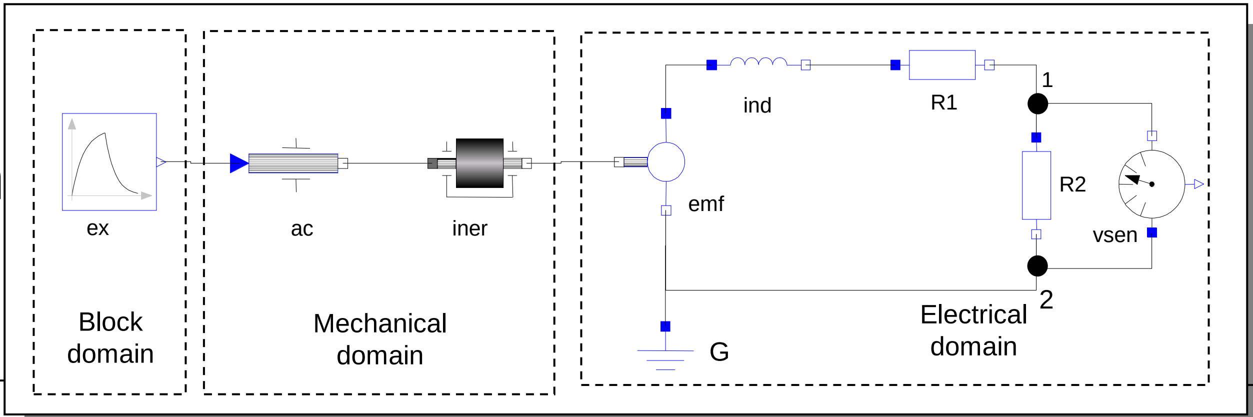
```
R1.p.v = R2.p.v;
R1.p.v = R3.p.v;
R1.p.i + R2.p.i + R3.p.i = 0;
```

SimpleCircuit Textual Modeling Exercise

- Exercise03-classes-textual-circuit.onb

Connecting Components from Multiple Domains

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

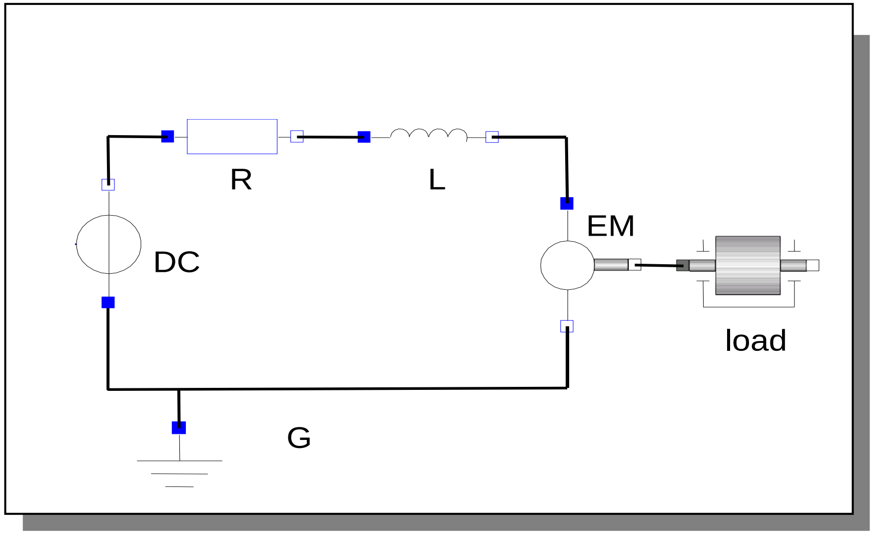



```
model Generator
  Modelica.Mechanics.Rotational.Accelerate ac;
  Modelica.Mechanics.Rotational.Inertia iner;
  Modelica.Electrical.Analog.Basic.EMF emf(k=-1);
  Modelica.Electrical.Analog.Basic.Inductor ind(L=0.1);
  Modelica.Electrical.Analog.Basic.Resistor R1,R2;
  Modelica.Electrical.Analog.Basic.Ground 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
DC.p.v == R.n.v

0 == R.p.i + L.n.i
R.p.v == L.n.v

0 == L.p.i + EM.n.i
L.p.v == EM.n.v

0 == EM.p.i + DC.n.i
EM.p.v == DC.n.v

0 == DC.n.i + G.p.i
DC.n.v == G.p.v

EM.u == EM.p.v - EM.n.v
0 == EM.p.i + EM.n.i
EM.i == EM.p.i
EM.u == EM.k * EM.ω
EM.i == EM.M / EM.k
EM.J * EM.ω == EM.M - EM.b * EM.ω

DC.u == DC.p.v - DC.n.v
0 == DC.p.i + DC.n.i
DC.i == DC.p.i
DC.u == DC.Amp * Sin[2 π DC.f * t]

R.u == R.p.v - R.n.v
0 == R.p.i + R.n.i
R.i == R.p.i
R.u == R.R * R.i

L.u == L.p.v - L.n.v
0 == L.p.i + L.n.i
L.i == L.p.i
L.u == L.L * L.i'
```

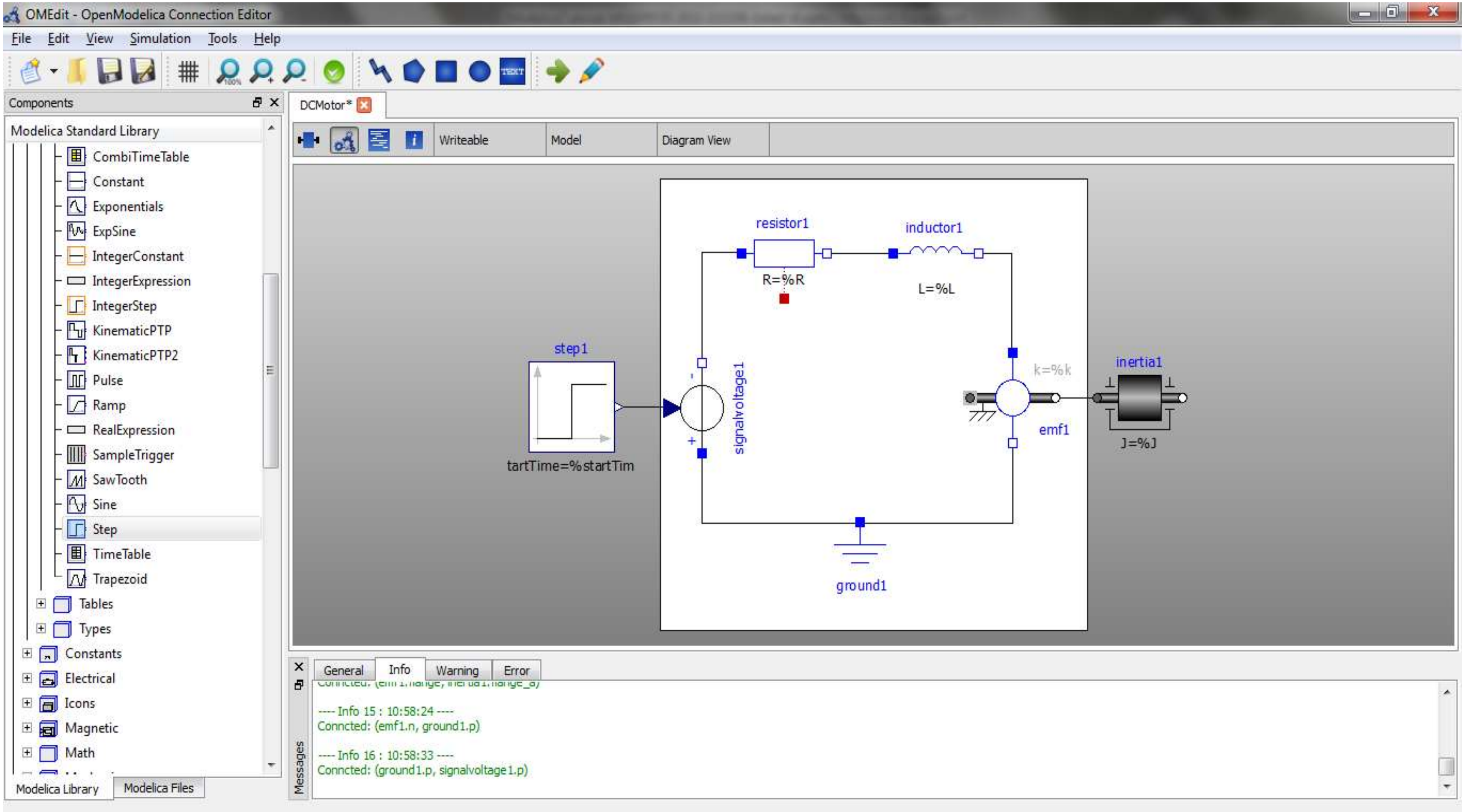
(`load` component not included)

Automatic transformation to ODE or DAE for simulation:

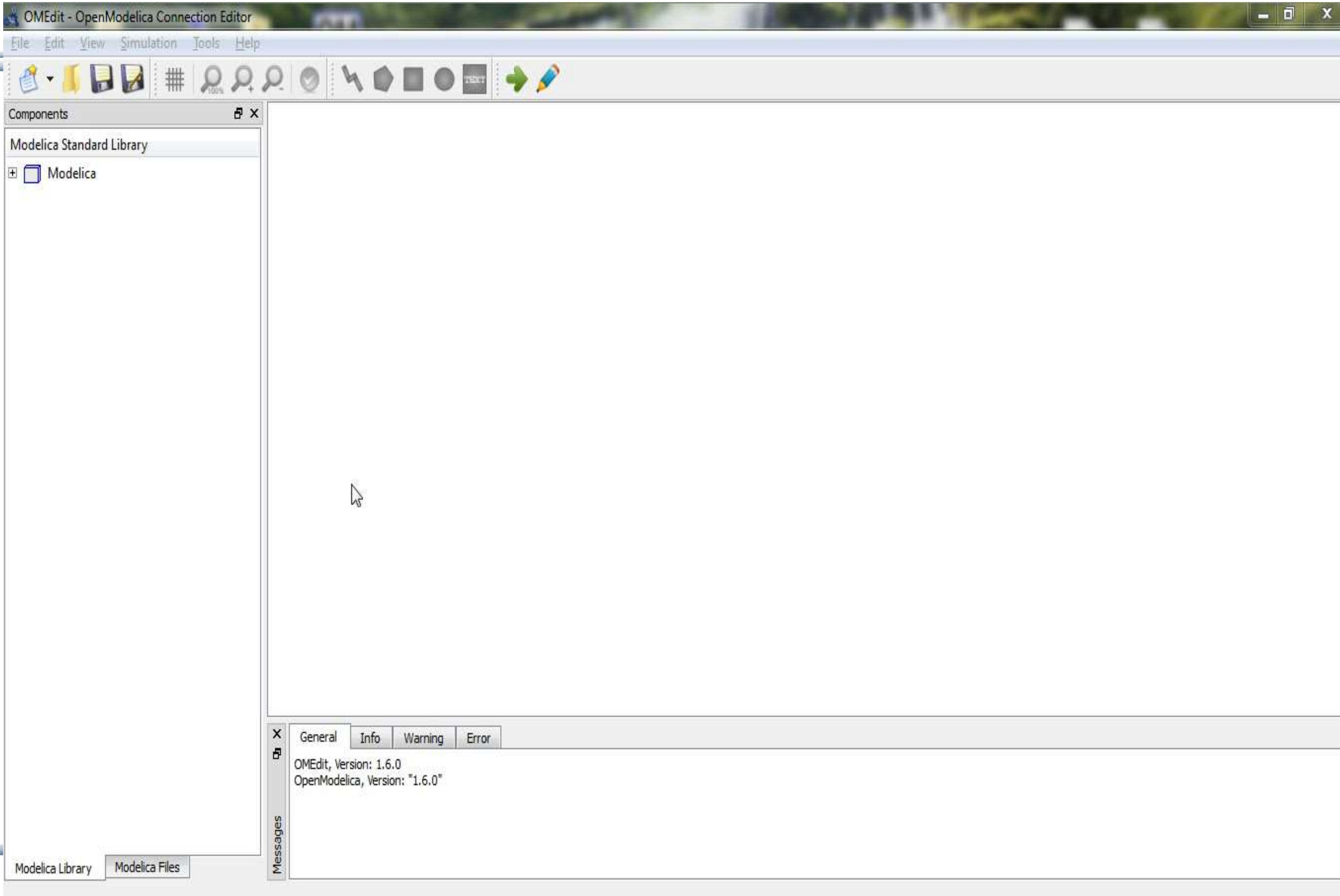
$$\frac{dx}{dt} == f[x, u, t] \qquad g\left[\frac{dx}{dt}, x, u, t\right] == 0$$

MODELICA

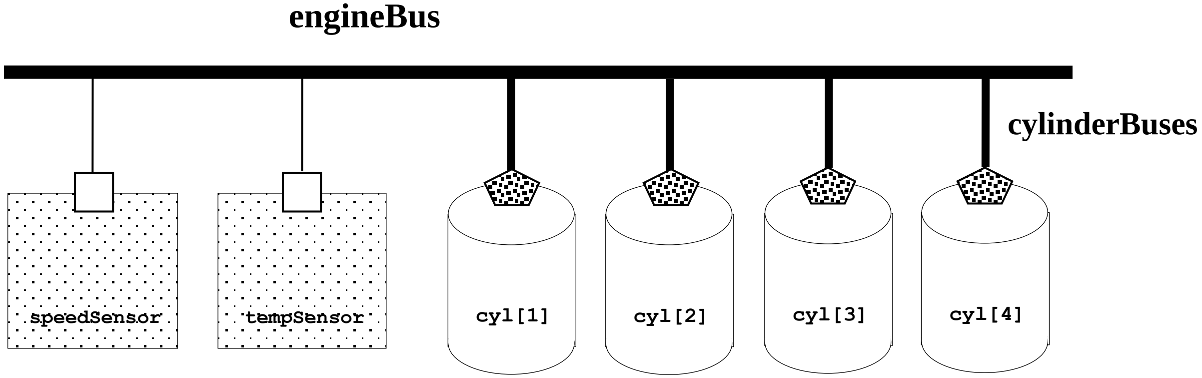
Graphical Modeling - Using Drag and Drop Composition – DCMotor with signalVoltage and step signal



Graphical Modeling Animation – DCMotor



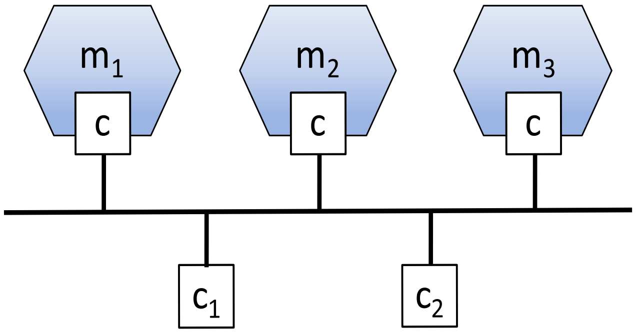
Expandable Connectors for Information Buses



Engine with engineBus, speedSensor, tempSensor and four nested cylinder buses which connect the cylinders with the engineBus Each cylinder contains a spark plug

```
model SparkPlug
  Real sparkAdvance;
  ...
end SparkPlug;
expandable connector EngineBus
end EngineBus;
expandable connector CylinderBus
  Real sparkAdvance;
end CylinderBus;
model Cylinder
  CylinderBus cylinderBus;
  SparkPlug sparkPlug;
  ...
```

Stream Connectors for Fluid Systems



Example `FluidSystem` model with three (N=3) inside connectors m_1.`c`, m_2.c, m_3.c and two (M=2) outside connectors c_1 and c_2

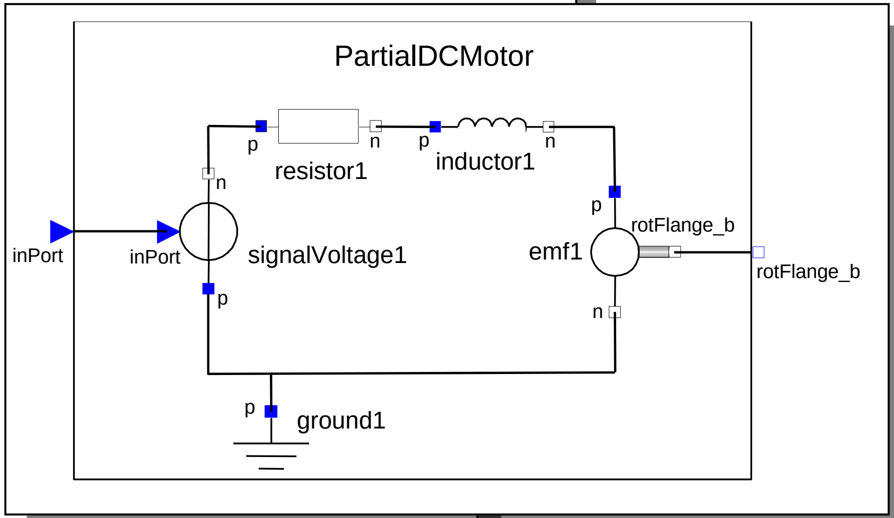
```
connector FluidPort
  flow   Real m_flow     "Flow of matter; m_flow > 0 if flow into component";
  stream Real h_outflow "Specific variable in component if m_flow < 0"
end FluidPort
model FluidSystem
  FluidComponent m1, m2, m3;
  FluidPort      c1, c2, c3;
equation
  connect(m1.c, m2.c);
  connect(m1.c, m3.c);
  connect(m1.c, c1);
  connect(m1.c, c2);
end FluidSystem;
```

Hierarchically Structured Components

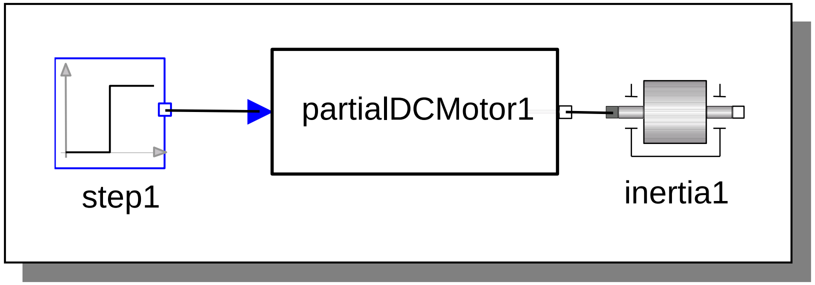
An *inside connector* is a connector belonging to an *internal component* of a structured component class.

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;
```



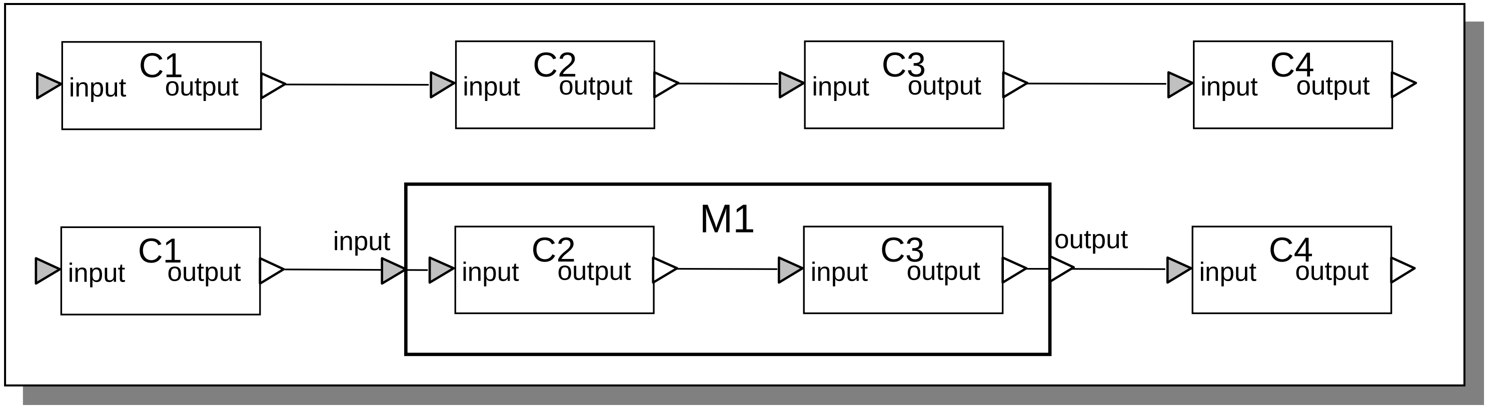
Hierarchically Structured Components cont'




```
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'

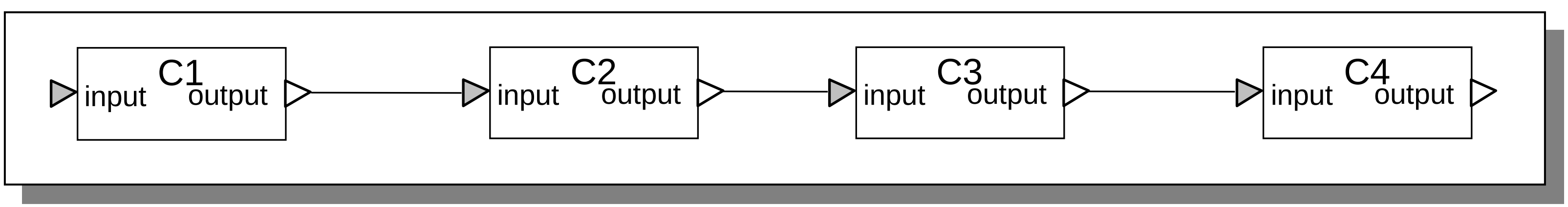
```
connector RealInput
  =  input Real;
```

```
connector RealOutput
  = output Real;
```

```
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`



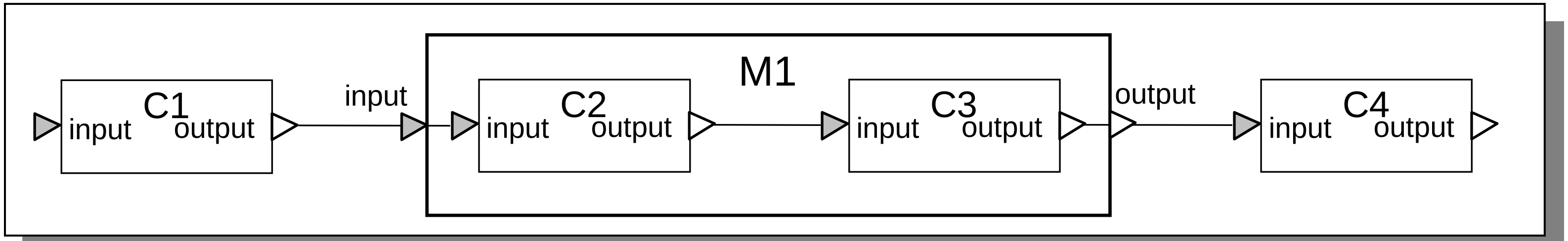
MODELICA

Connector Restrictions cont’

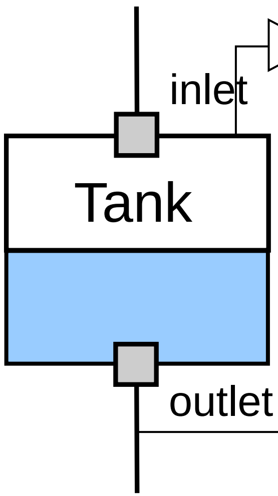
```
class M   "Structured class M"
  RealInput  u; // Outside input connector
  RealOutput y; // Outside output connector
  C  C2;
  C  C3;
end M;
```

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




```
connector Stream
   Real pressure;
   flow Real volumeFlowRate;
end Stream;
```

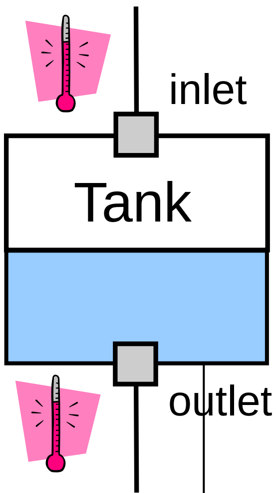
External interfaces to component classes are defined primarily through the use of connectors.

```
model Tank
  parameter Real Area=1;
  replaceable connector TankStream = Stream;
  TankStream inlet, outlet; // The connectors
  Real level;
equation
  // Mass balance
  Area*der(level) = inlet.volumeFlowRate +
     outlet.volumeFlowRate;
  outlet.pressure = inlet.pressure;
end Tank;
connector Stream     // Connector class
  Real pressure;
  flow Real volumeFlowRate;
end Stream
```

Parameterization of interfaces

The `Tank` model has an external interface in terms of the connectors `inlet` and `outlet`

Parameterization and Extension of Interfaces – cont'



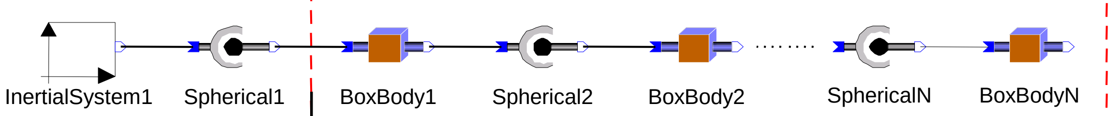
We would like to extend the Tank model to include temperature-dependent effects, analogous to how we extended a resistor to a temperature-dependent resistor

```
model HeatTank
  extends Tank(redeclare connector TankStream = HeatStream);
  Real temp;
equation
  // Energy balance for temperature effects
  Area*level*der(temp) = inlet.volumeFlowRate*inlet.temp +
                         outlet.volumeFlowRate*outlet.temp;
  outlet.temp = temp;   // Perfect mixing assumed.
end HeatTank;
```

```
connector HeatStream
  extends Stream;
  Real temp;
end HeatStream;
```

Arrays of Connectors

Part built up with a for-equation (see Lecture 5)



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;
```


Graphical Exercise 3.1

- Draw the `DCMotor` model according to the picture below using the *graphic connection editor* using models from the following Modelica libraries:
  `Mechanics.Rotational,`
  `Electrical.Analog.Basic,`
  `Electrical.Analog.Sources –` **`signalVoltage`**
  `and` **`step`** `in Blocks.Sources`

- Simulate it for 15s (use the simulation Setup S) and plot the variables for the outgoing rotational speed on the inertia axis and the voltage on the voltage source in the same plot.
  Hint: right-click and re-simulate setup on the simulation result to change simulation time