Components, Connectors and Connections
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 are instances of *connector classes*

**electrical connector**

**connector class**

**keyword** `flow` indicates that currents of connected pins sum to zero.

**an instance** `pin` of class `Pin`

**mechanical connector**

**connector class**

**an instance** `flange` of class `Flange`
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:](image)
## 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></td>
</tr>
<tr>
<td>Hydraulic</td>
<td>Pressure</td>
<td>Volume flow</td>
<td>Volume</td>
<td>HyLibLight</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>Under construction</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.

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:

\[ v_1 = v_2 = v_3 = \ldots v_n \]

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

```modelica
partial model TwoPin
  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;
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;
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**

```modelica
fixed causality

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;

corrected

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

multiple input single output block
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;
```
The following equations are automatically derived from the Modelica model:

<table>
<thead>
<tr>
<th>Equation 1</th>
<th>Equation 2</th>
<th>Equation 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 == DC.p.i + R.n.i</td>
<td>EM.u == EM.p.v - EM.n.v</td>
<td>R.u == R.p.v - R.n.v</td>
</tr>
<tr>
<td>DC.p.v == R.n.v</td>
<td>0 == EM.p.i + EM.n.i</td>
<td>0 == R.p.i + R.n.i</td>
</tr>
<tr>
<td></td>
<td>EM.i == EM.p.i</td>
<td>R.i == R.p.i</td>
</tr>
<tr>
<td>0 == R.p.i + L.n.i</td>
<td>EM.u == EM.k * EM.ω</td>
<td>R.u == R.R * R.i</td>
</tr>
<tr>
<td>R.p.v == L.n.v</td>
<td>EM.i == EM.M / EM.k</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EM.J * EM.ω == EM.M - EM.b * EM.ω</td>
<td>L.u == L.p.v - L.n.v</td>
</tr>
<tr>
<td>0 == L.p.i + EM.n.i</td>
<td>0 == DC.p.v - DC.n.v</td>
<td>0 == L.p.i + L.n.i</td>
</tr>
<tr>
<td>L.p.v == EM.n.v</td>
<td>DC.u == DC.p.v - DC.n.v</td>
<td>L.i == L.p.i</td>
</tr>
<tr>
<td></td>
<td>0 == DC.p.i + DC.n.i</td>
<td>L.u == L.L * L.i'</td>
</tr>
<tr>
<td>0 == EM.p.i + DC.n.i</td>
<td>DC.i == DC.p.i</td>
<td></td>
</tr>
<tr>
<td>EM.p.v == DC.n.v</td>
<td>DC.u == DC.Amp * Sin[2 π DC.f * t]</td>
<td></td>
</tr>
<tr>
<td>0 == DC.n.i + G.p.i</td>
<td></td>
<td>(load component not included)</td>
</tr>
<tr>
<td>DC.n.v == G.p.v</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Automatic transformation to ODE or DAE for simulation:

\[
\frac{dx}{dt} = f[x, u, t] \quad g\left[\frac{dx}{dt}, x, u, t\right] = 0
\]
Graphical Modeling - Using Drag and Drop Composition
Graphical Modeling Animation – DCMotor
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.
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
A circuit consisting of four connected components $C_1$, $C_2$, $C_3$, and $C_4$ which are instances of the class $C$
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.
Parameterization and Extension of Interfaces

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

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;
end Stream;
```
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

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));
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;
```
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.
Dodatek:
Třída má jednu definici a libovolně instanci

- **Block**
  - „input“, „output“ konektory

- **Class**
  - Vše

- **Connector**
  - Spájení - „connect(c1, c2);“

- **Function**
  - Jako Block s algoritmem

- **Model**
  - Model

- **Package**
  - Prostor jmen

- **Record**
  - Bez rovnic a algoritnů

- **Type**
  - Typ proměnných

- **ExternalObject**
  - Externý objekt (napr. C/C++, ..)

---

**Sekce equation ..**
**Rovnice**
**Sekce Algorithm ..**
**Algoritmus**

**Počítání rovnic**
**Model jako soustava rovnic**
Flow proměnná v konektoru

$m1.m\_flow \rightarrow m2.m\_flow$

$m1.m\_flow > 0$

$0 = m1.m\_flow + m2.m\_flow$
flow Real m_flow;  // tok látky – transport hmoty
stream Real h;    // specifická enthalpie – množství transportované s hmotou

\[
\begin{align*}
0 &= m_1.m\_flow \times (\text{if } m_1.m\_flow > 0 \text{ then } h_{\text{mix}} \text{ else } m_1.h) + \\
&\quad m_2.m\_flow \times (\text{if } m_2.m\_flow > 0 \text{ then } h_{\text{mix}} \text{ else } m_2.h) + \\
&\quad m_3.m\_flow \times (\text{if } m_3.m\_flow > 0 \text{ then } h_{\text{mix}} \text{ else } m_3.h) \\
0 &= m_1.m\_flow + m_2.m\_flow + m_3.m\_flow
\end{align*}
\]

Martin Otter, Francesco Casella: Overview and Rationale for Modelica Stream connectors