Introduction to Modelica

Michael Wetter
Simulation Research Group

February 11, 2019

BERKELEY LAB

Lawrence Berkeley National Laboratory

Introduction

Prerequisite

Having a basic understanding of what Modelica is.

Purpose

Gain better understanding of Modelica and be able to develop simple models.

Resources

The slides follow largely, and use many examples from, the online book from Michael Tiller:
http://book.xogeny.com

Modelica reference: http://modref.xogeny.com/

Other references:
http://simulationresearch.lbl.gov/modelica/userGuide/gettingStarted.html

Interactive tour: http://tour.xogeny.com

Basic syntax

Basic equations

Consider

$\dot{x} = 1 - x$

Initial conditions

$x_0 = 2$

```
model FirstOrderInitial "First order equation with initial value"
  Real x "State variable";
initial equation
  x = 2 "Used before simulation to compute initial values";
equation
  der(x) = 1-x "Drives value of x toward 1.0";
end FirstOrderInitial;
```

$\dot{x}_0 = 0$

```
model FirstOrderSteady
  "First order equation with steady state initial condition"
  Real x "State variable";
initial equation
  der(x) = 0 "Initialize the system in steady state";
equation
  der(x) = 1-x "Drives value of x toward 1.0";
end FirstOrderSteady;
```

Adding units

$$m\,c_p\,\frac{dT}{dt} = h\,A\,(T_{inf} - T)$$

```
model NewtonCoolingWithUnits "Cooling example with physical units"
  parameter Real T_inf(unit="K")=298.15 "Ambient temperature";
  parameter Real T0(unit="K")=363.15 "Initial temperature";
  parameter Real h(unit="W/(m2.K)")=0.7 "Convective cooling coefficient";
  parameter Real A(unit="m2")=1.0 "Surface area";
  parameter Real m(unit="kg")=0.1 "Mass of thermal capacitance";
  parameter Real c_p(unit="J/(K.kg)")=1.2 "Specific heat";
  Real T(unit="K") "Temperature";
initial equation
  T = T0 "Specify initial value for T";
equation
  m*c_p*der(T) = h*A*(T_inf-T) "Newton's law of cooling";
end NewtonCoolingWithUnits;
```

To avoid this verbosity, `Modelica.SIunits` declares types such as

```
type Temperature=Real(unit="K", min=0);
```

Now, we can write

```
parameter Modelica.SIunits.Temperature T_inf=298.15 "Ambient temperature";
```

Discrete behavior

`If-then` to model time events

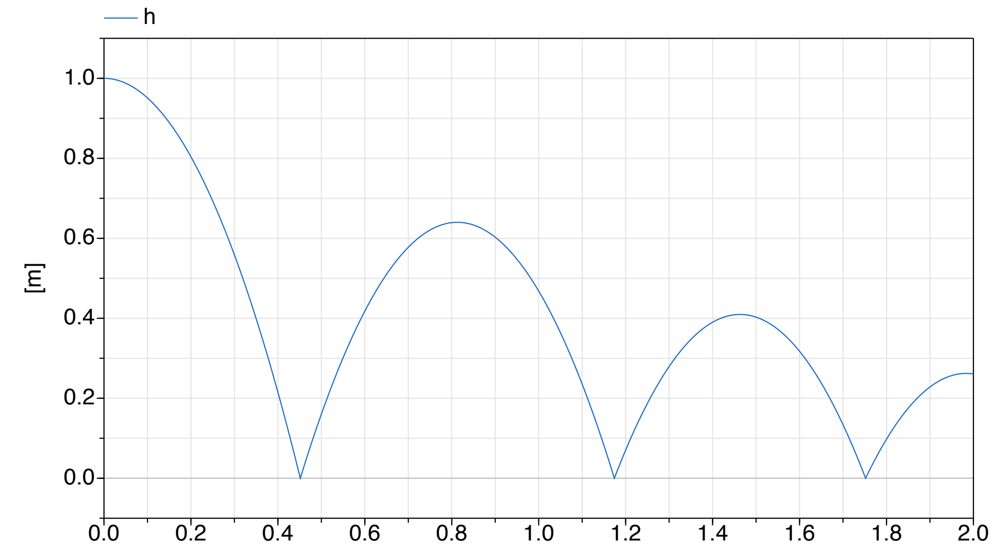
```
model NewtonCoolingDynamic
  "Cooling example with fluctuating ambient conditions"
…
initial equation
  T = T0 "Specify initial value for T";
equation
  if time<=0.5 then
    T_inf = 298.15 "Constant temperature when time<=0.5";
  else
    T_inf = 298.15-20*(time-0.5) "Otherwise, increasing";
  end if;
  m*c_p*der(T) = h*A*(T_inf-T) "Newton's law of cooling";
end NewtonCoolingDynamic;
```

Note: time is a built-in variable.

An alternative formulation is

```
T_inf = 298.15 - (if time<0.5 then 0 else 20*(time-0.5));
```

when construct




```
model BouncingBall "The 'classic' bouncing ball model"
  type Height=Real(unit="m");
  type Velocity=Real(unit="m/s");
  parameter Real e=0.8 "Coefficient of restitution";
  parameter Height h0=1.0 "Initial height";
  Height h;
  Velocity v;
initial equation
  h = h0;
equation
  v = der(h);
  der(v) = -9.81;
  when h<0 then
    reinit(v, -e*pre(v));
  end when;
end BouncingBall;
```

Becomes active *when* the condition becomes true

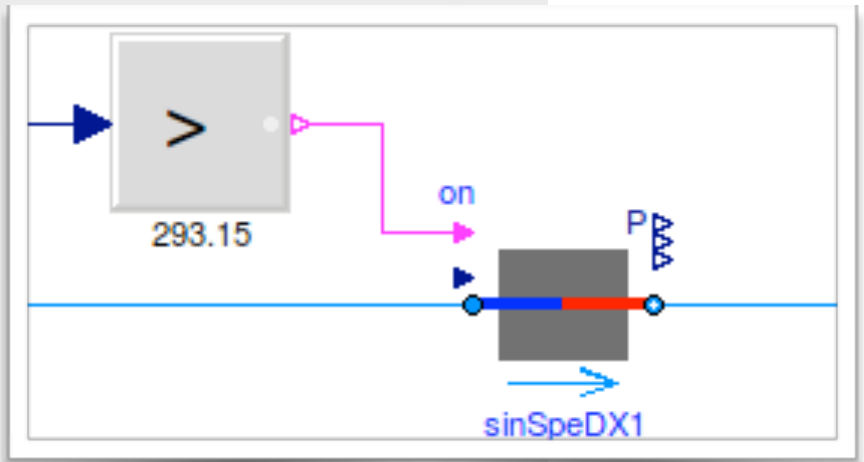
Reinitializes the state variable

Use the value that *v* had prior to this section

Avoid chattering by using hysteresis

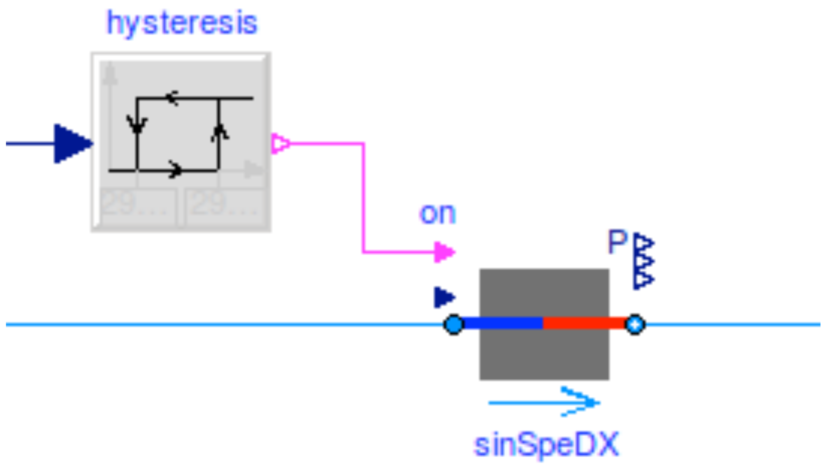
What will go wrong with this code?

```
model ChatteringControl "A control strategy that will ‘chatter'"

  type HeatCapacitance=Real(unit="J/K");
  type Temperature=Real(unit="K");
  type Heat=Real(unit="W");
  type Mass=Real(unit="kg");
  type HeatTransferCoefficient=Real(unit="W/K");

  parameter HeatCapacitance C=1.0;
  parameter HeatTransferCoefficient h=2.0;
  parameter Heat Qcapacity=25.0;
  parameter Temperature Tamb=285;
  parameter Temperature Tbar=295;

  Boolean heat "Indicates whether heater is on";
  Temperature T;
  Heat Q;
initial equation
  T = Tbar+5;
equation
  heat = T<Tbar;
  Q = if heat then Qcapacity else 0;
  C*der(T) = Q-h*(T-Tamb);
end ChatteringControl;
```

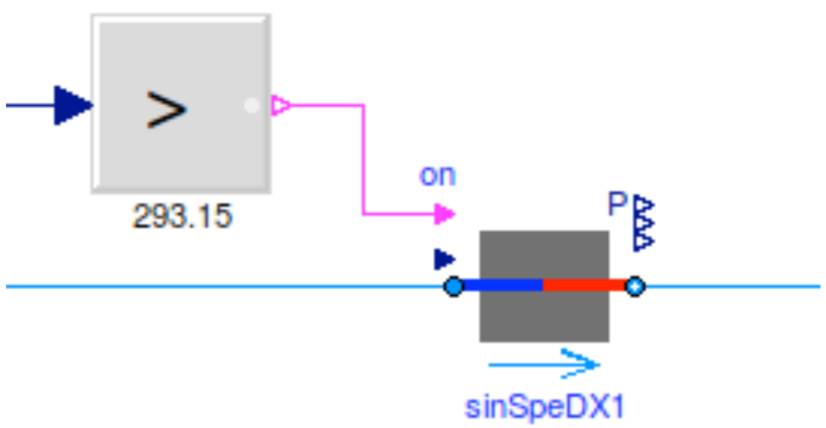


Such a problem was indeed reported by a user

Correct configuration



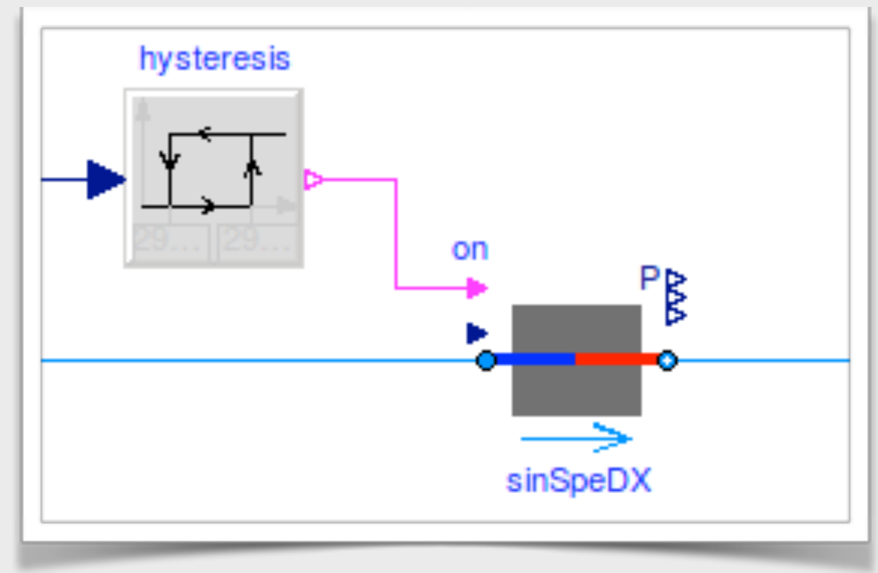
Avoid this configuration



This can work in fixed time step simulators, but it won't in variable time step simulators that handle events.

We need to add a hysteresis when switching the value of **heat**

```
model HysteresisControl "A control strategy that doesn't chatter"
  ...
  Boolean heat(start=false) "Indicates whether heater is on";
  parameter Temperature Tbar=295;
  Temperature T;
  Heat Q;
initial equation
  T = Tbar+5;
  heat = false;
equation
  Q = if heat then Qcapacity else 0;
  C*der(T) = Q-h*(T-Tamb);
  when {T>Tbar+1, T<Tbar-1} then
    heat = T<Tbar;
  end when;
end HysteresisControl;
```



Active when any element is true

Arrays

Arrays are fixed at compile time. They can be declared as

```
parameter Integer n = 3;

Real x[n];
Real y[size(x,1), 2];
Real z[:] = {2.0*i for i in 1:n};  // {2, 4, 6}
Real fives[:] = fill(5.0, n);      // {5, 5, 5}
```

Many functions take arrays as arguments, see http://book.xogeny.com/behavior/arrays/functions/.

For example, $s = \sum_{i=1}^{3} z_i$

```
s = sum(z);
```

Looping

```
parameter Integer n = 3;
Real x[n];
equation
  for i in 1:n loop
    x[i] = i;
  end for;
```

Functions

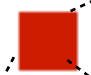
Functions have imperative programming assignments

```
within Buildings.Fluid.HeatExchangers.BaseClasses;

function lmtd "Log-mean temperature difference"
  input Modelica.SIunits.Temperature T_a1 "Temperature at port a1";
  input Modelica.SIunits.Temperature T_b1 "Temperature at port b1";
  input Modelica.SIunits.Temperature T_a2 "Temperature at port a2";
  input Modelica.SIunits.Temperature T_b2 "Temperature at port b2";
  output Modelica.SIunits.TemperatureDifference lmtd
    "Log-mean temperature difference";

protected
  Modelica.SIunits.TemperatureDifference dT1
   "Temperature difference side 1";
  Modelica.SIunits.TemperatureDifference dT2
    "Temperature difference side 2";

algorithm
  dT1  := T_a1 - T_b2;
  dT2  := T_b1 - T_a2;
  lmtd := (dT2 - dT1)/Modelica.Math.log(dT2/dT1);

annotation (…);
end lmtd;
```

`algorithm` sections can also be used in a `model` or `block` if needed.

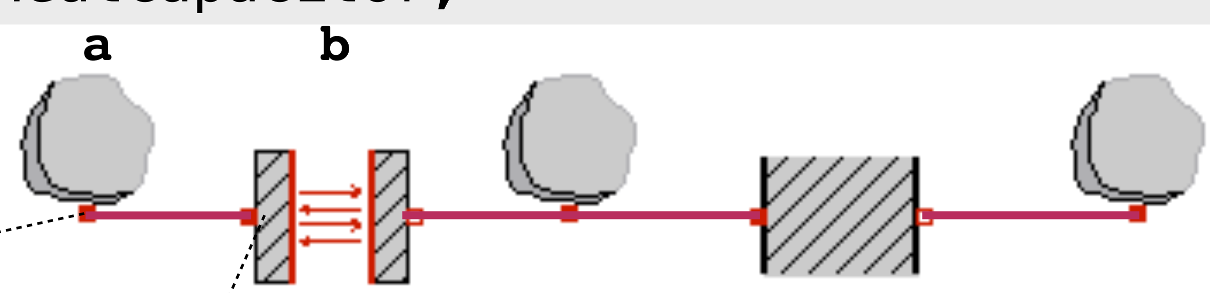
Object-oriented modeling

Object-oriented modeling



```
connector HeatPort_a "Thermal port for 1-dim. heat
transfer"
  Modelica.SIunits.Temperature T "Port temperature";
  flow Modelica.SIunits.HeatFlowRate Q_flow
    "Heat flow rate (positive if flowing from
     outside into the component)";
end HeatPort_a;
```

```
model HeatCapacitor "Lumped thermal element storing heat"

  parameter Modelica.SIunits.HeatCapacity C "Heat
capacity";
  Modelica.SIunits.Temperature T "Temperature of element";
  Interfaces.HeatPort_a port;

equation
  T = port.T;
  C*der(T) = port.Q_flow;
end HeatCapacitor;
```




```
connect(a.port, b.port);
```

```
a.port.T = b.port.T;
0 = a.port.Q_flow + b.port.Q_flow;
```

Packages

In Modelica, everything is part of a package

models are organized in hierarchical packages

```
within Buildings.Fluid.HeatExchangers.BaseClasses;
function lmtd "Log-mean temperature difference"
  input Modelica.SIunits.Temperature T_a1 "Temperature at port a1";
  …
```

Type definitions are part of packages

Packages can contain

- other packages,
- constants,
- functions,
- models,
- blocks,
- types

They cannot contain

- parameters,
- variable declaration,
- equation or algorithm sections

Packages

Basic syntax

```
package OuterPackage "A package that wraps a nested package"
  // Anything contained in OuterPackage
  package NestedPackage "A nested package"
    // Things defined inside NestedPackage
  end NestedPackage;
end OuterPackage;
```

Storing in the file system

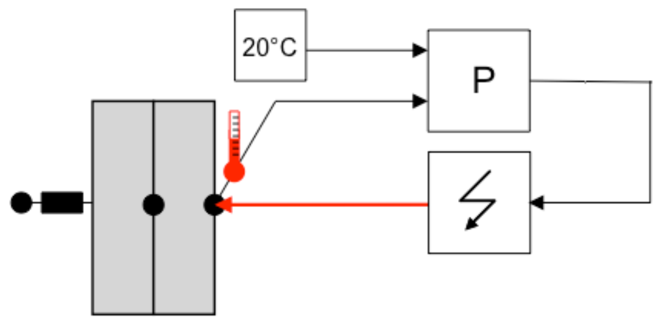
```
/RootPackage                 # Top-level package stored as a directory
  package.mo                 # Indicates this directory is a package
  package.order              # Specifies an ordering for this package
  NestedPackageAsFile.mo     # Definitions stored in one file
  /NestedPackageAsDir        # Nested package stored as a directory
    package.mo               # Indicates this directory is a package
    package.order            # Specifies an ordering for this package
```

Referencing resources with a URL

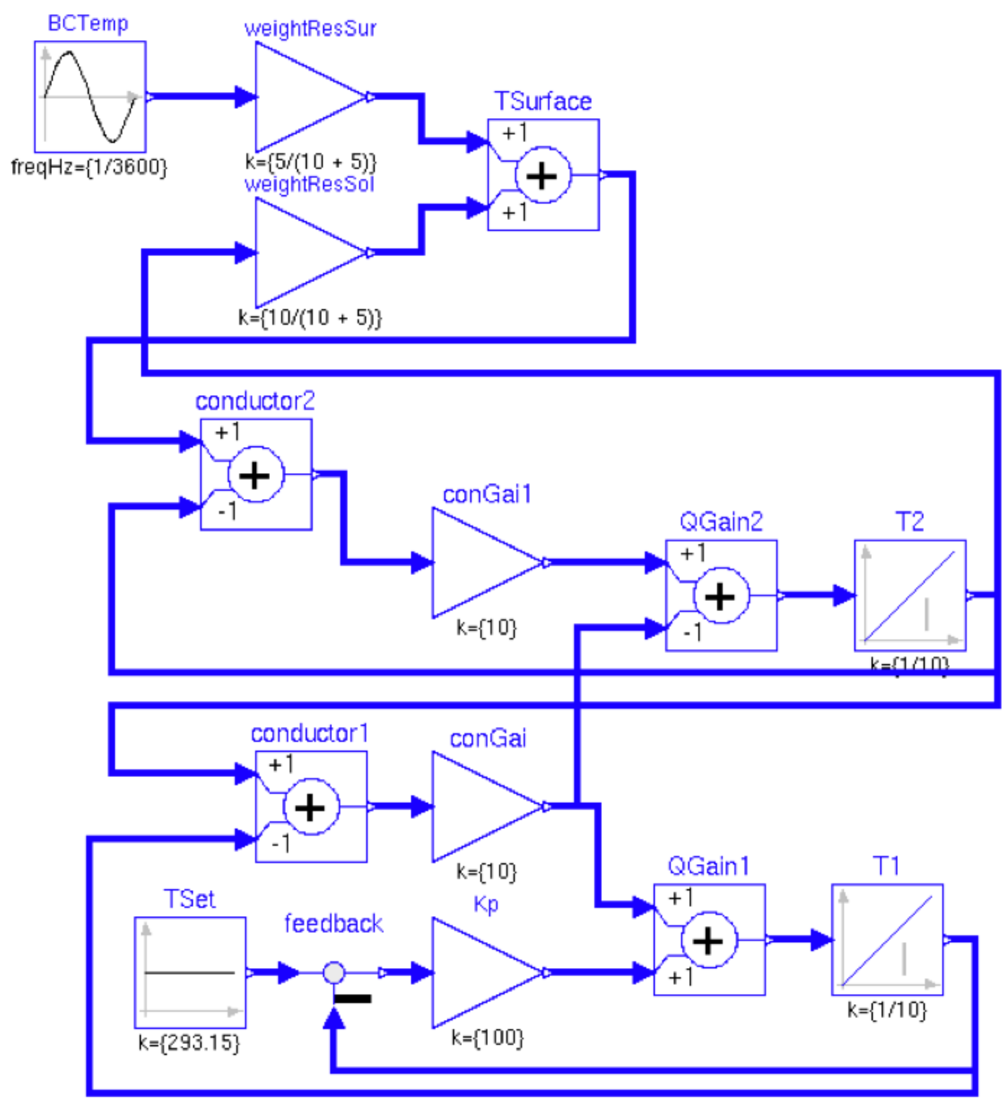
```
modelica://RootPackage/Resources/logo.jpg
```

A convention of the Buildings library is that each package must be in a separate directory, and each class in a separate file.
Reason: easier merging and version control.

Acausal connectors are used to enable assembling models schematically

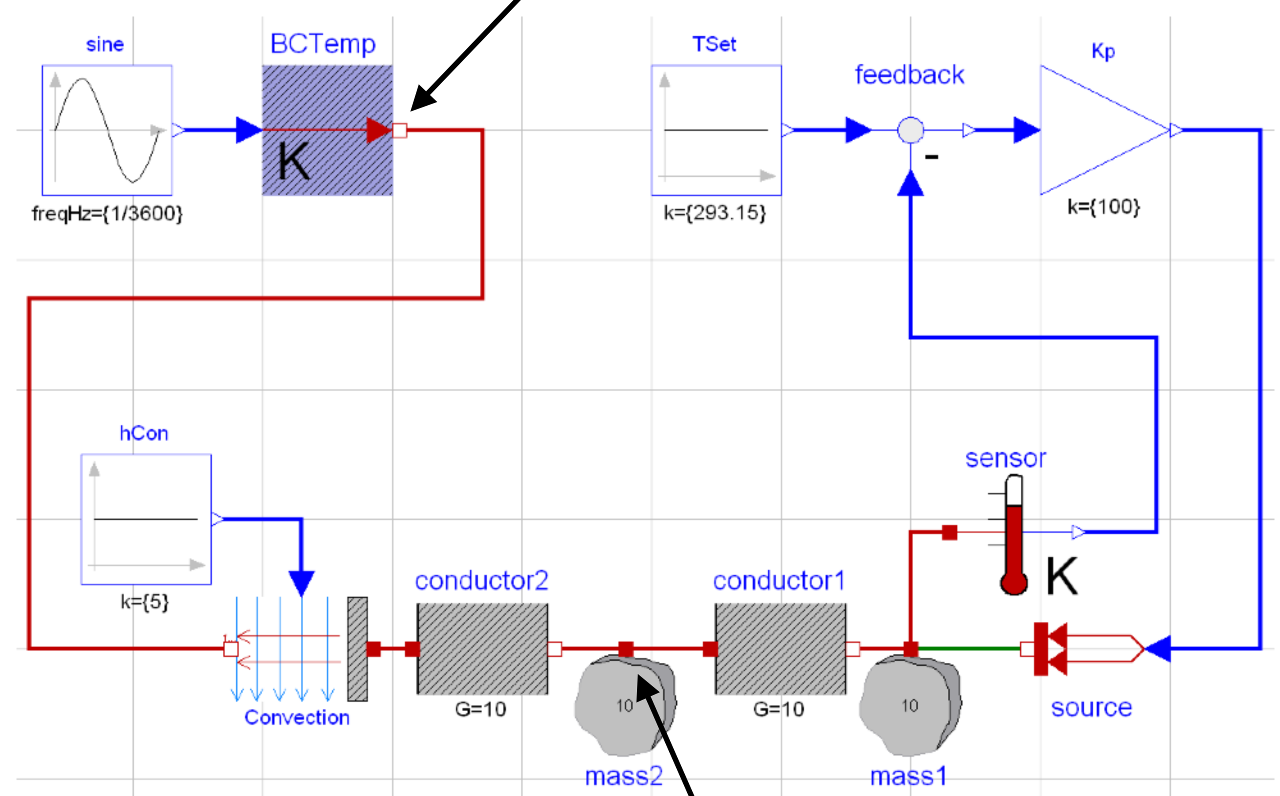


Block Diagram Modeling



Acausal Modeling

This port carries temperature and heat flow rate. Hence, an electrical pin cannot be connected to it



What does it mean to connect three ports?

Connectors declare the interfaces, or ports, of models.

No equations are allowed.

Connectors for most physical ports exists in the MSL.

```
 connector Thermal
     Modelica.SIunits.Temperature T;
     flow Modelica.SIunits.HeatFlowRate Q_flow;
  end Thermal;
```

```
connector FluidPort
  replaceable package Medium =
Modelica.Media.Interfaces.PartialMedium;

  flow Medium.MassFlowRate m_flow;
  Medium.AbsolutePressure p;
  stream Medium.SpecificEnthalpy h_outflow;
  stream Medium.MassFraction Xi_outflow[Medium.nXi];
  stream Medium.ExtraProperty C_outflow[Medium.nC];
end FluidPort;
```

Physical connectors and balanced models

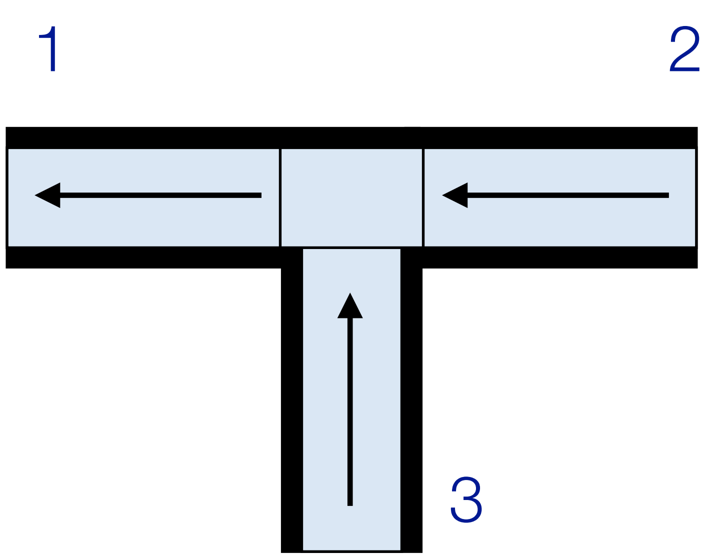
Domain	Potential	Flow	Stream
Heat flow	T	Q_flow	
Fluid flow	p	m_flow	h_outflow X_outflow C_outflow
Electrical	V	I	
Translational	x	F	

Acausal connectors

- input/output determined at compilation time
- can connect none or multiple components to a port
- A connector should contain all information required to uniquely define the boundary condition

Requirement of locally balanced models

- # of equations = # of variables, at each level of model hierarchy.



$$\sum_{i=1}^{3} \dot{m}_i = 0$$

$$p_1 = p_2 = p_3$$

$$h_{1,out} = -\frac{h_{2,in}\,\dot{m}_2 + h_{3,in}\,\dot{m}_3}{\dot{m}_1}$$

Components

A simple heat storage element

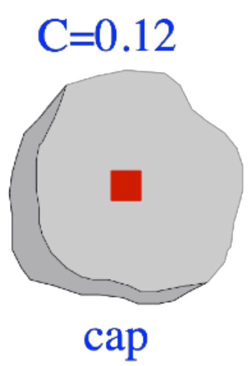
```
within Modelica.Thermal.HeatTransfer;

package Interfaces "Connectors and partial models"

  partial connector HeatPort "Thermal port for 1-dim. heat transfer"
    Modelica.SIunits.Temperature T "Port temperature";
    flow Modelica.SIunits.HeatFlowRate Q_flow
      "Heat flow rate (positive if flowing from outside into the
component)";
  end HeatPort;

  connector HeatPort_a
    "Thermal port for 1-dim. heat transfer (filled rectangular icon)"
    extends HeatPort;

    annotation(...,
      Icon(coordinateSystem(preserveAspectRatio=true,
                            extent={{-100,-100},{100,100}}),
                            graphics={Rectangle(
                              extent={{-100,100},{100,-100}},
                              lineColor={191,0,0},
                              fillColor={191,0,0},
                              fillPattern=FillPattern.Solid)}));
  end HeatPort_a;

end Interfaces;
```

A simple heat storage element

```
within ModelicaByExample.Components.HeatTransfer;
model ThermalCapacitance "A model of thermal capacitance"
  parameter Modelica.SIunits.HeatCapacity C "Thermal capacitance";
  parameter Modelica.SIunits.Temperature T0 "Initial temperature";
  Modelica.Thermal.HeatTransfer.Interfaces.HeatPort_a port
    annotation (Placement(transformation(extent={{-10,-10},{10,10}})));
initial equation
  port.T = T0;
equation
  C*der(port.T) = port.Q_flow;
end ThermalCapacitance;
```

$$C\frac{dT}{dt} = \dot{Q}$$

```
within ModelicaByExample.Components.HeatTransfer.Examples;
model Adiabatic "A model without any heat transfer"
  ThermalCapacitance cap(C=0.12, T0(displayUnit="K") = 363.15)
    "Thermal capacitance component"
    annotation (Placement(transformation(extent={{-30,-10},{-10,10}})));
end Adiabatic;
```



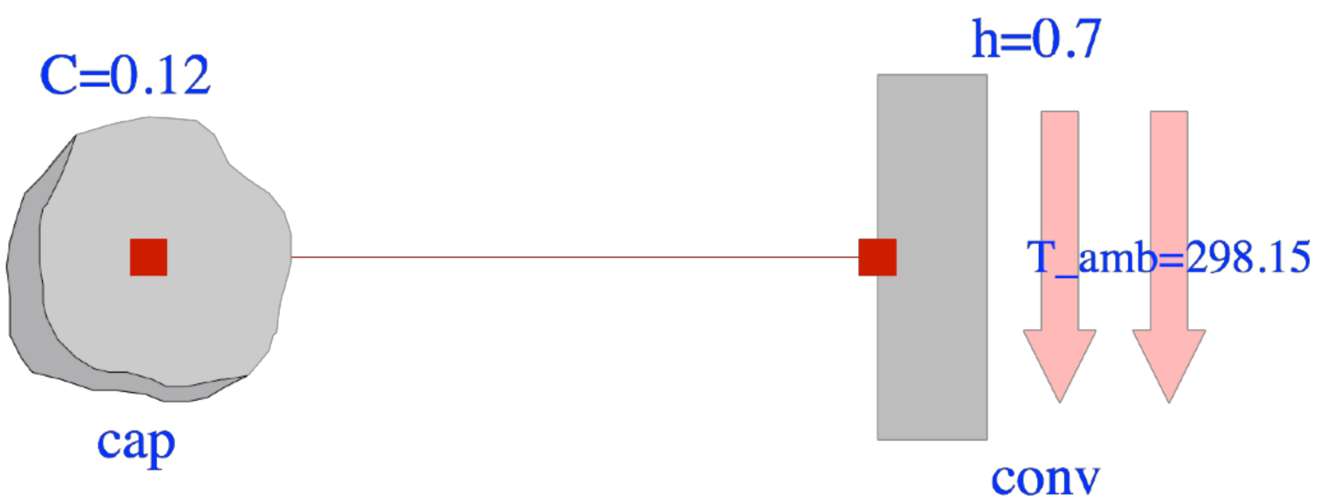
Add convection to ambient

```
within ModelicaByExample.Components.HeatTransfer;
model ConvectionToAmbient "An overly specialized model of convection"
  parameter Modelica.SIunits.CoefficientOfHeatTransfer h;
  parameter Modelica.SIunits.Area A;
  parameter Modelica.SIunits.Temperature T_amb "Ambient temperature";
  Modelica.Thermal.HeatTransfer.Interfaces.HeatPort_a port_a
    annotation (Placement(transformation(extent={{-110,-10},{-90,10}})));
equation
  port_a.Q_flow = h*A*(port_a.T-T_amb) "Heat transfer equation";
end ConvectionToAmbient;
```

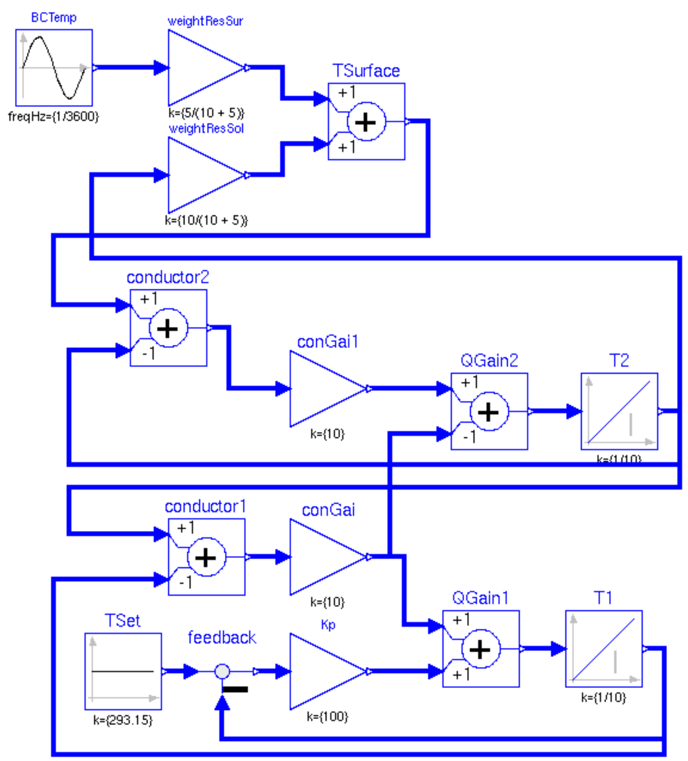
Add convection to ambient

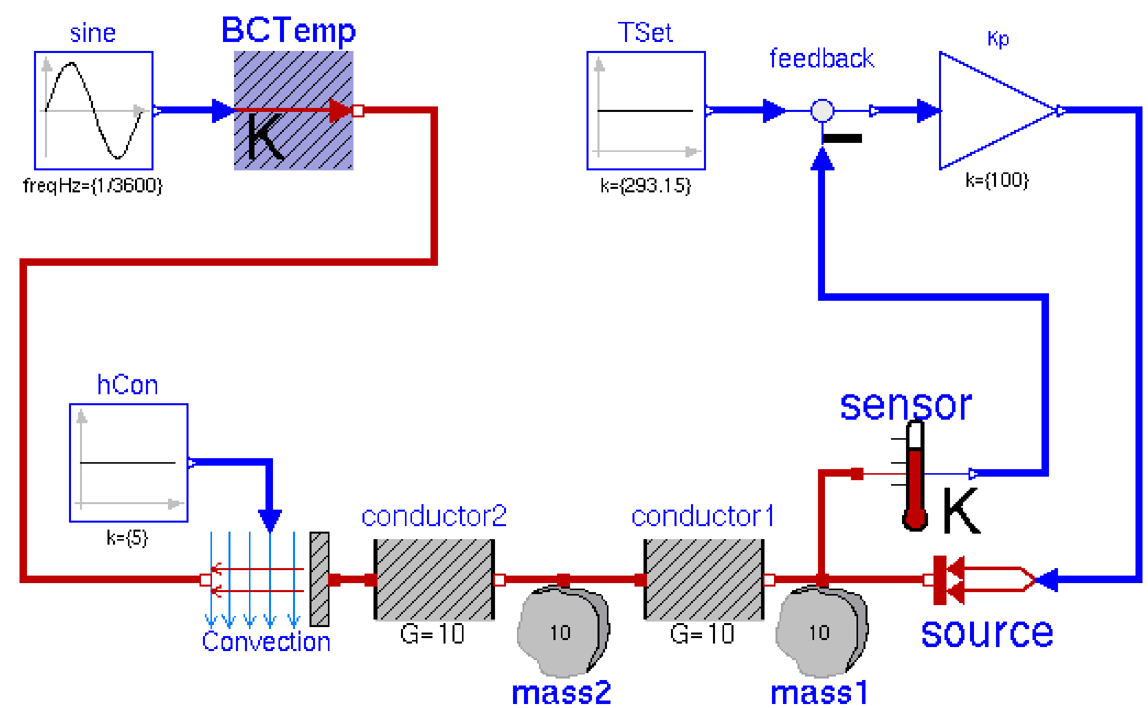
```
within ModelicaByExample.Components.HeatTransfer.Examples;
model CoolingToAmbient "A model using convection to an ambient condition"

  ThermalCapacitance cap(C=0.12, T0(displayUnit="K") = 363.15)
    "Thermal capacitance component"
    annotation (Placement(transformation(extent={{-30,-10},{-10,10}})));

  ConvectionToAmbient conv(h=0.7, A=1.0, T_amb=298.15)
    "Convection to an ambient temprature"
    annotation (Placement(transformation(extent={{20,-10},{40,10}})));

equation
  connect(cap.port, conv.port_a)
   annotation (
  Line(points={{-20,0},{20,0}},
       color={191,0,0},
      smooth=Smooth.None));
end CoolingToAmbient;
```

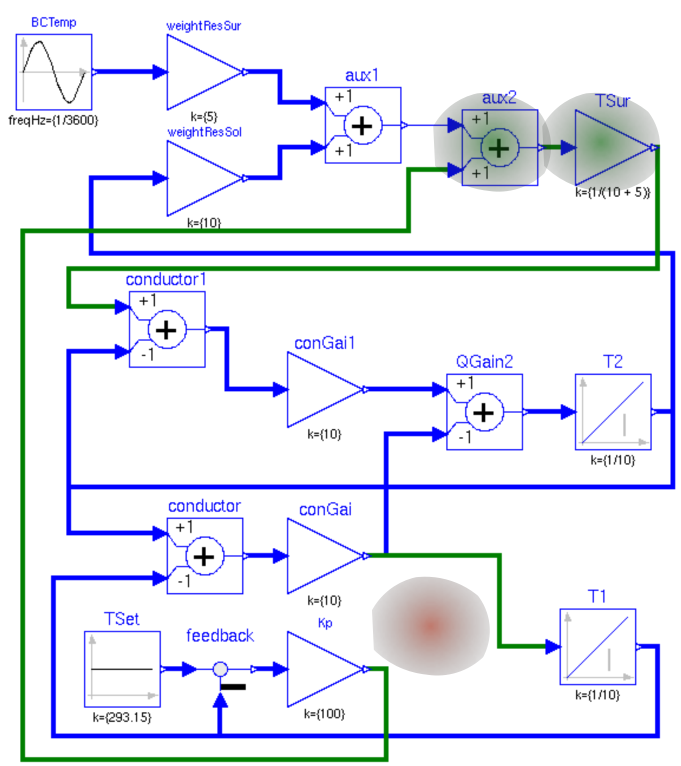
```
cap.port.T = conv.port_a.T;
cap.port.Q_flow + conv.port_a.Q_flow = 0;
```

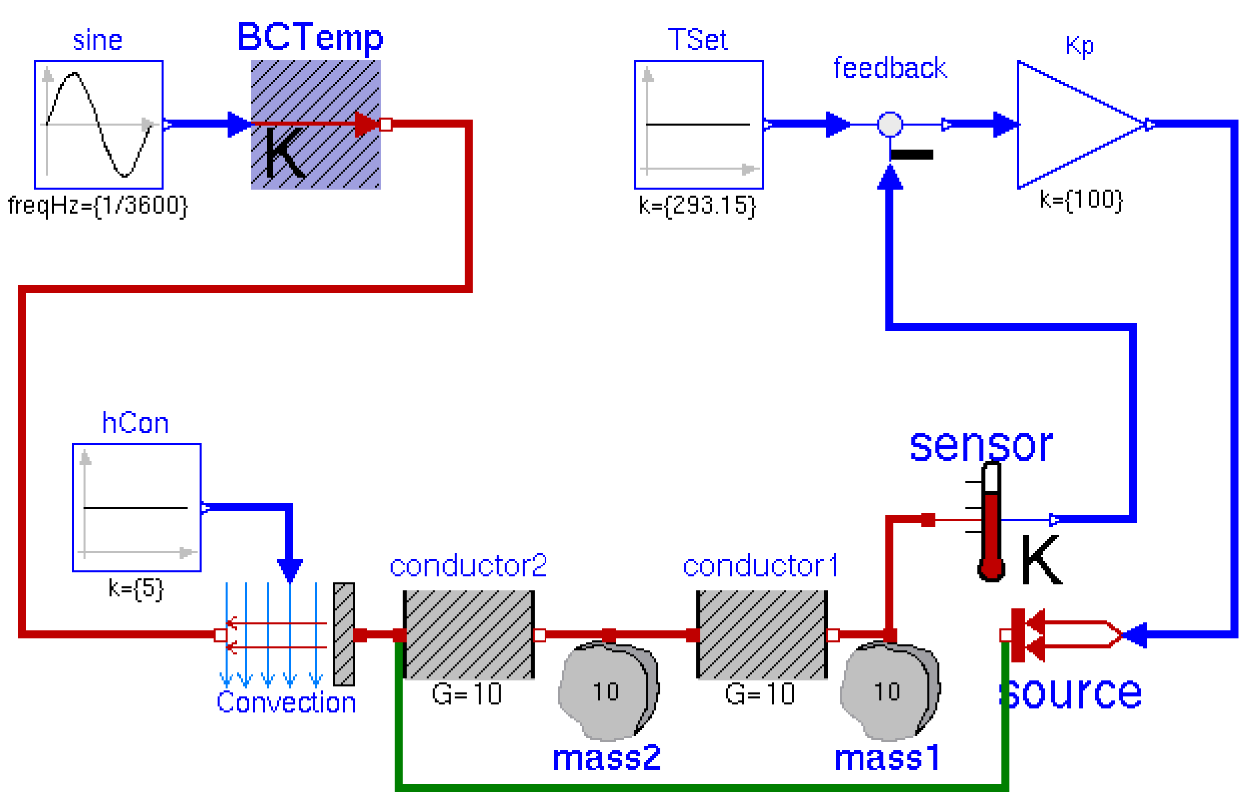


Acausal components leads to much higher readability and reusability









Thermofluid flow modeling

Stream of fluids

For electrical systems, we have a potential (voltage) that drives a flow (current).

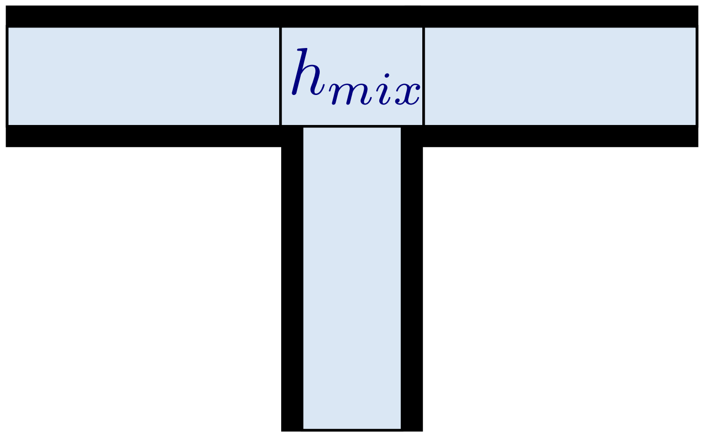
For fluids, we have a potential (pressure) that drives a flow (mass flow rate) **which carries properties (temperatures, mass concentration).**

Conservation of mass

$$0 = \sum_{i=1}^{n} \dot{m}_i$$

Conservation of energy

$$0 = \sum_{i=1}^{n} \dot{m}_i \begin{cases} h_{mix}, & \text{if } \dot{m}_i > 0 \\ h_{outflow,i}, & \text{otherwise.} \end{cases}$$



If mass flow rates are computed based on (nonlinear) pressure drops, this cannot be solved reliably as the residual equation depends on a boolean variable.

For reliable solution of fluid flow network, stream variables have been introduced

Connector variables have the `flow` and `stream` attribute:

```
connector FluidPort
  replaceable package Medium =
Modelica.Media.Interfaces.PartialMedium;

  flow Medium.MassFlowRate m_flow;
  Medium.AbsolutePressure p;
  stream Medium.SpecificEnthalpy h_outflow;
  stream Medium.MassFraction Xi_outflow[Medium.nXi];
  stream Medium.ExtraProperty C_outflow[Medium.nC];
end FluidPort;
```

`stream` variables are the properties that are carried by the flow.

Thermofluid components have one balance equation for each outflowing stream:

```
port_a.m_flow * (inStream(port_a.h_outflow) - port_b.h_outflow) = -Q_flow;
port_a.m_flow * (inStream(port_b.h_outflow) - port_a.h_outflow) = +Q_flow;
```

R. Franke, F. Casella, M. Otter, M. Sielemann, H. Elmqvist, S. E. Mattsson, and H. Olsson.
Stream connectors – an extension of modelica for device-oriented modeling of convective transport phenomena.
In F. Casella, editor, Proc. of the 7-th International Modelica Conference, Como, Italy, Sept. 2009.

?