Modelica Buildings Library

Michael Wetter and Thierry S. Nouidui
Simulation Research Group

July 7, 2015

BERKELEY LAB
Lawrence Berkeley National Laboratory

Overview

Intended use

Users

- Equipment manufacturers, design firms, academia.
- Engine for “Spawn of EnergyPlus” HVAC and controls
- Model-based design process (e.g., FLEXLAB).
- FDD algorithms (ongoing for DoD).
- Simulation engine for http://www.learnhvac.org and http://www.learngreenbuildings.org

License

- All development is open-source under Modelica 2.0 license (similar than BSD).

Objectives

For Spawn of EnergyPlus

- modular models for controls and HVAC

For building designers and manufacturers

- open-source, free library of component and system models
- collection of case studies and demonstrations

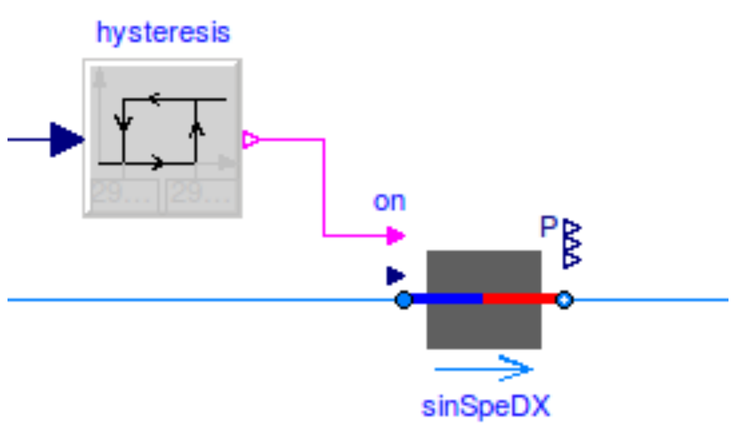
For researchers and manufacturers

- library and tools for rapid virtual prototyping and model-based design

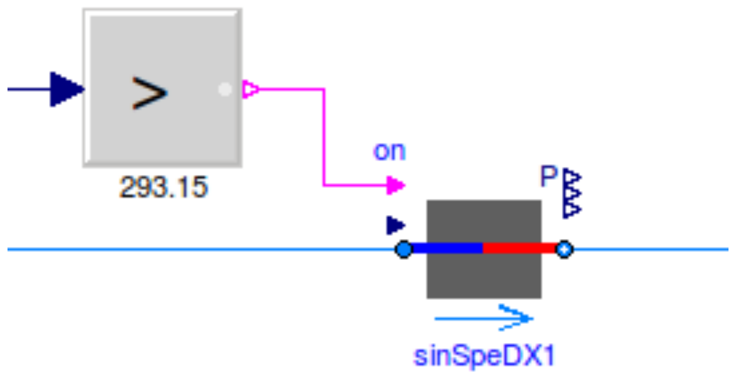
For simulation tool developers

- collaborative environment
- software components with liberal open-source license, vetted by experts from around the world

Correct configuration



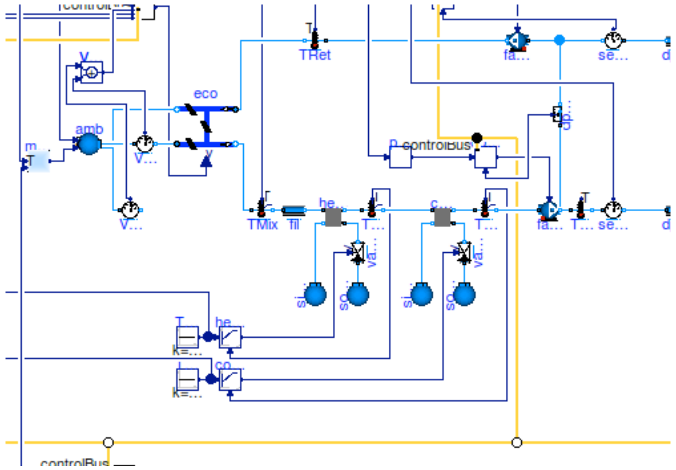
Avoid this configuration



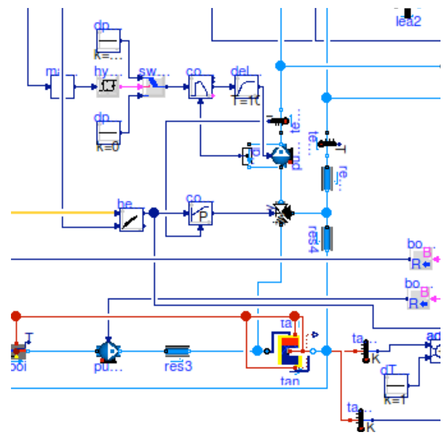
User guide with best practice.

Scope

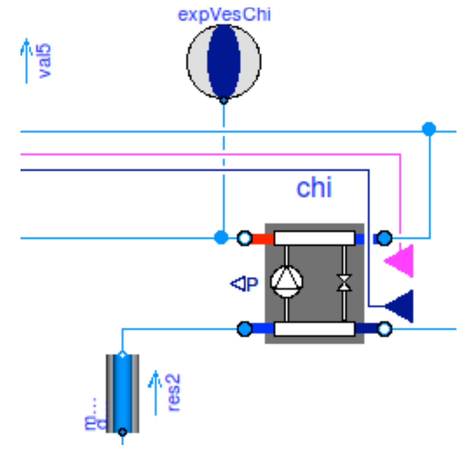
Air-based HVAC



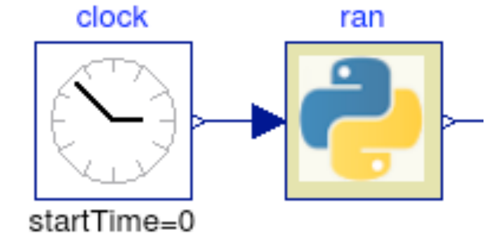
Hydronic heating



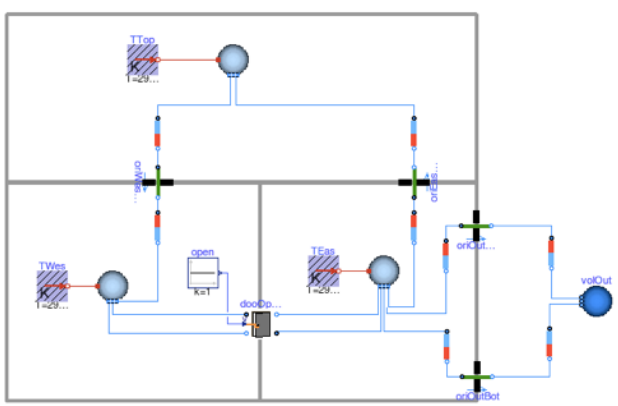
Chiller plants



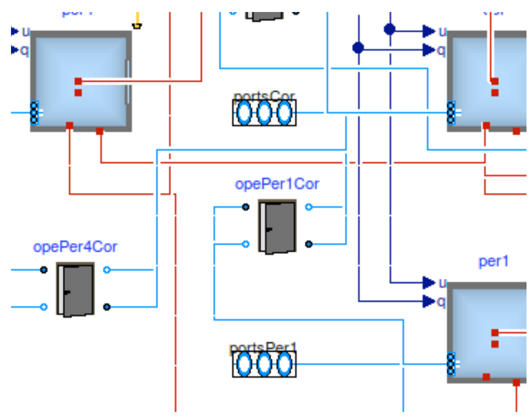
Embedded Python



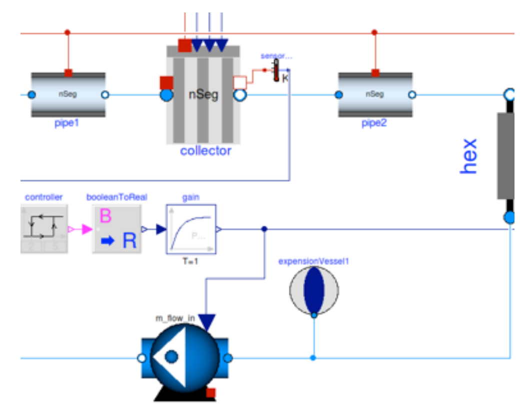
Natural ventilation,
multizone air exchange,
contaminant transport



Room heat transfer,
incl. window (TARCOG)

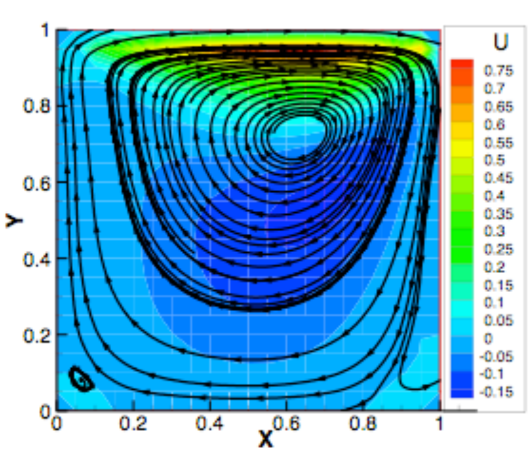


Solar collectors

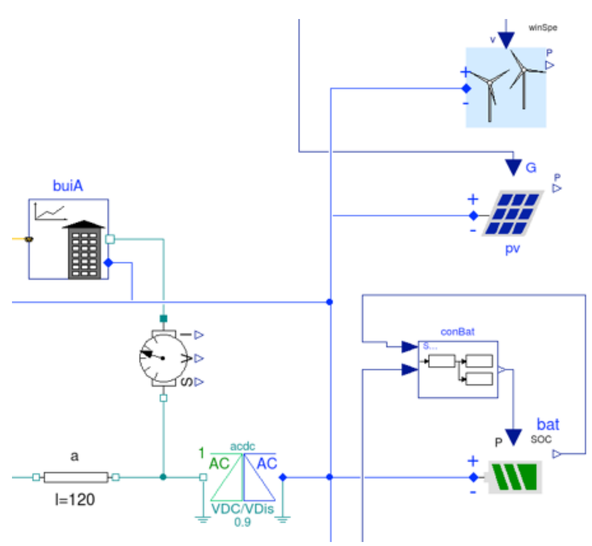


FLEXLAB

Room air flow



Electrical systems

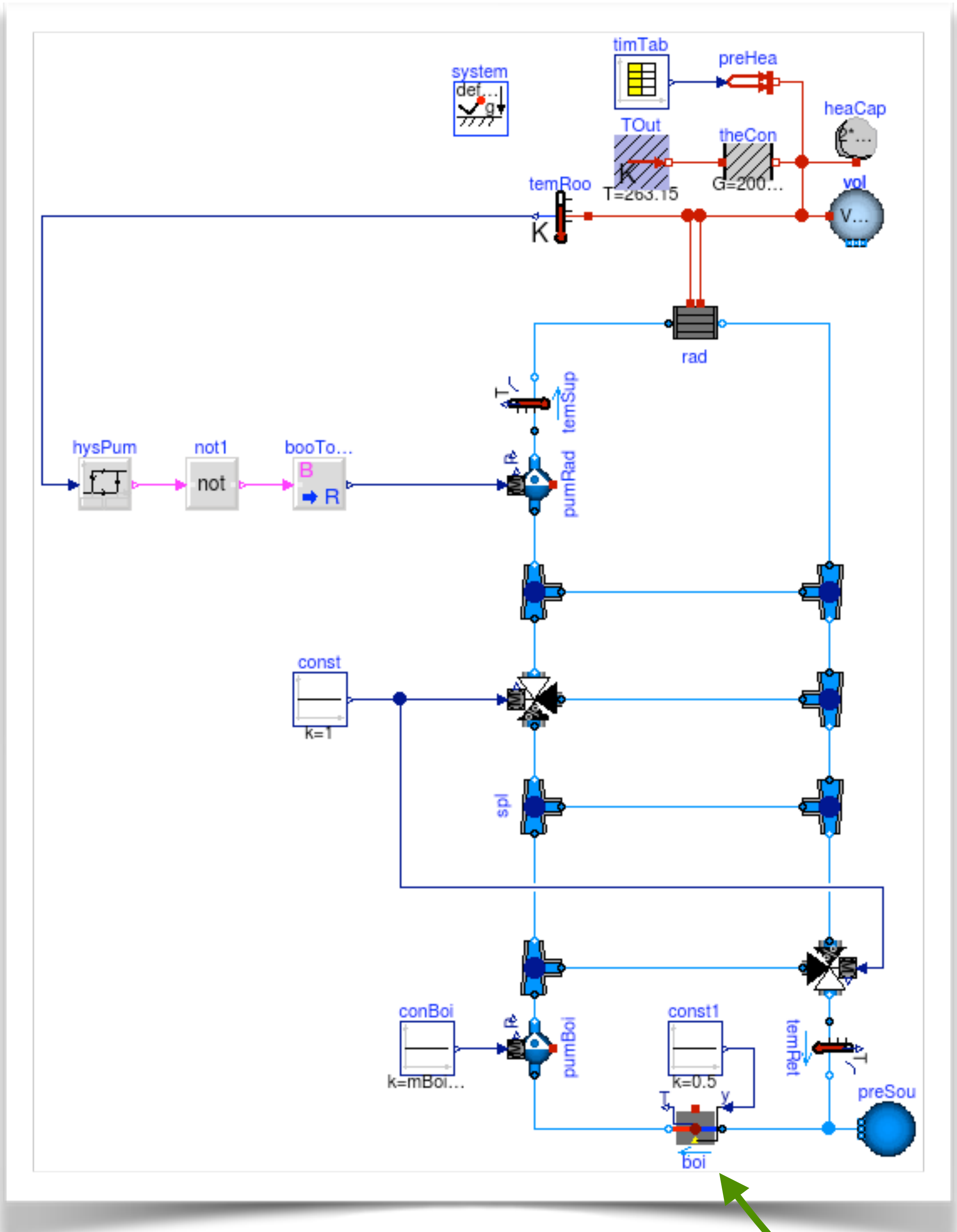


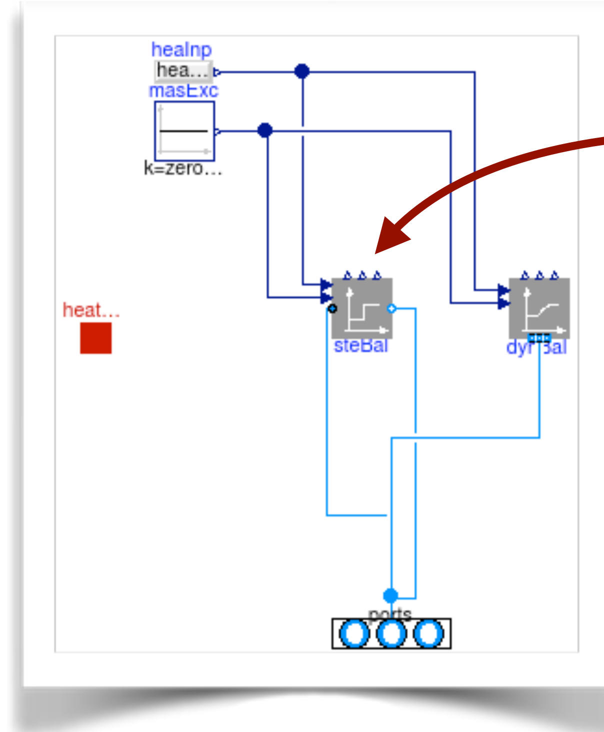
500+ validated component models.

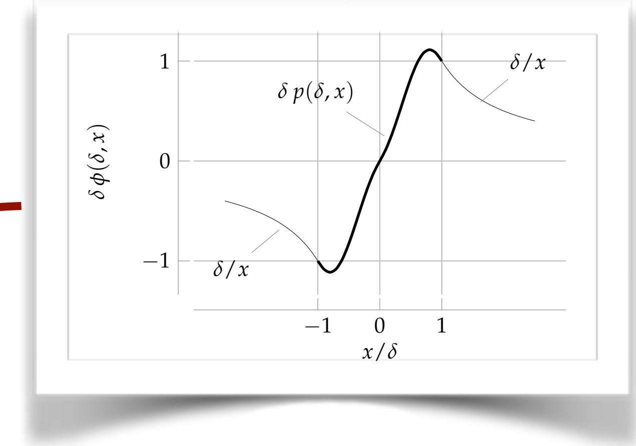
Free, open-source.

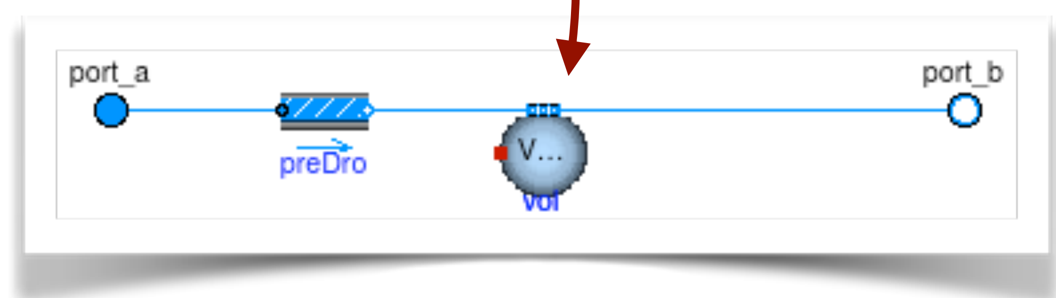
http://simulationresearch.lbl.gov/modelica

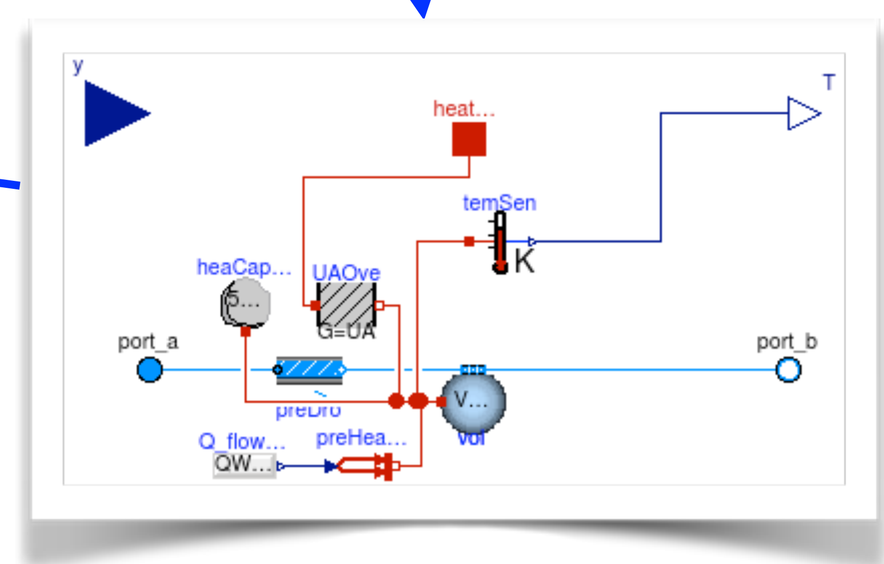
Example application

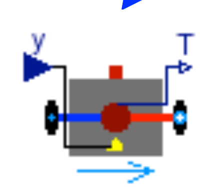












Legend:

- Library developer
- Component developer
- End user

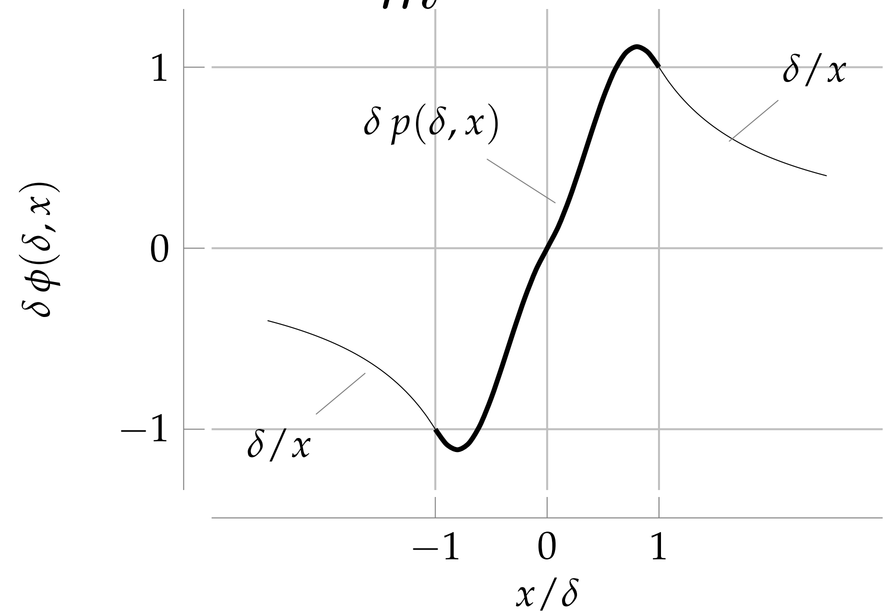
Main modeling assumptions

Media	Can track moisture (X) and contaminants (C).
HVAC equipment	Most equipment based on performance curve, or based on nominal conditions and similarity laws. Refrigerant is not modeled. Most equipment optional steady-state or 1st order transient.
Flow resistances	Based on m_flow_nominal and dp_nominal plus similarity law. Optional flag to linearize or to set dp=0.
Room model	Any number of constructions are possible. Layer-by-layer window model (similar to Window 6). Optional flag to linearize radiation and/or convection.
Electrical systems	DC. AC 1-phase and 3-phase (dq, dq0). Quasi-stationary or dynamic phase angle (but not frequency).

Special modeling approach

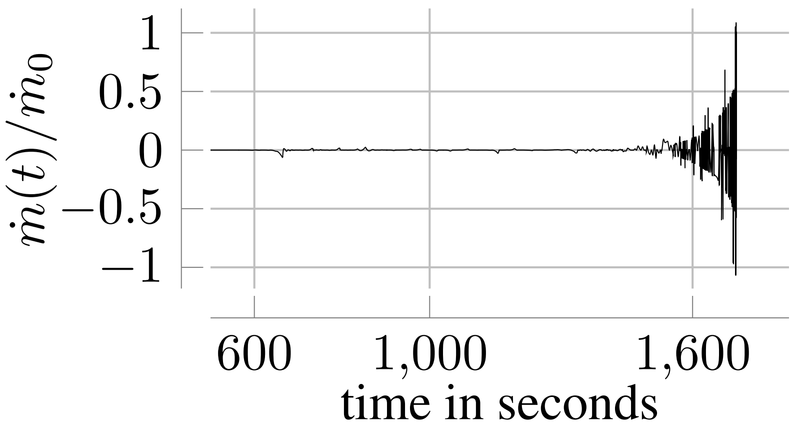
All equations of physical systems are once continuously differentiable.

$$\Delta h = \frac{\dot Q}{\dot m} \approx \dot Q \, \phi(\dot m_\delta, \dot m)$$



Special treatments to avoid numerical problems if m_flow is in neighborhood around 0.

Fan/pump model for which we can prove existence of unique solution.
See paper at Building Simulation 2013.



Validation

Room model: ANSI/ASHRAE 140.

Window model: Window 6 plus full-scale experiments.

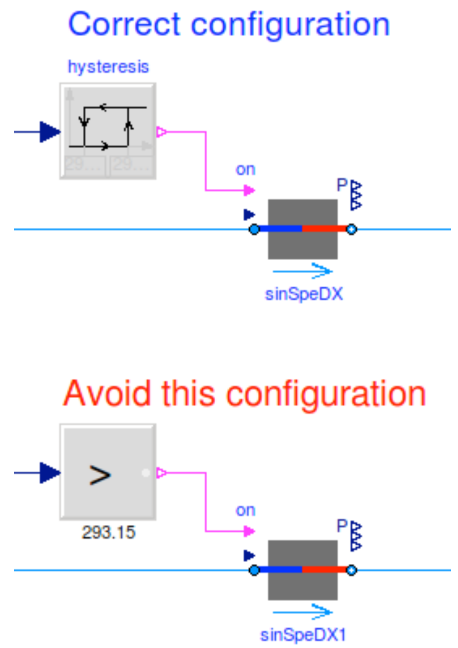
Comparative model validation for

- Window model (Window 6)
- Multizone air exchange (CONTAM)
- DX coils (EnergyPlus)
- Solar collectors (TRNSYS)

Where possible, all components were verified with analytical solutions.

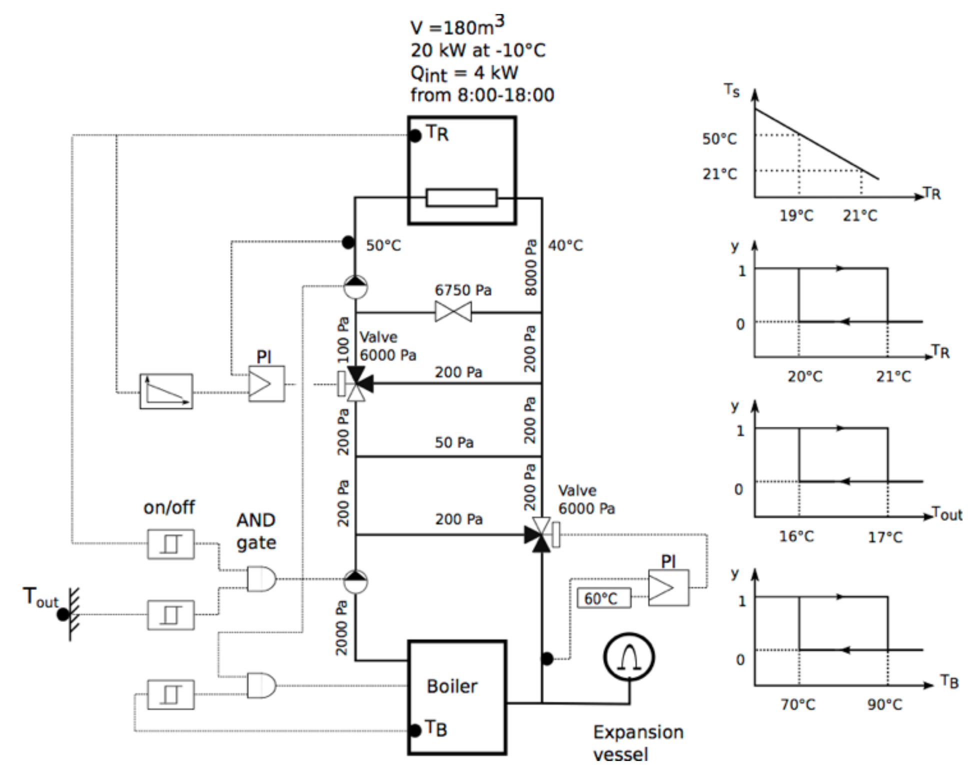
500+ regression tests compare results to reference results as part of development,
see https://github.com/lbl-srg/modelica-buildings/wiki/Unit-Tests

Documentation and distribution



Documentation

- General <u>user guide</u> (getting started, best practice, developer instructions, ...).
- 14 <u>user guides</u> for individual packages.
- 2 <u>tutorials</u> with step-by-step instructions.
- All models contain “info” section.
- Small test models for all classes, large test cases for “smoke tests,” and various validation cases.



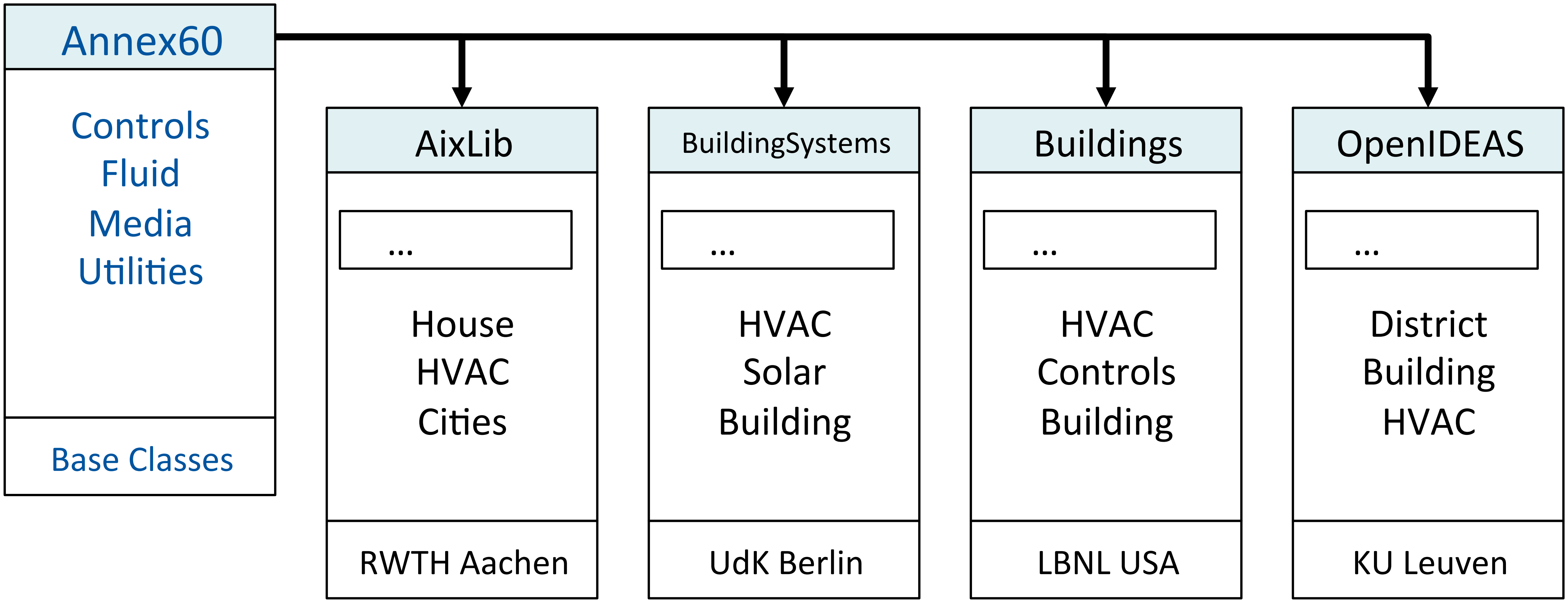
Distribution

- Main site
  http://simulationresearch.lbl.gov/modelica
- Development site with version control, wiki and issue tracker:
  https://github.com/lbl-srg/modelica-buildings

Collaborative development within IEA EBC Annex 60

Goal of activity 1.1 (library development):

Develop and distribute a well documented, vetted and validated open-source Modelica library that serves as the core of future building simulation programs.

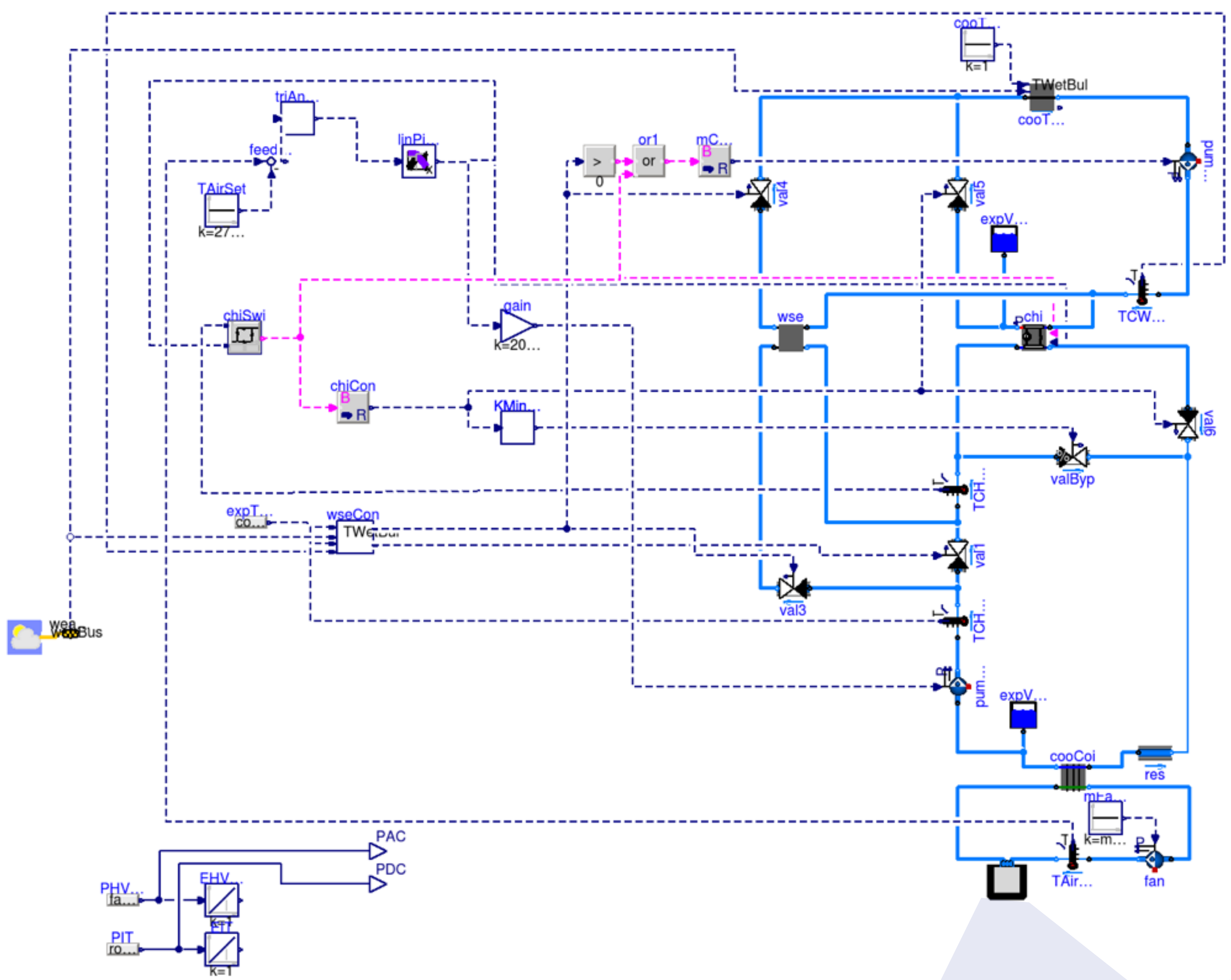


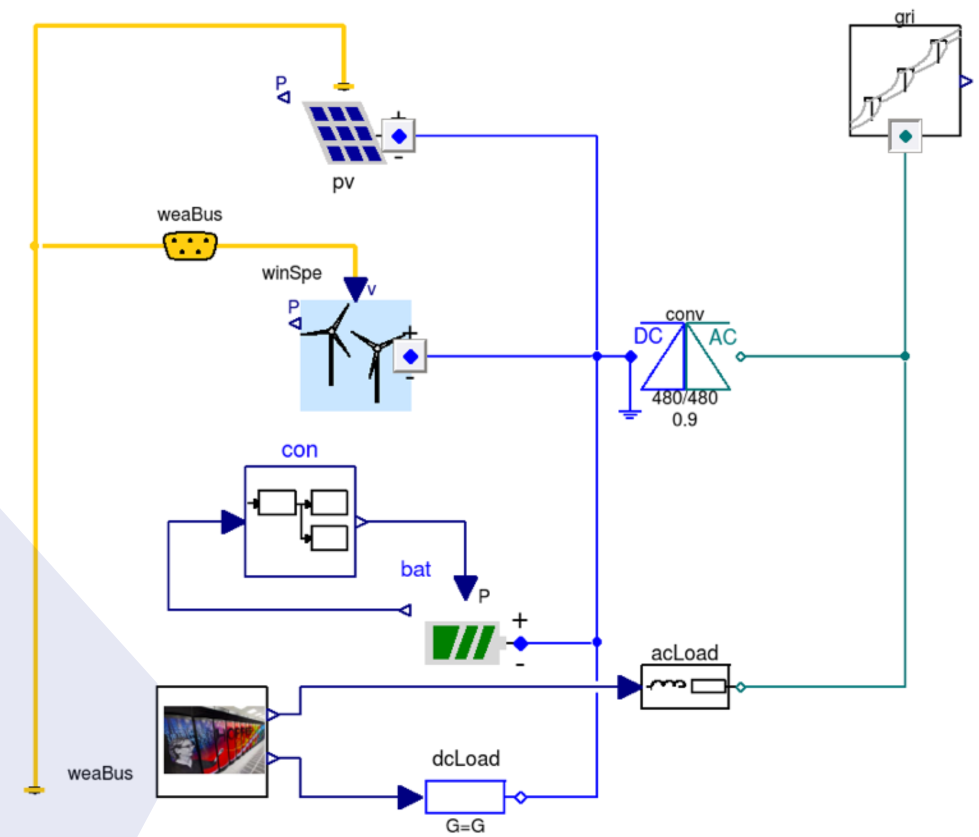
Development at https://github.com/iea-annex60/modelica-annex60

Example Applications

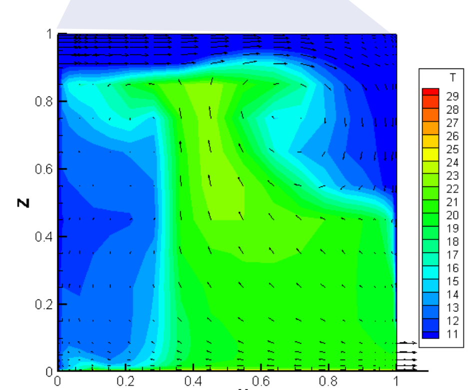
Virtual prototyping through graphical modeling of multi-physics systems

Feedback control   Cooling plant



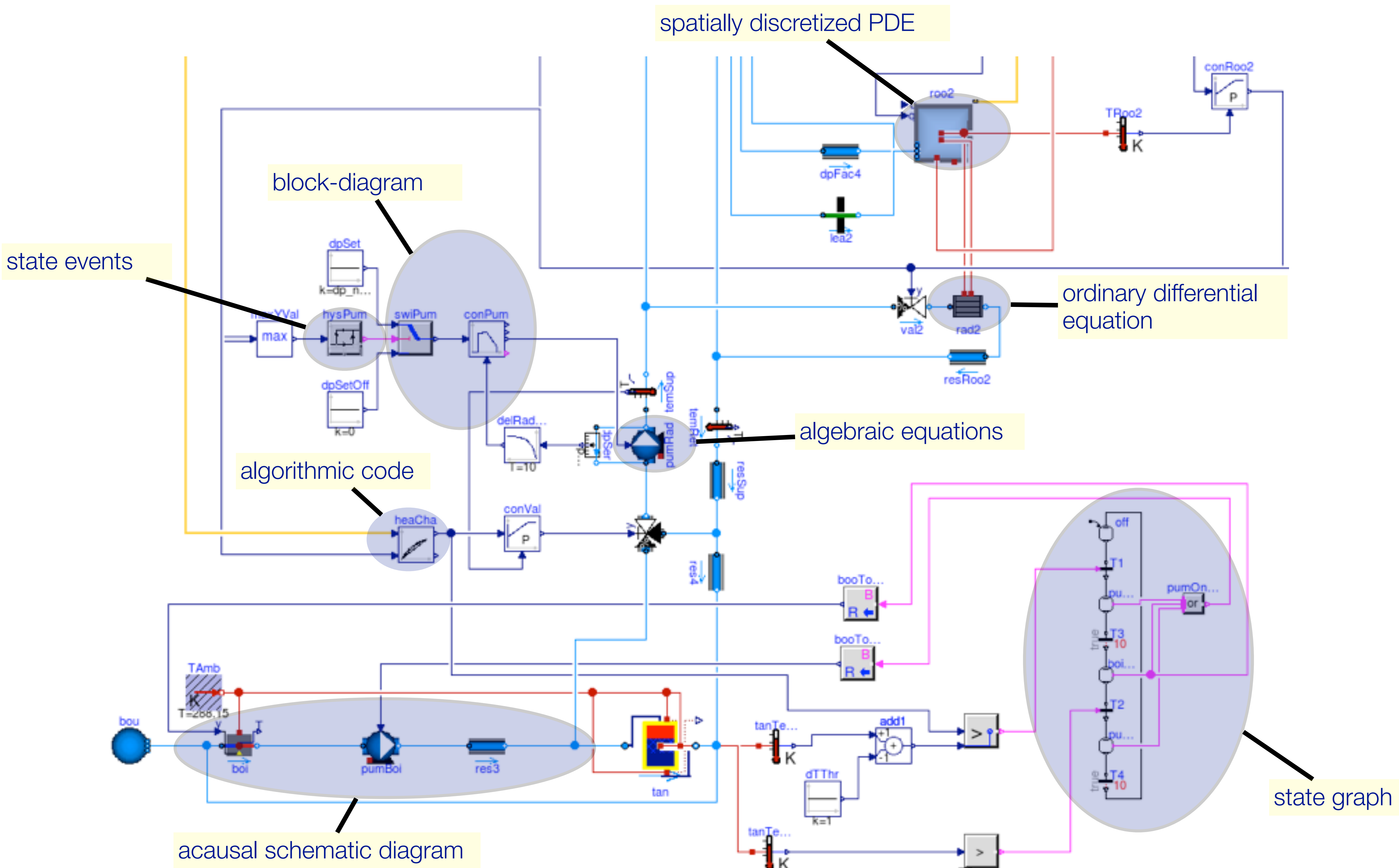


DC renewable plant   AC grid



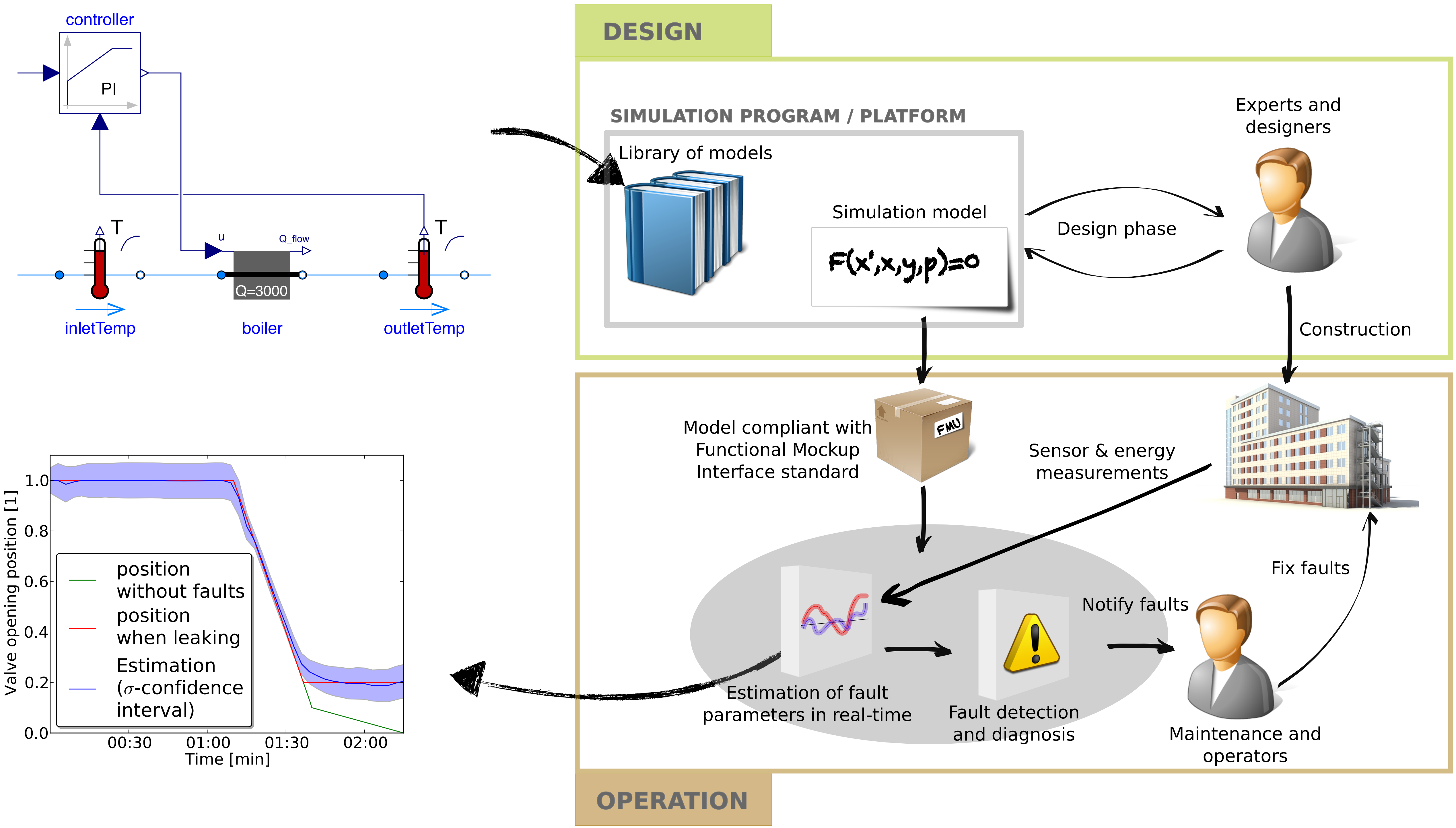
Temperature stratification based on CFD

Modeling of hydronic systems

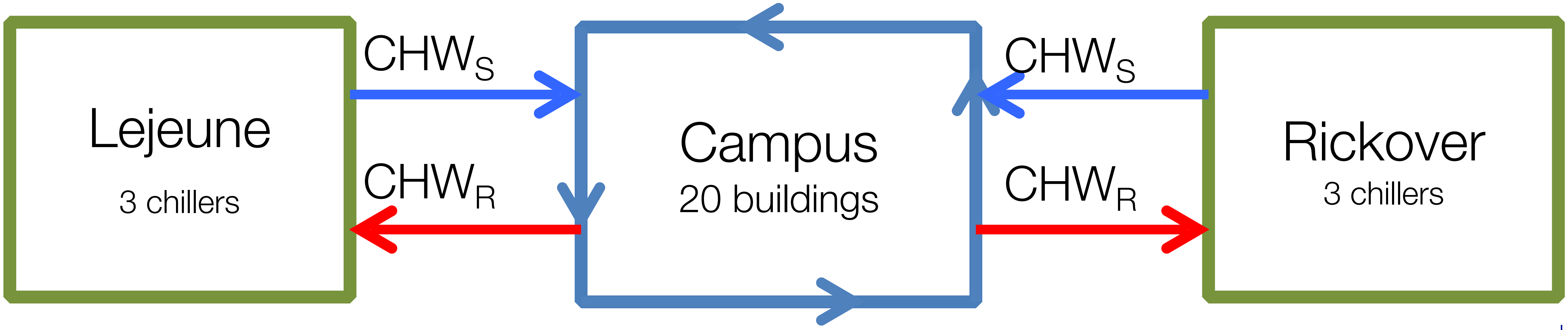


The above model is composed hierarchically, partially through automatic generation, of 1700 component models.

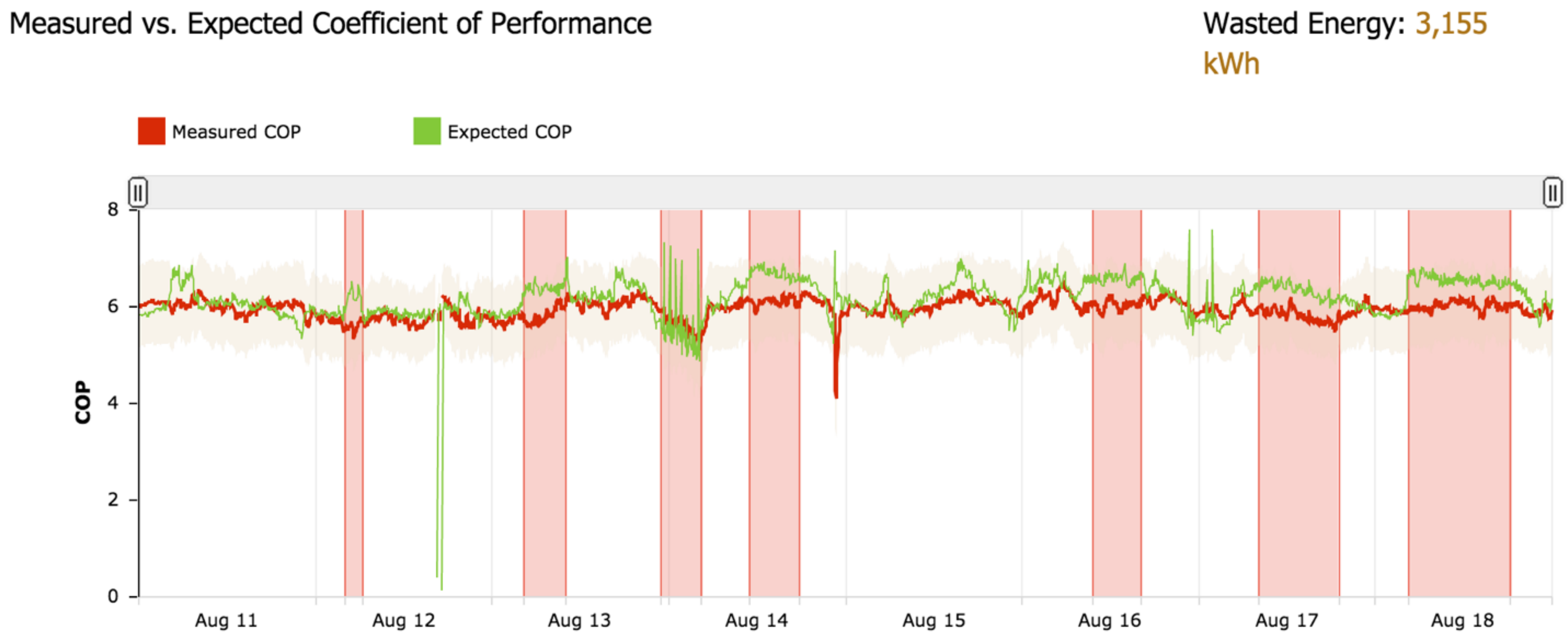
Fault detection based on design models under consideration of uncertainties



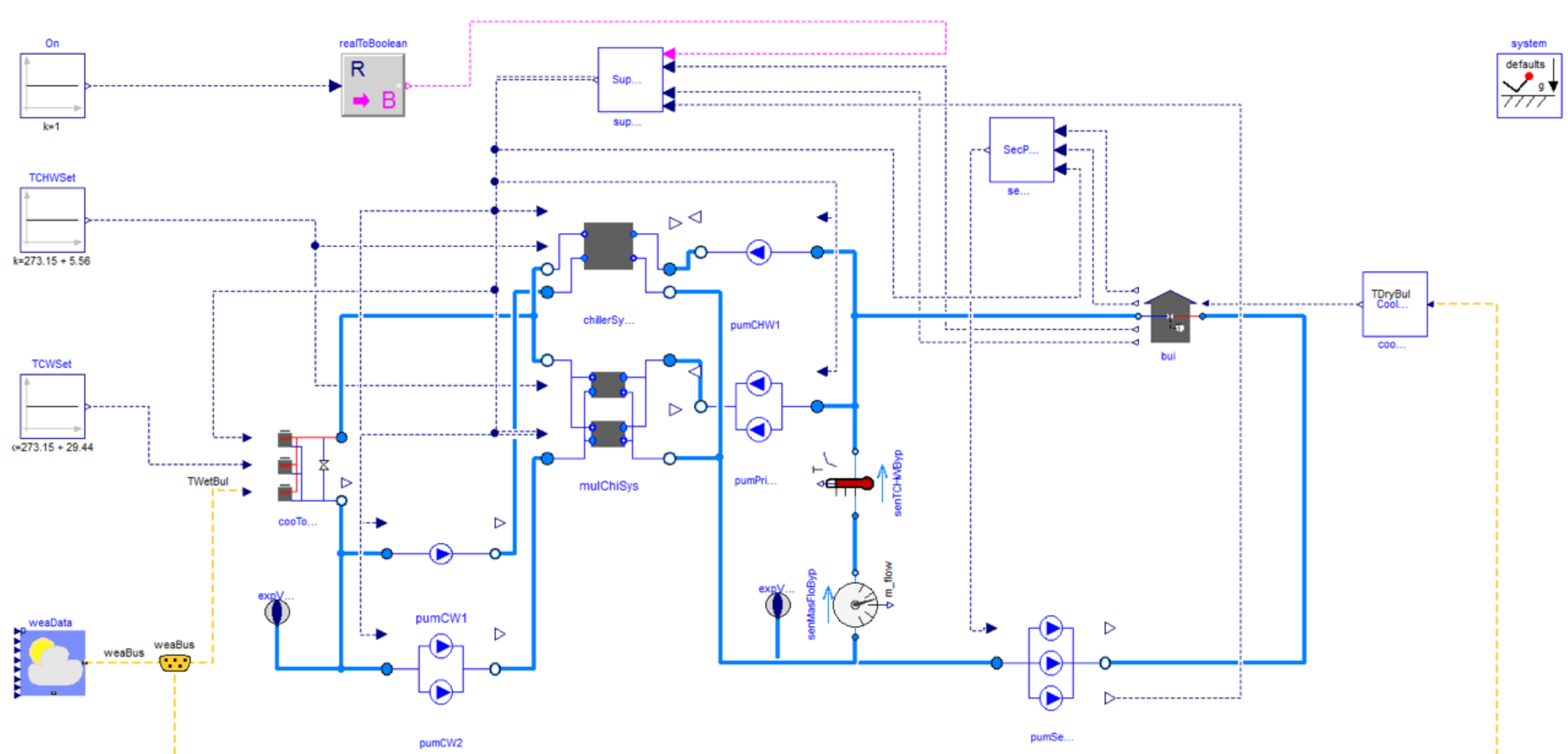
FDD applied to district chilled water plant



FDD applied to district chilled water plant



User-interface



Underlying analysis model

Structure of the library

Organization of the main packages

```
Buildings
  Airflow
    Multizone
  BoundaryConditions
    SolarGeometry
    SolarIrradiation
    SkyTemperature
    WeatherData
  Controls
    Continuous
    DemandResponse
    Discrete
    Predictors
    SetPoints
  Electrical
    {AC, DC}
  Fluid
    Actuators
    Boilers
    Chillers
    FixedResistances
    HeatExchangers
    …
  HeatTransfer
    Conduction
    Convection
    Radiosity
    Windows
  Rooms
    CFD
    MixedAir
  Utilities
    Comfort
    Math
    Psychrometrics

  Resources
    C-Sources
    Data
    Documentation
    Include
    Library
    ReferenceResults
    Scripts
    bin
    src
    weatherdata
```

Organization of individual packages

Packages are typically structured as shown on the right.

To add a new class, look first at `Interfaces` and `BaseClasses`.

You probably will never implement a component without extending a base class, such as from `Buildings.Fluid.Interfaces`

```
Tutorial
UsersGuide

Any other classes (models,
functions etc.)

Data
Types
Examples
Validation
Benchmarks
Experimental
Interfaces
BaseClasses
Internal
Obsolete
```

Best practice and modeling hints

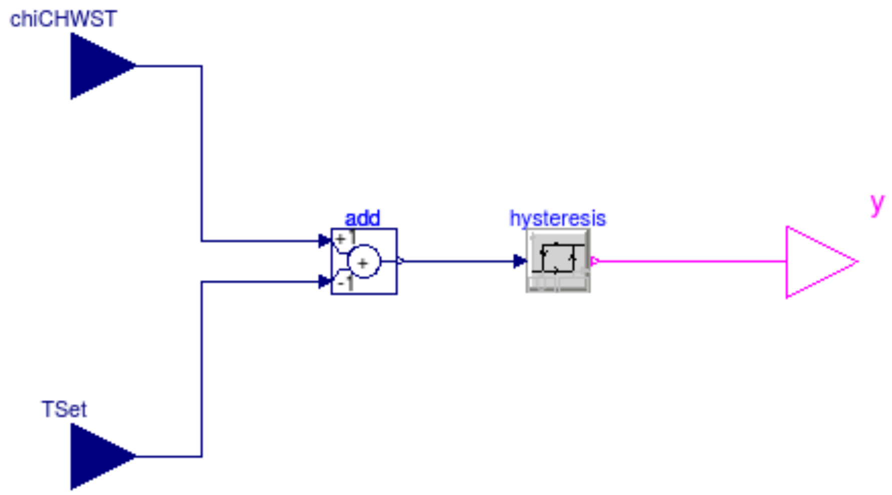
Building large system models

How do you build and debug a large system model?

1. Split the model into smaller models.
2. Test the smaller models for well known conditions.
3. Add smaller models to unit tests.

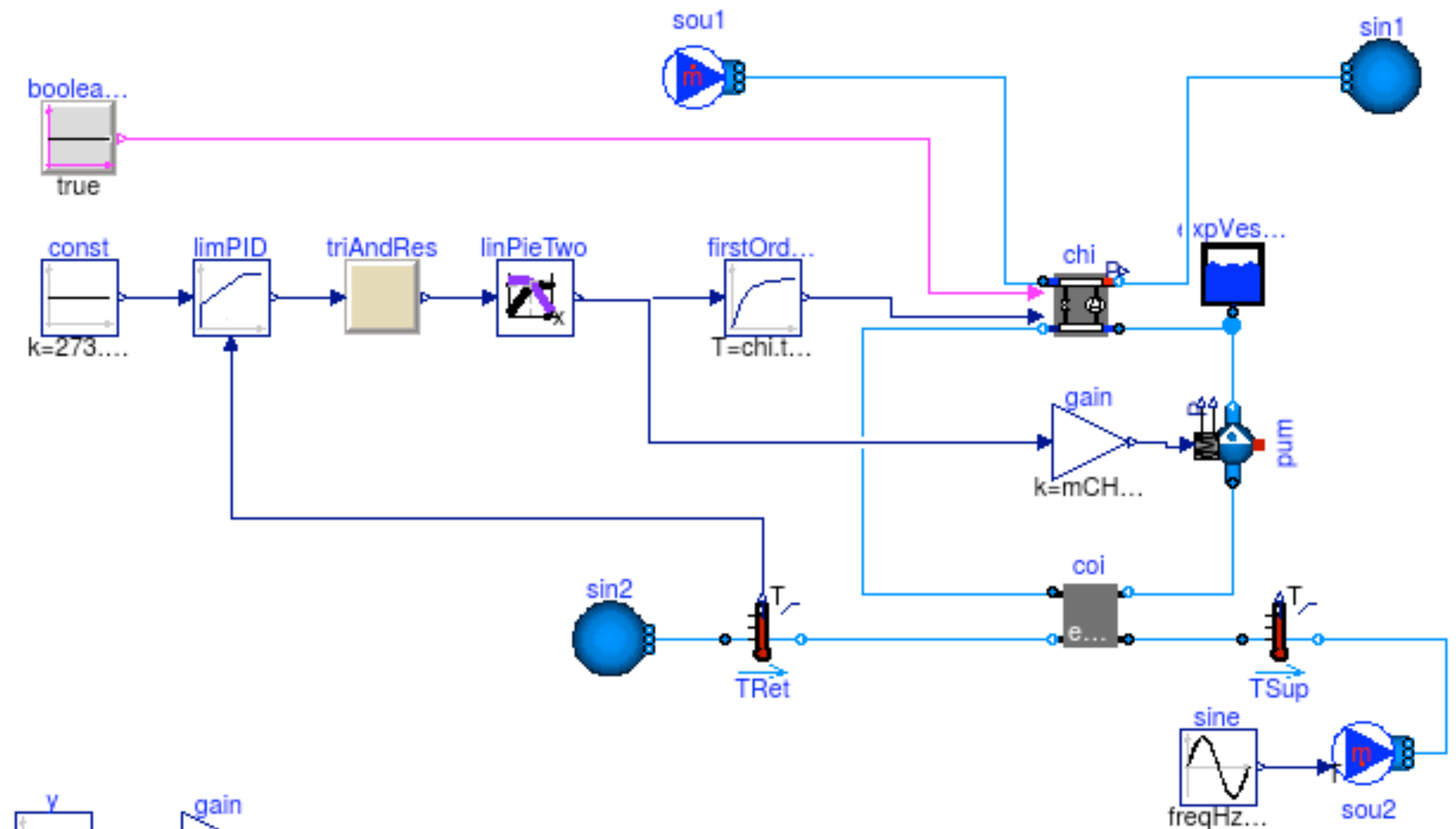
For example, see Chiller Plant

Each small models contains a simple unit test.

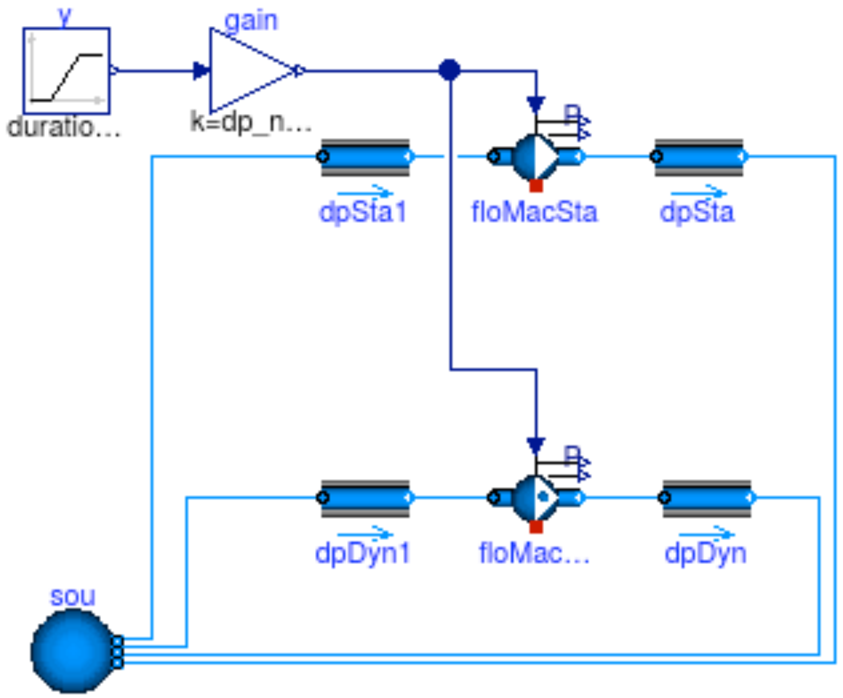


Use small unit tests, as in

[Chiller plant base classes](#)



[Pumps](#)



Propagate common parameters

Don't assign values to the same parameters

```
Pump pum(m_flow_nominal=0.1) "Pump";
TemperatureSensor sen(m_flow_nominal=0.1) "Sensor";
```

Instead, propagate parameters

```
Modelica.SIunits.MassFlowRate m_flow_nominal = 0.1
  "Nominal mass flow rate";
Pump pum(final m_flow_nominal=m_flow_nominal) "Pump";
TemperatureSensor sen(final m_flow_nominal=m_flow_nominal) "Sensor";
```

Assignments can include computations, such as

```
Modelica.SIunits.HeatFlowRate QHea_nominal = 3000
  "Nominal heating power";
Modelica.SIunits.TemperatureDifference dT = 10
  "Nominal temperature difference";
Modelica.SIunits.MassFlowRate m_flow_nominal = QHea_nominal/dT/4200
  "Nominal mass flow rate";
...
```

Always define the media at the top-level

Top-level system-model

```
replaceable package Medium = Buildings.Media.Air
  "Medium model";
```

Propagate medium to instance of model

```
TemperatureSensor sen(
  redeclare final package Medium = Medium,
  final m_flow_nominal=m_flow_nominal) "Sensor";
```

Note: For arrays of parameters, use the **each** keyword, as in

```
TemperatureSensor sen[2](
  each final m_flow_nominal=m_flow_nominal)
"Sensor";
```

Exercise 1: Propagate parameters and media

Common parameters are design flow rates for

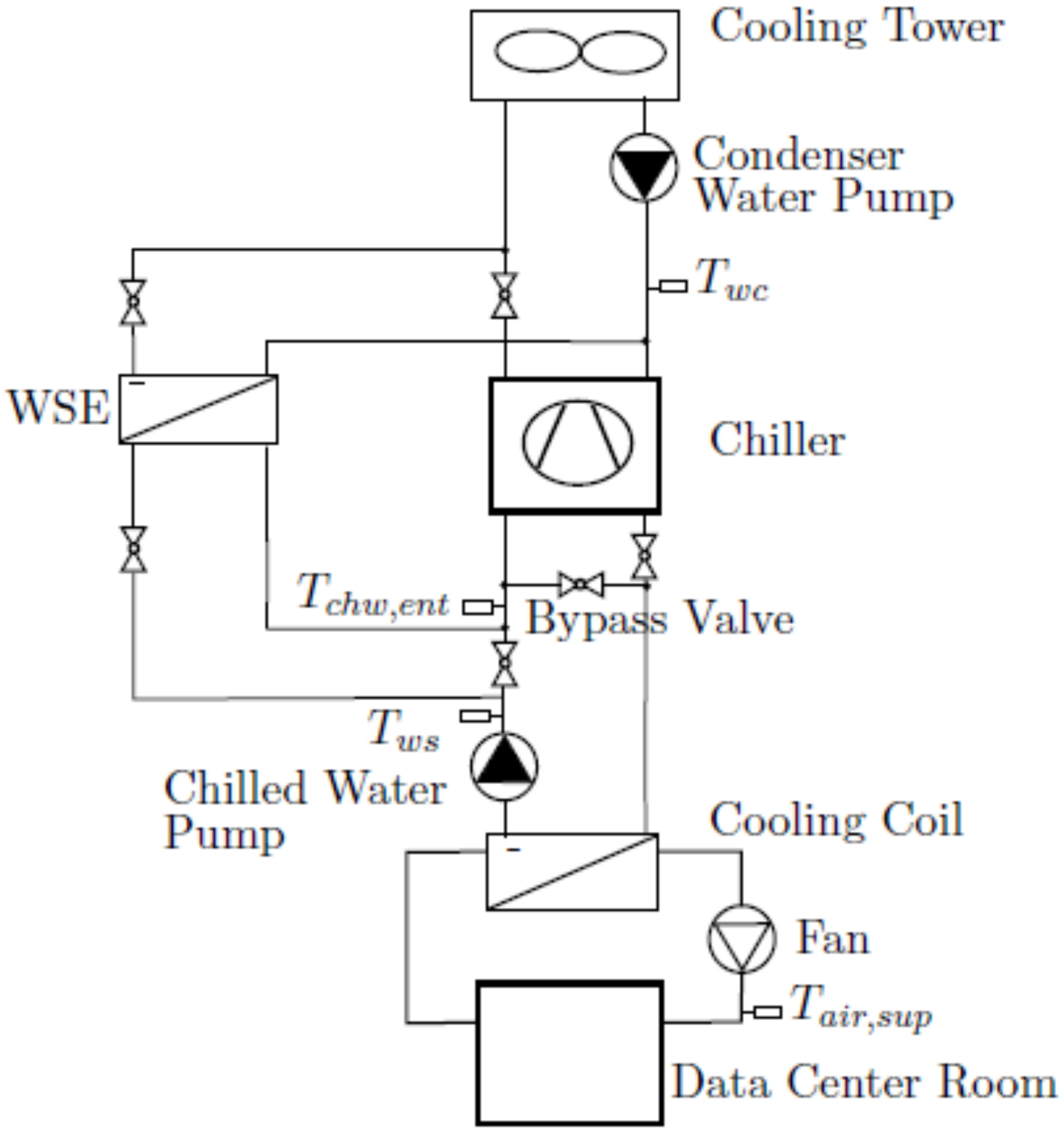
- air,
- water and
- condenser loop.

Common media are air and water for chiller and condenser loop.

Find where common parameters are used and propagated in Buildings.Examples.ChillerPlant.DataCenterDiscreteTimeControl

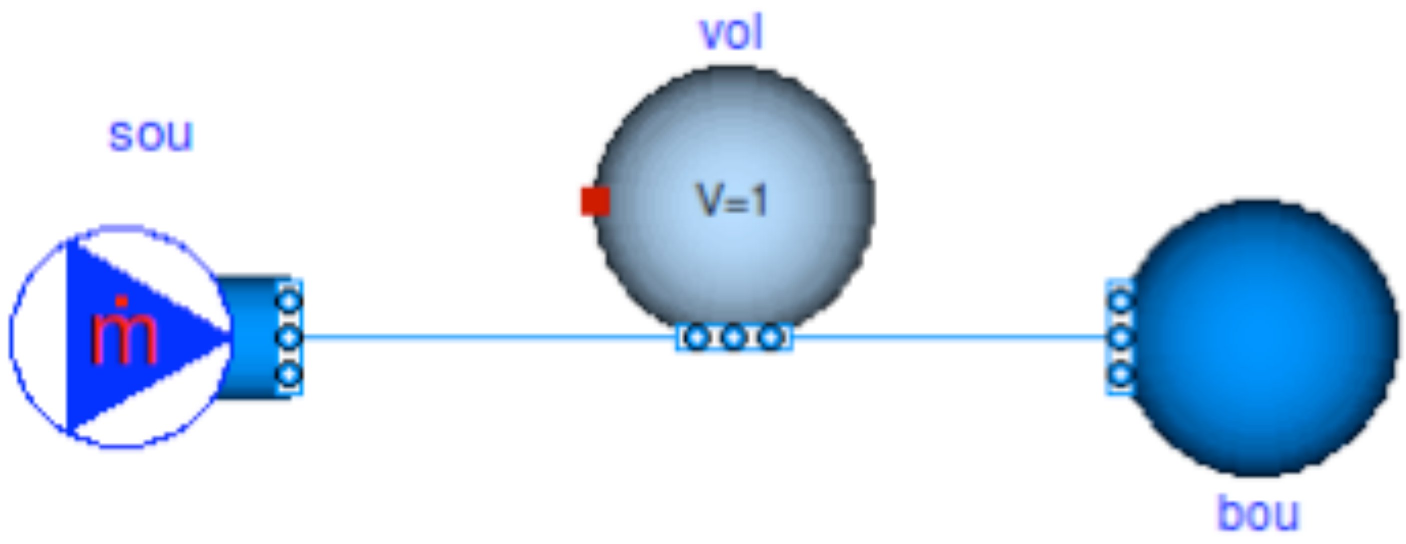
Find where air mass flow rate in air/water heater exchanger is propagated.

(Hint: In Dymola, you can expand all inherited classes.)



Exercise 2: Modeling of a simple thermofluid flow system

How do you implement a source and boundary condition with a tank in between to create the model below:

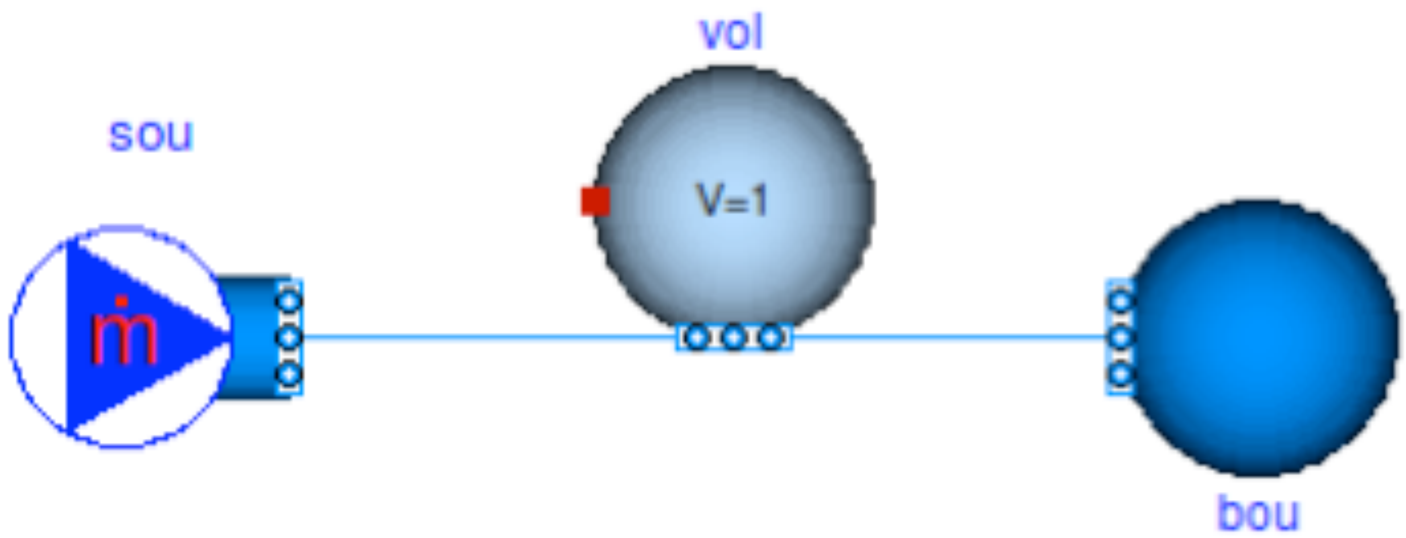


Exercise 2: Modeling of a simple thermofluid flow system

1. Make instances using models from Buildings.Fluid.Sources and Buildings.Fluid.MixingVolumes.

2. Assign the parameters.

3. Check and simulate the model.



Further resources

Tutorials

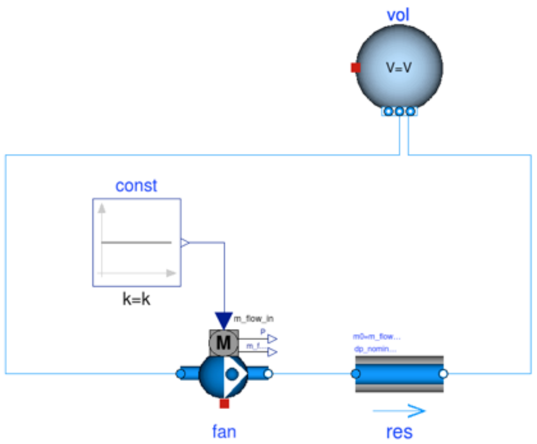
- Buildings.Examples.Tutorial

User guides

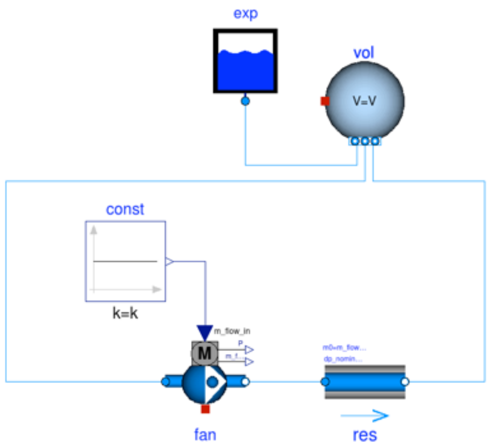
- User guides for specific packages of models.
- User guide with general information.

Setting a reference pressure

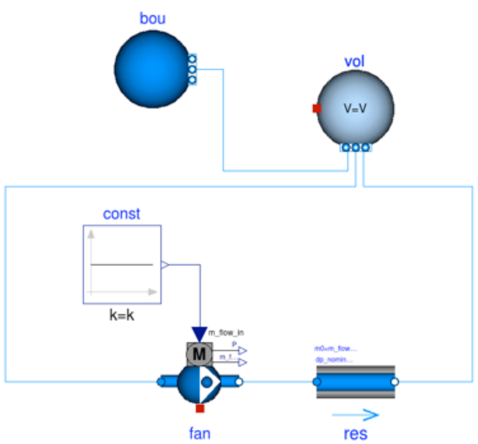
Underdetermined model as no pressure is assigned



Well defined model, but additional state for pressure as reservoir *p/p0=V0/p*

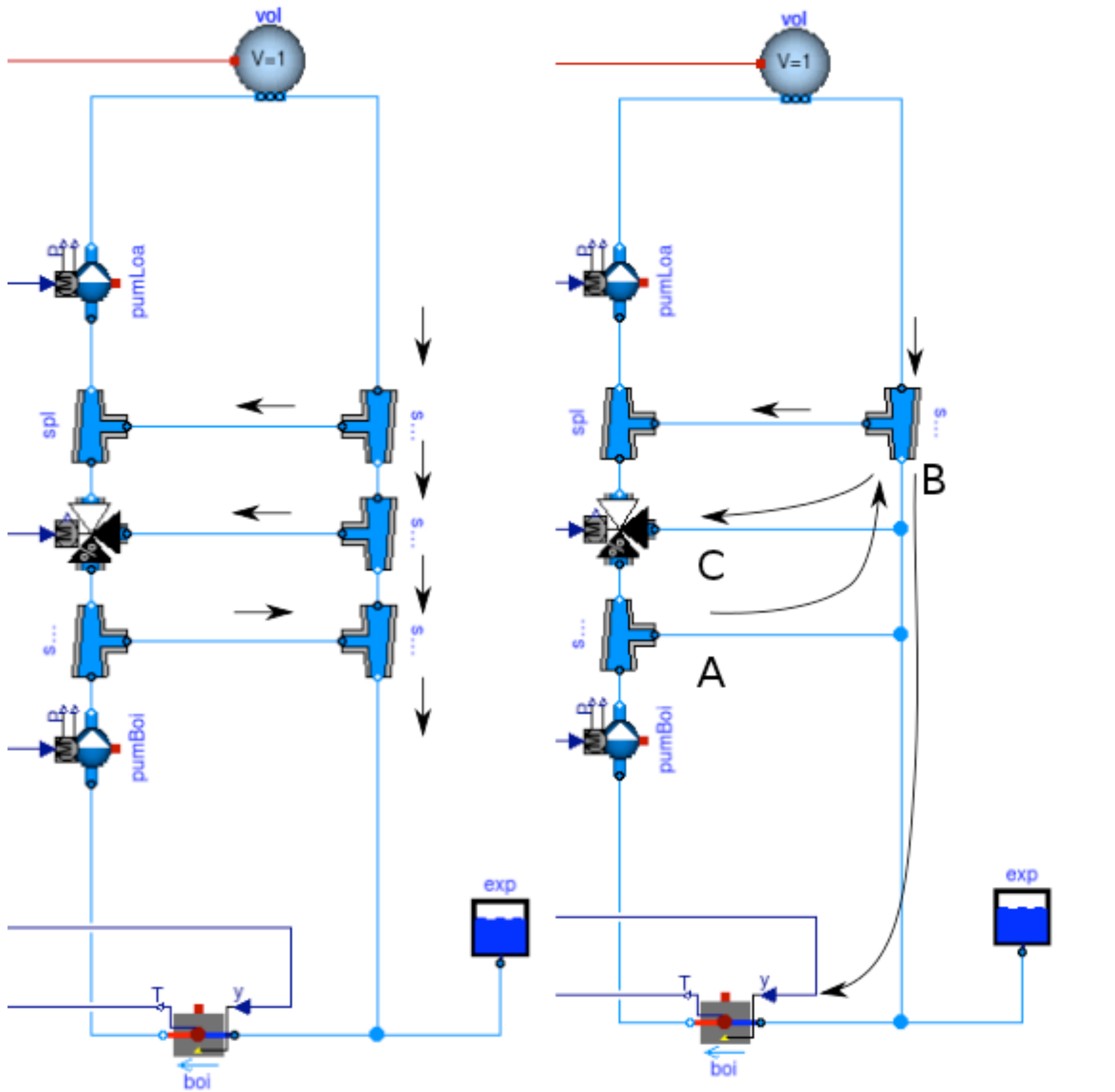


Most efficient model as reservoir *p* is constant



Modeling of fluid junctions

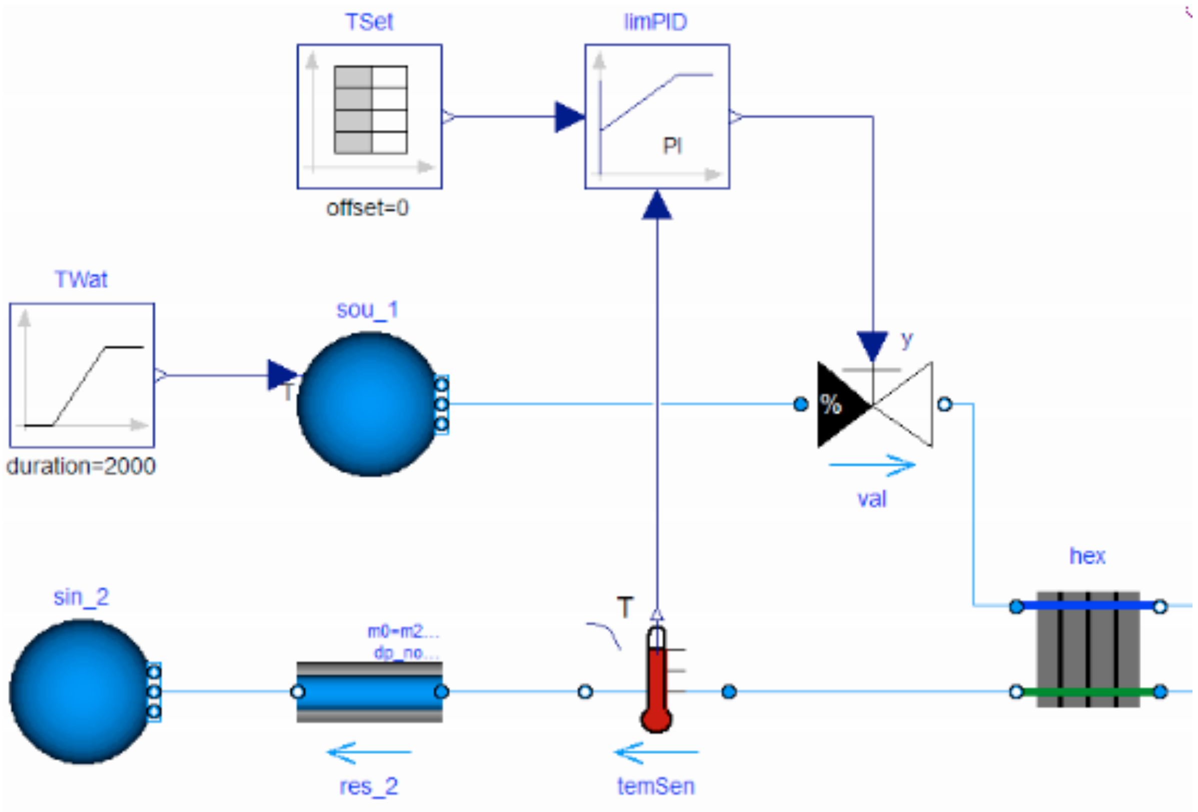
In the model on the right, mixing takes place in the fluid port B because the boiler, port A and port C all connect to port B.



Avoid oscillations of sensor signal

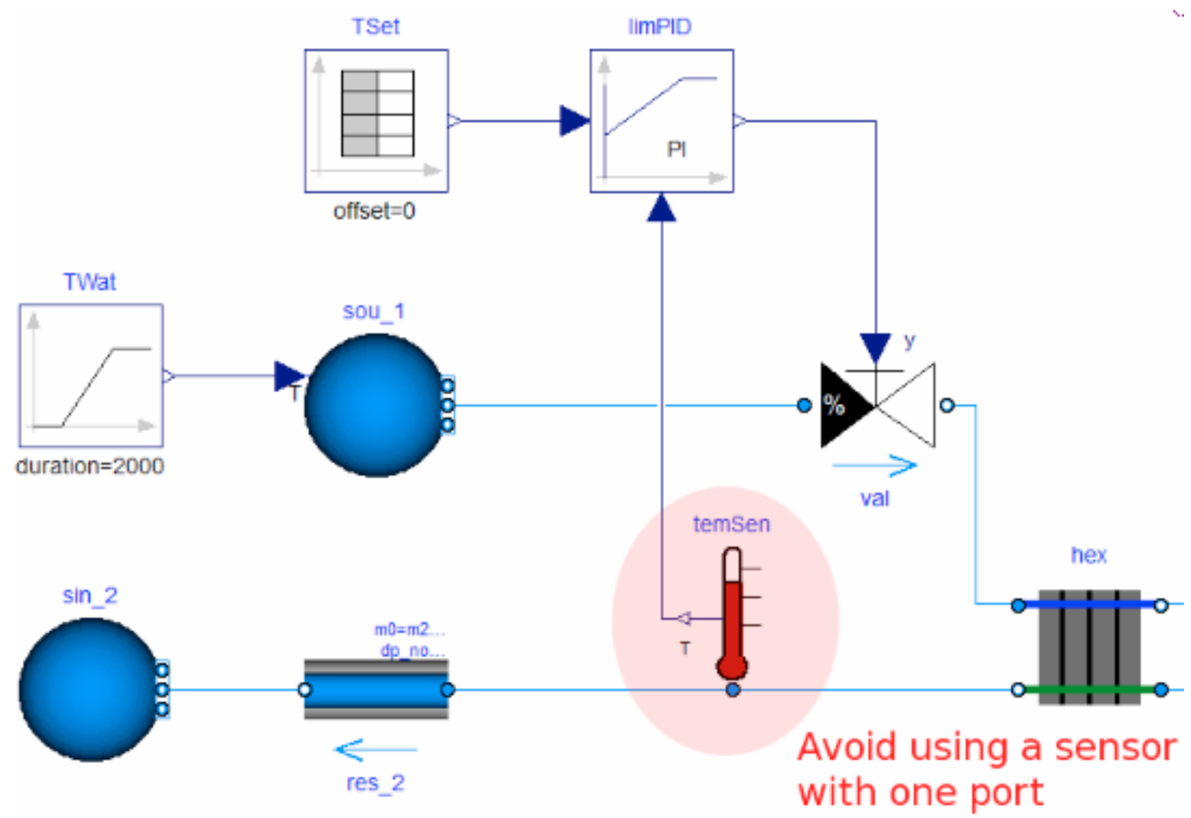
Correct use because

$$\tau \frac{dT}{dt} = \frac{|\dot{m}|}{\dot{m}_0} (\theta - T)$$



Incorrect, as sensor output oscillates if mass flow rate changes sign.
This happens for example if the mass flow rate is near zero and approximated by a solver.

See also User Guide.



Avoid events

This triggers events:

```
T_in = if port_a.m_flow > 0  then port_a.T else port_b.T;
```

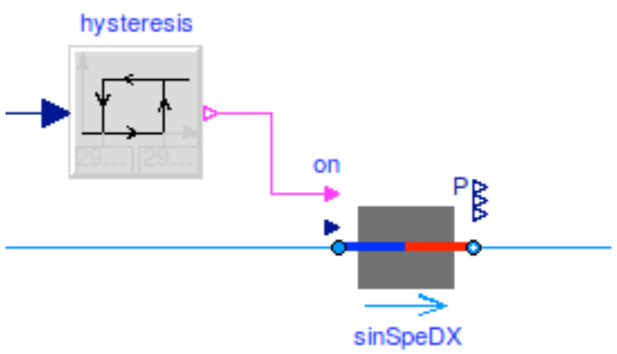
Avoid events using regularization:

```
T = Modelica.Fluid.Utilities.regStep(
      x = port_a.m_flow,
      y1 = T_a_inflow,
      y2 = T_b_inflow,
      x_small = m_flow_nominal*1E-4);
```

See also User Guide.

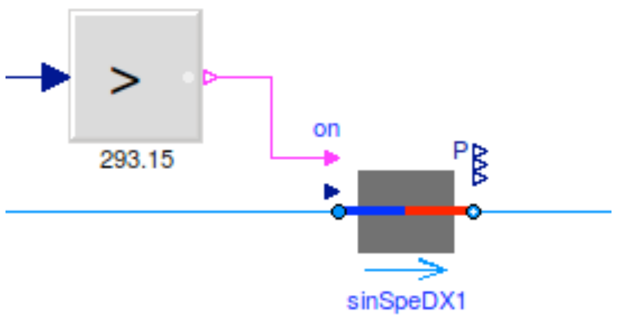
Beware of oscillating control

Correct configuration



If the control input oscillates around zero, then this model stalls

Avoid this configuration



What happens if this model is simulated with an adaptive time step?

```
model Test
  Real x(start=0.1);
equation
  der(x) = if x > 0 then -1 else 1;
end Test;
```

Setting of nominal values is important for scaling of residuals

If pressure is around 1E5 Pa, set `p(nominal=1E5)`.

In Dymola, nominal values will be used to scale the residuals, such as in `dsmodel.c`:

```
{ /* Non-linear system of equations to solve. */
  ...
const char*const varnames_[]={"floMac1.VMachine_flow",
                              "floMac2.VMachine_flow"};
const double nominal_[]={0.001, 0.001};
...
```

In Dymola, the local integration error is

$\epsilon \leq t_{rel} \, |x^i| + t_{abs}$

where the absolute tolerance is scaled with the nominal value as

$t_{abs} = t_{rel} \, |x_{nom}{}^i|.$

Development

Overview

Main topics

- Coding style and conventions
- Requirements
- Organization of the library
- Adding a new model
- Adding regression tests

Further literature

- User Guide -> Development
- Style guide
- Coding convention

Coding style and conventions

Based on Modelica Standard Library.

Most variables are 3 letter camel case to avoid too long names.

Code duplication avoided where practical.

Additional information at
https://github.com/lbl-srg/modelica-buildings/wiki/Style-Guide and
http://simulationresearch.lbl.gov/modelica/releases/latest/help/Buildings_UsersGuide.html

Requirements

Physical requirements

Mathematical requirements

Organization of individual packages

Packages are typically structured as shown on the right.

To add a new class, look first at `Interfaces` and `BaseClasses`.

You probably will never implement a component without extending a base class, such as from `Buildings.Fluid.Interfaces`

```
Tutorial
UsersGuide

Any other classes (models,
functions etc.)

Data
Types
Examples
Validation
Benchmarks
Experimental
Interfaces
BaseClasses
Internal
Obsolete
```

Implementing new thermofluid flow devices

Buildings.Fluid.Interface provides base classes.

Buildings.Fluid.Interface.UsersGuide describes these classes.

Alternatively, simple models such as the models below may be used as a starting point for implementing new models for thermofluid flow devices:

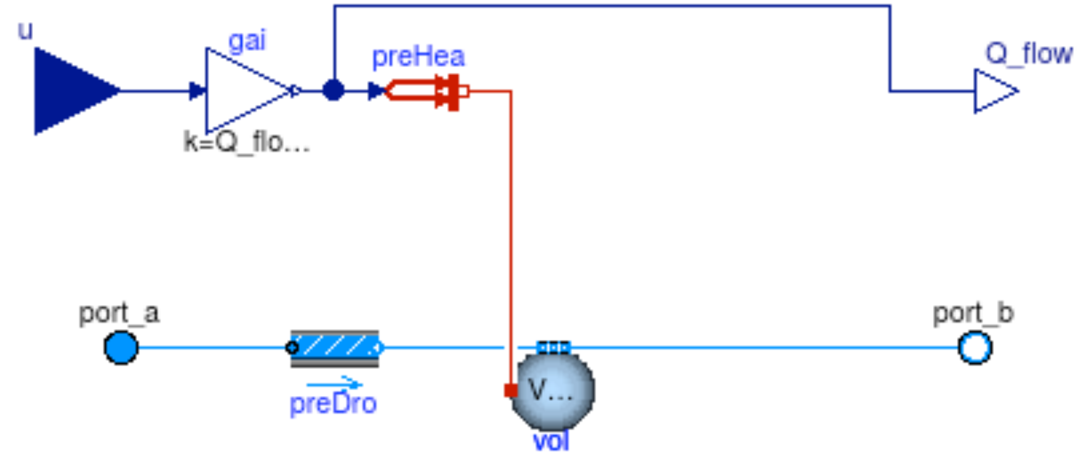
Buildings.Fluid.HeatExchangers.HeaterCooler_u
For a device that adds heat to a fluid stream.

Buildings.Fluid.MassExchangers.Humidifier_u
For a device that adds humidity to a fluid stream.

Buildings.Fluid.Chillers.Carnot
For a device that exchanges heat between two fluid streams.

Buildings.Fluid.MassExchangers.ConstantEffectiveness
For a device that exchanges heat and humidity between two fluid streams.

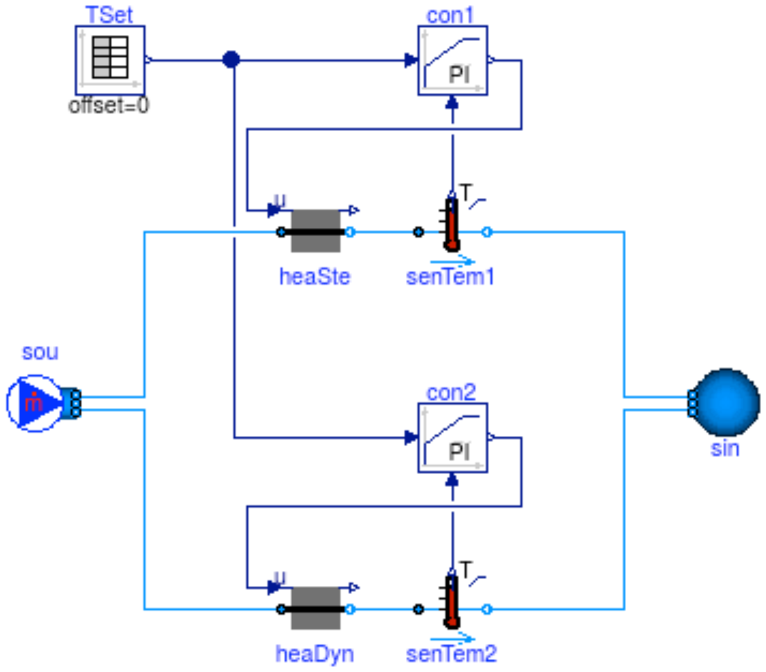
Adding a heat exchanger



See HeaterCooler_u

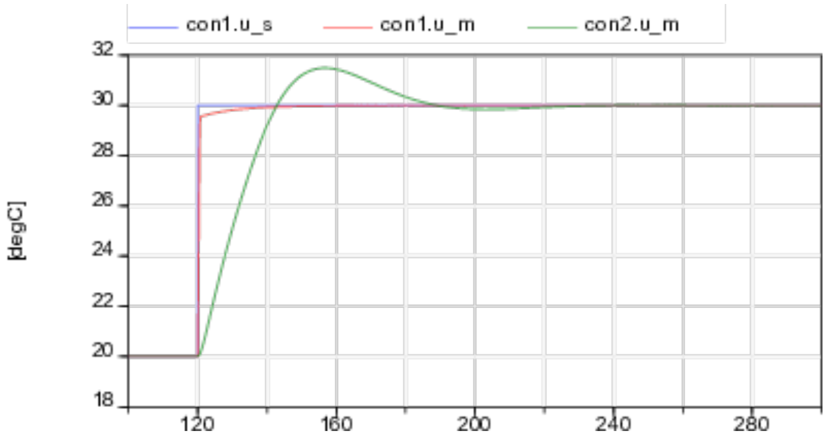
```
within Buildings.Fluid.HeatExchangers;

model HeaterCooler_u "Heater or cooler with prescribed heat flow rate"
  extends Buildings.Fluid.Interfaces.TwoPortHeatMassExchanger(
    redeclare final Buildings.Fluid.MixingVolumes.MixingVolume vol(
      prescribedHeatFlowRate=true));

  parameter Modelica.SIunits.HeatFlowRate Q_flow_nominal
    "Heat flow rate at u=1, positive for heating";

  Modelica.Blocks.Interfaces.RealInput u "Control input";
  Modelica.Blocks.Interfaces.RealOutput Q_flow(unit="W")
    "Heat added to the fluid";

protected
  Buildings.HeatTransfer.Sources.PrescribedHeatFlow preHea
    "Prescribed heat flow";
  Modelica.Blocks.Math.Gain gai(k=Q_flow_nominal) "Gain";

equation
  connect(u, gai.u); ... // other connect statements
  annotation (...);      // documentation
end HeaterCooler_u;
```

Add examples and validations to unit testing framework



1. Add validation and stress tests for different model configurations.

2. Validate results and add main outputs to plot script. These variables become part of the regression tests.



3. Run `modelica-buildings/bin/runUnitTests.py`

4. Update `Buildings/package.mo` release notes.

5. Issue pull request on https://github.com/lbl-srg/modelica-buildings.

See Unit Test documentation.

?