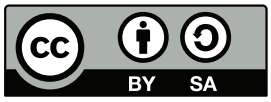
Task 3: IDEAS Tutorial, part 2

Filip Jorissen, Damien Picard, Jelger Jansen, Iago Cupeiro Figueroa

KU Leuven, September 20, 2021



Introduction

The goal of this exercise is to become familiar with the Modelica packages of IDEAS that are complementary to `IDEAS.Buildings`.

For this exercise you will extend the model of a simple house, using HVAC and other components. This exercise starts/extends from the model `IDEAS.Examples.Tutorial.Example5`. Model complexity will be increased in each step by extending the model from the previous step. I.e. for each step a new model should be created that uses the Modelica `extends` clause to extend the previous model. In between steps, the result differences can be compared.

In the following sections the exercise is discussed in several steps. Each step first qualitatively explains the model part. Secondly the names of the required IDEAS models are listed. Thirdly we provide high-level instructions of how to set up the model. If these instructions are not clear immediately, have a look at the model documentation and at the type of connectors the model has, try out some things, make an educated guess, etc. Depending on the parameter values that you choose, results may differ slightly. We also list some side notes, which are not strictly required for this exercise, but which may help you to effectively use Modelica and IDEAS in the future. In some cases we list some important background information.

The main packages that you will need are:

1. `IDEAS.Fluid`: a package containing air flow and hydronic system components,
2. `IDEAS.Media`: a package containing commonly used media implementations that are tailored to building applications,
3. `IDEAS.Utilities`: a package containing components that simplify model development and automation of workflows.

1 Heating system

Qualitative discussion In this step you will extend the model `IDEAS.Examples.Tutorial.Example5` by adding a HVAC system. The system consists of a water-water heat pump, radiators, a storage tank, circulation pumps and a low temperature heat source for the heat pump. Use constant control set points for the heat pump and pumps. Add a system that measures the heat pump electrical energy use.

Required models This step requires the following components:

- `IDEAS.Fluid.HeatPumps.ScrollWaterToWater`
- `IDEAS.Fluid.HeatExchangers.Radiators.RadiatorEN442_2`
- `IDEAS.Fluid.Actuators.Valves.TwoWayTRV`
- `IDEAS.Fluid.Movers.FlowControlled_dp`
- `IDEAS.Fluid.Sources.Boundary_pT`

- `IDEAS.Media.Water`
- `IDEAS.Fluid.Storage.Stratified`
- `Modelica.Blocks.Sources.IntegerConstant`
- `Modelica.Blocks.Continuous.Integrator`

Important notes on Fluid system modelling The model components of `IDEAS.Buildings` have been integrated where possible. E.g. a zone can always have an occupancy schedule, or lighting models. However, for HVAC, many configurations exist in reality. Some zones have a radiator, which physically resides within the zone, while others may use floor heating, which is a property of the floor. Some zones may even have multiple radiators with different sizes, etc. Due to this multitude of options, HVAC components have not been integrated into the `IDEAS.Buildings` components (yet). Instead, they can be added manually by the user. This requires some knowledge by the user about how to interconnect the different components, and even about the relevant physical principles that are at play. One notable example is the fact that we use pressure-driven flow models to compute mass flow rates through hydronic systems. I.e. the mass flow rates are computed from pressure drop equations and the pump head that is generated by a pump. The advantage is that pump and fan powers can be computed accurately and complex hydronic systems can be modelled, where Modelica will automatically deduce the mass flow rate in each branch. The disadvantage is that (somewhat) realistic pressure drop parameters have to be supplied to the model. An alternative modelling approach, which is commonly used by other building energy simulation tools, is to 'outsmart' physics and to directly prescribe mass flow rates. Depending on the user experience, it is however easy to make modelling errors that cause non-physical results or non-physical models, which cause Dymola to throw a 'singular system' error. For instance, when prescribing a flow (instead of pressure), the situation can occur where a flow rate is forced through a closed valve, which is not possible in practice. Our model equations will predict a huge pressure drop over the valve, which can easily lead to a pump power of a few MW and a large temperature increase of the water when passing though the pump [1].

To summarize, modelling fluid flows in Modelica can be quite involved, but you can achieve a high level of detail in doing so. Proceed with care, think about how a system physically works and make sure to include all relevant components. One notable example is that in each fluid loop the *absolute* pressure of that loop has to be defined somewhere: pumps and valves only provide information about *differential* pressures. For this purpose use `IDEAS.Fluid.Sources.Boundary_pT` and connect it to the loop, which will set the absolute pressure at the connection point.

Instructions Start by adding one radiator for each zone and by connecting them to the respective zone models using the (red) heat ports. See the port documentation[2] for more hints about how to do so. The radiator parameters can be set to 500 W, 45/35 degrees inlet/outlet[3] and a pressure drop of 0 Pa[4].

Typically, radiators have a Thermostatic Radiator Valve (TRV) that automatically reduces the flow towards the radiator when the zone temperature is sufficiently high. We model this by adding a TRV model, which uses the zone temperature `TSensor` as an input. Physically, each radiator has its own TRV, so this should be reflected in the model. The nominal mass flow rate of the valve can be set to the nominal mass flow rate of the respective radiator and its nominal pressure drop can be set to 20 kPa.

Add a heat pump. Use performance data for the Viessmann Vitocal 300G BW 301.A21 heat pump. These performance data are for a 21 kW heat pump, which is much more than we require for our heating system (2 radiators of 500 W). Therefore set the heat pump scaling factor to 0.025, which rescales the heat pump behaviour to a thermal power of 2.5 %. Furthermore, choose realistic nominal mass flow rates and set the nominal pressure drop to 10 kPa. Set `enable_variable_speed=false` and set the control input (orange triangle) such that the heat pump is on using an `IntegerConstant`.

Add a storage tank that buffers the warm water that is produced. Connect the storage tank at the secondary (the warm side) outlet of the heat pump. Choose a volume of 100 l (0.1 m^3), a height of 0.5 m and an insulation thickness of 10 cm.

[1] Since the model assumes that the flow work is injected into the stream as a thermal power.
[2] By hovering your mouse over the port.
[3] From these *nominal* values, the model computes the *actual* heat flow rate based on the *actual* inlet temperature.
[4] We here implicitly assume that the radiator pressure drop is negligible compared to the valve pressure drop, which we are about to add.

Add two pumps. At the secondary side of the heat pump, set the pump nominal flow rate to the sum of the radiator nominal flow rates and set the nominal pressure head to 20 kPa. At the primary side, do the same. Both pumps should always be on.

The primary side of the heat pump and its circulation pump should be connected to a heat source. Use the model `IDEAS.Fluid.Sources.Boundary_pT` for this, which acts as a fluid source and a fluid sink. You can specify the pressure and the temperature of the fluid. Set the temperature to 10 degrees.

Add a *water* Medium to the model in the text view and use it in all components.

Compute the electrical energy by integrating the electrical power of the heat pump using an `Integrator`.

A reference implementation for this example is shown in Figure 1.

Side notes By default, the Dymola 'canvas' has a size of 100 pt by 100 pt. When extending a model it can sometimes be useful to enlarge this canvas size by dragging the corners of the (white) canvas. This avoids a lot of scrolling across the model.

IDEAS has multiple heat pump models, which have a different level of detail and which use a different set of parameters as an input. The models `IDEAS.Fluid.HeatPumps.ScrollWaterToWater` and `IDEAS.Fluid.HeatPumps.ReciprocatingWaterToWater` are the most physically detailed and models and should be used whenever possible.

Modelica uses SI units by default, which means that temperatures by default are presented with the units of Kelvin.

Reference result We simulate this model with the settings

1. Start time = 1e7,
2. Stop time = 1.1e7,
3. Number of intervals = 5000.

We thus perform a simulation that starts 10^7 seconds after new year and ends 10^6 seconds later, which is a period of 11.6 days.

We now plot the zone temperatures, the heat pump condenser temperature `heaPum.con.T`, the heat pump heat flow rate `heaPum.QCon_flow` and the radiator heat flow rates `rad.Q_flow`.

The results are shown in Figure 2 and deserve some explanation. Firstly, we see that the zone temperatures do not significantly drift below the 21 degree set point of the radiator thermostatic valves. The temperatures do rise above the set point, which is caused by solar heat gains, not by the radiators.

The supply water temperature of the radiators rises up to 65 degrees. The radiator nominal dimensions are for a supply water temperature of 45 degrees. The elevated temperatures mean that the radiator can emit a higher than nominal thermal power. However, in the bottom graph the radiator thermal power never surpasses even half of its nominal value. This is caused by the thermostatic valves, which reduce the mass flow rate that enters the radiator, and consequently lowers the thermal power. This reduced mass flow rate causes a large temperature difference across the heat pump, which in turn explains the elevated heat pump condenser temperatures. The heat pump model contains a built-in temperature protection, which disables the heat pump when the temperature exceeds a predefined threshold. Consequently, the heat pump continuously switches on and off and operates at high temperatures for most of the time. This switching also negatively impacts the computation time, as you may have noticed.

This example illustrates the importance of control, which is currently not modelled. All pumps and the heat pump are assumed to be active continuously, which is detrimental for the system performance. The COP (`heaPum.com.COP`) is only about 2.7.

2 Adding HVAC control

Qualitative discussion This step adds a controller that disables the heat pump when the supply water temperature exceeds 45 degrees.

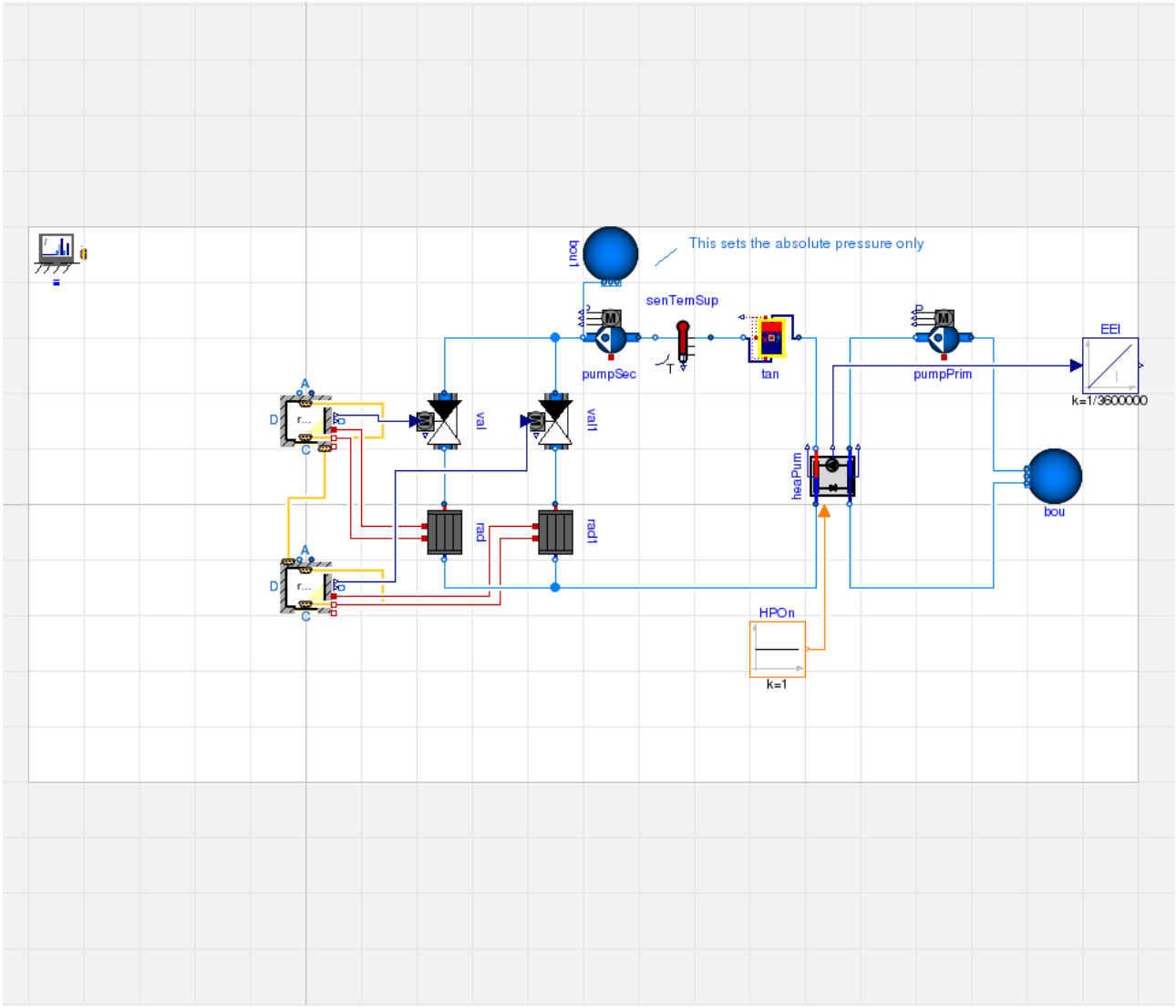


Figure 1: The schematic of Example 6.

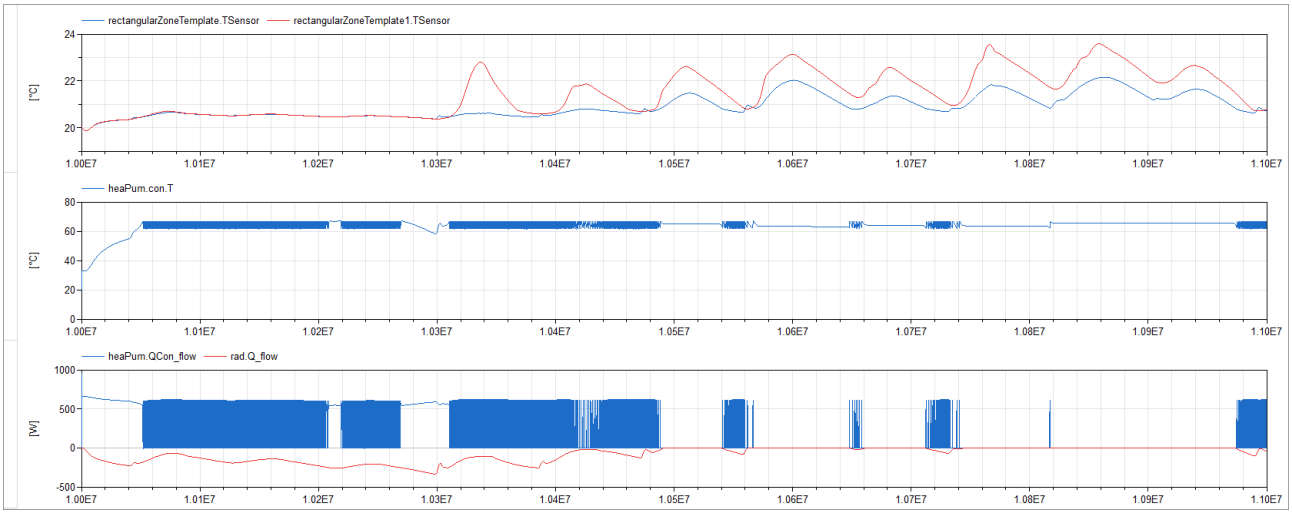


Figure 2: The model response for Example 6.

Required models This step requires the following components:

- `IDEAS.Fluid.Sensors.TemperatureTwoPort`
- `Modelica.Blocks.Logical.Hysteresis`
- `Modelica.Blocks.Math.BooleanToReal`

Instructions Before extending the previous step, add a temperature sensor between the storage tank and the secondary circulation pump. This measurement serves as an input to the hysteresis controller. Now extend the model and configure the controller such that it switches to a 'false' signal below 40 degrees and to 'true' above 45 degrees[5].

The output of the hysteresis controller is thus 'true' when the supply temperature is high enough and false otherwise. This Boolean signal has to be converted in a Real control signal that can be accepted by the heat pump model. For this purpose use the `BooleanToReal` block.

The heat pump already has a control signal. Since blocks cannot be removed from an extension of a model[6], we can (in this case) work around the existing control signal by changing the heat pump model input type to `enable_variable_speed=true`. The model then accepts any real signal and connections to the other control signal are ignored.

Side notes Note that the control signal of the heat pump could also have been used to disable the circulation pumps. However, disabling the circulation pump disables the flow of water, which causes the temperature sensor output to remain fixed at its last value. Since this value, at the point of disabling the pump, was more then 45 degrees, it would stay at that value and consequently the system would not be enabled again. Note that this is not necessarily a model error, since the same problem could occur in reality! This problem can be alleviated by modelling heat losses in the sensor.

We have used a hysteresis controller to enable and disable the heat pump, while we might as well have used a simple 'greater or equal' comparison. If there were no storage tank, this would however introduce very rapid switching between turning the heat pump on and off. Indeed, enabling the heat pump will, almost instantaneously increases the supply water temperature, causing the heat pump to be disabled again. This effect is called chattering and can be very detrimental for the simulation speed and should therefore be avoided. Note that this effect would also occur in practice if hysteresis controllers were not used.

In this example we use a trick to override the heat pump control signal, by enabling a new control signal and by disabling the old one. This does not lead to too many equations, since the Modelica specification states that connections to removed connectors or components should be ignored.

Reference result Figure 3 compares the results with the results without controller. We see that indeed, the supply temperature is reduced significantly. This causes the zone temperature to be slightly lower, up to about 0.25 K. The COP however increases significantly, from about 2.7 to about 3 to 4. Consequently, the energy use over the period is reduced from 16.43 kWh to 11.84 kWh.

Note that this heating system configuration is still not efficient since the small flow rates still cause large temperatures to occur within the heat pump and thus cause a small COP. COP's of more than 5 are obtainable when using a bypass and a separate pump to charge the storage tank.

To further play with this model, you can try to enable/disable the circulation pumps with the same control signal and see what happens. Secondly, you can set the heat pump control signal to 0.3 instead of 1, which causes the heat pump to be modulated to 30% thermal power. This has a large impact on the COP since the temperature difference across the heat pump is reduced. However, the heat pump is unable to satisfy the heat demand.

3 Workflow automation

Qualitative discussion Extracting results from Modelica/Dymola can be tedious. Therefore several custom tools have been developed to facilitate exporting simulation results. For time-series data, a generic CSV writer can be found in `IDEAS.Utilities.IO.Files.CSVWriter`. This model generates a CSV file at a user-defined location that contains data for each of the inputs of the block. The delimiter can be modified in the

[5] Take into account that the measurement has units of Kelvin!

[6] Unless they are conditional, but we have not implemented that in this example.

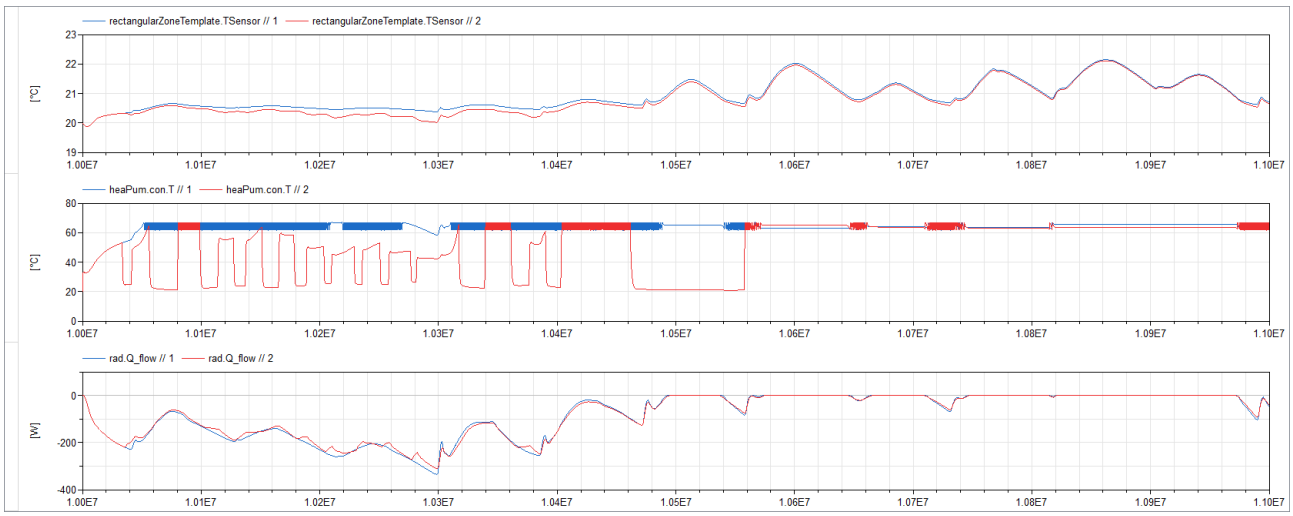


Figure 3: Comparison with (red) and without (blue) control for zone temperature, supply water temperature and radiator thermal power.

advanced parameter tab. The model `IDEAS.Utilities.IO.Files.CombiTimeTableWriter`, does the same, albeit using a slightly different file format which can be read directly back into Modelica using the file reader `Modelica.Blocks.Sources.CombiTimeTable`.

In this example we will not output time series data, but instead a single value: the total energy use at the end of the simulation, using the json file format. Note that the external library `ExternData` can be used to read json files.

Required models This step requires the following component:

- `IDEAS.Utilities.IO.Files.JSONWriter`

Instructions Add the model, choose a file path and indicate the appropriate time when the result should be saved. Connect the appropriate signal to the input of the block.

Side notes Unless you specify a full path, the file will be generated in the current working directory of your Dymola workspace. See File > Working directory > Browse to inspect your current working directory. In this directory Dymola will generate all its own result files, C code, executables, etc. It is good practice to set this directory where the resulting output does not cause a mess. Note that by default Dymola sets this directory to the library directory when opening a library.

Reference result Check the contents of the generated file. Depending on the chosen value for the parameter `varKeys`, you should get something similar to:

```
{
  "Electrical energy [kWh]" : 1.2577137592e+01
}
```

4 Adding CO_2-controlled ventilation

Qualitative discussion You will now add a CO_2-controlled ventilation system. Add the occupancy model from Example 4 to one zone and a fixed occupancy of 1 person to the other zone. The ventilation system consists of two fans, two supply and two return air VAVs (Variable Air Volume), a heat recovery unit and an outdoor air source. The control consists of PI controllers with a set point of 1000 ppm.

Required models This step requires the following components:

- Custom occupant model from Example 4
- `IDEAS.Fluid.Sources.OutsideAir`

- `IDEAS.Fluid.Actuators.Dampers.PressureIndependent`
- `IDEAS.Fluid.HeatExchangers.ConstantEffectiveness`
- `IDEAS.Controls.Continuous.LimPID`
- `Modelica.Blocks.Sources.Constant`

Important notes on Media `IDEAS.Media` contains a selection of medium *packages*. Media packages contain the properties of media such as water or air. Depending on the user requirements, the level of detail in these packages can change a lot. For building applications, we use relatively simple media.

`IDEAS.Media.Water` assumes constant density and heat capacity and is thus a very simple medium. `IDEAS.Media.Antifreeze` contains media such as glycol water, which is a commonly used antifreeze.

`IDEAS.Media.Air` is a simple air medium that neglects the temperature dependency of the density of air to facilitate the numerics. By default, air is however compressible such that the density varies with pressure. `IDEAS.Media.Air` also contains water vapour, which allows the zone humidity to be computed. By default `IDEAS.Media.Air` does not model CO_2. However, any number of 'trace substances' can be added to the air model. This can be CO_2, VOC, dust, as long as it is simply transported together with the air stream without changing properties. Note that you have to ensure that all coupled components must have the same Medium for the model to work. I.e. if one zone is linked to a second zone through an air connection, then each of them must have the same set of trace substances.

Instructions Extend this model from Example 7 and modify its existing medium declaration to add CO_2, as follows:

```
extends Example7(
    redeclare package Medium = IDEAS.Media.Air(extraPropertiesNames={"CO2"}));
```

For one zone, add the occupancy model from Example 4. You can copy it from `IDEAS.Examples.Tutorial.Example4`. For the other zone, add a fixed occupancy of 1 person.

For making a connection with the outside air we use `IDEAS.Fluid.Sources.OutsideAir`, which is similar to `IDEAS.Fluid.Sources.Boundary_pT` except that it automatically sets the outdoor dry bulb temperature and humidity.

These were two tricky steps that can be quite complex for new users. Now add two supply and two return air VAVs [7]. Connect them to the `FluidPort`s of the zones. Ensure that the same port is not used twice, otherwise the air will not enter the zone and simply return from one connection into the other at the port. Also, use the convention that air enters in the blue port and leaves through the white port. For this exercise, assume that the VAV has a nominal flow rate of 100 m^3/h, which equals $100 * 1.2/3600$ kg/s. Assume a nominal pressure drop of 50 Pa and also add `dpFixed_nominal=50`, which causes the VAV model to include a pressure drop of ducts, grills, filters or bends that are connected at the inlet or outlet of the VAV.

Next, add a supply fan and a return fan to which the VAVs are connected in parallel. The fan pressure head is constant at 200 Pa and its nominal flow rate is the sum of the VAV flow rates.

Finally, add a heat recovery heat exchanger with a constant effectiveness of 80 %. Choose reasonable nominal flow rates and set the nominal pressure drop to 100 Pa. Make sure to use the new air Medium for all components that were just added.

The ventilation system model is now ready but cannot be simulated yet since the control input of the VAVs is not set. For this purpose, add two PI controllers and connect their outputs to the VAVs. Connect the zone `ppm` outputs to the PI controller measurement inputs `u_m`. Connect a constant set point of 1000 ppm to the PI input `u_s`. Add a minimum VAV opening of 10 %. Furthermore, set `k=0.005`, `Ti=300`, `reverseAction=false` and `controllerType=PI`. See the side notes for a motivation for these parameters, which is beyond the scope of this course.

Side notes Some background information on PI controllers, which are often used in buildings: PID controllers track a set point for some variable (in this case CO_2 concentration) `u_s` by checking the error compared to a measurement of that variable `u_m` and by adjusting the control output `y`, which should be connected to some actuator (in this case the VAV) that affects the measured variable. The PID controller control internally computes the error $e = u_s - u_m$ and the output y consists of three (optional) parts:

[7] A VAV works like an 'air valve'. It has a built-in disc that is rotated to control the air flow rate. An internal controller rotates the disc such that a set point mass flow rate is tracked. The set point equals `m_flow_nominal*y` where y is the control input.

1. the proportional part (P), which is proportional to e,
2. the integral part (I), which is proportional to the time integral $\int_0^\tau e \, \mathrm{d}t$
3. the differential part (D), which is proportional to the time derivative $\frac{\mathrm{d}e}{\mathrm{d}t}$.

In a PI controller, the differential part is not included, etc. A PID controller has an upper bound and lower bound for the output and further has three main parameters that should be tuned for the system to work effectively. Conceptually speaking, we have to tell the PID controller *how much* it should change its output for an observed error e. To tune our PI, we used the following reasoning. If the PI suddenly observes an error e of 200 ppm, we want the PI to change to fully open (assuming it was closed before). The PI implementation is as follows [8]

$$y = ke + k/Ti \int e \mathrm{d}t. \tag{1}$$

Since the integral term takes a while to respond, an instantaneous step is only affected by $y = ke$. Furthermore, we want $\Delta y = 1$ for $\Delta e = 200$, which results in $k = 0.005$.

The proportional term k may not be sufficient to track the set point. The remaining error can be compensated by the integral term, which can also be tuned using parameter Ti. The reasoning here is as follows. If an error of 200 ppm persists for Ti seconds, we want the PI controller output to increase an additional 100 % during that period. In this case we chose $Ti = 300$ seconds. We have also set `reverseAction=false` to indicate to the PI, that if the set point is larger than the measurement, the control output should decrease instead of increase, which is the default. The same behaviour can be obtained by switch the two PI inputs.

Note that it may be tempting to set k large and Ti small in order to get a controller that reacts very quickly. However, if the system itself (the VAV and zone CO_2 concentration) reacts with a delay, this can cause the system to oscillate. You can observe such behaviour when setting `riseTime=300` for the VAVs and `Ti=60` in the PIs. These oscillations cause the solver to take smaller time steps, which prolongs the simulation time. Note that this is again not a software problem since these oscillations can/would occur in practice too!

By default, both the humidity of air, and the generation of water vapor by occupants are simulated. Furthermore, natural air infiltration in zones[9] is quite low, which can cause the zone humidity to increase strongly, even above 100 % since water condensation is not modelled by default. This can explain seemingly wrong results when not modelling a ventilation system.

Note that the default outdoor CO_2 concentration is fixed to 400 ppm. It can be changed using the parameter `ppmCO2` in the `SimInfoManager`. You can even make it time-dependent by overriding the value of the `RealExpression` block `CEnv` in the `SimInfoManager`.

Reference result The zone temperature, CO_2 concentrations and PI control signals are plotted in Figure 4. Note the small overshoot of the PI controller outputs and the exponential decay towards the outdoor CO_2 concentration when there are no occupants.

5 Computation time

The models that you have just created are fairly slow and computation time can easily explode when controller oscillations occur or when the heat pump experiences frequent on/off switches. These effects cause a lot of fast transients that force the solver to take small steps, which takes a lot of time.

Fortunately, there are many tricks that can be used to speed up the solver. The fundamental principle is that we remove small time constants from the problem,. `IDEAS.Examples.Tutorial.Example10` implements changes that cause the simulation to become 2 times faster. If we do this systematically, and we remove **all** fast time constants, we can use a different solver (Euler), using which the simulation time also becomes 2 times smaller when using a fixed time step of 20 seconds. These are modest improvements since this small example model behaves rather well. However, for large models, the difference in computation time when using Euler integration can become a factor 1000. The modifications however require a bit of knowledge about solvers and the models that you are using, including some of the more advanced parameters. To learn more about this, we refer to [1, 2, 3].

[8] You can check `IDEAS.Controls.Continuous.LimPID` to verify this.
[9] See parameter n50.

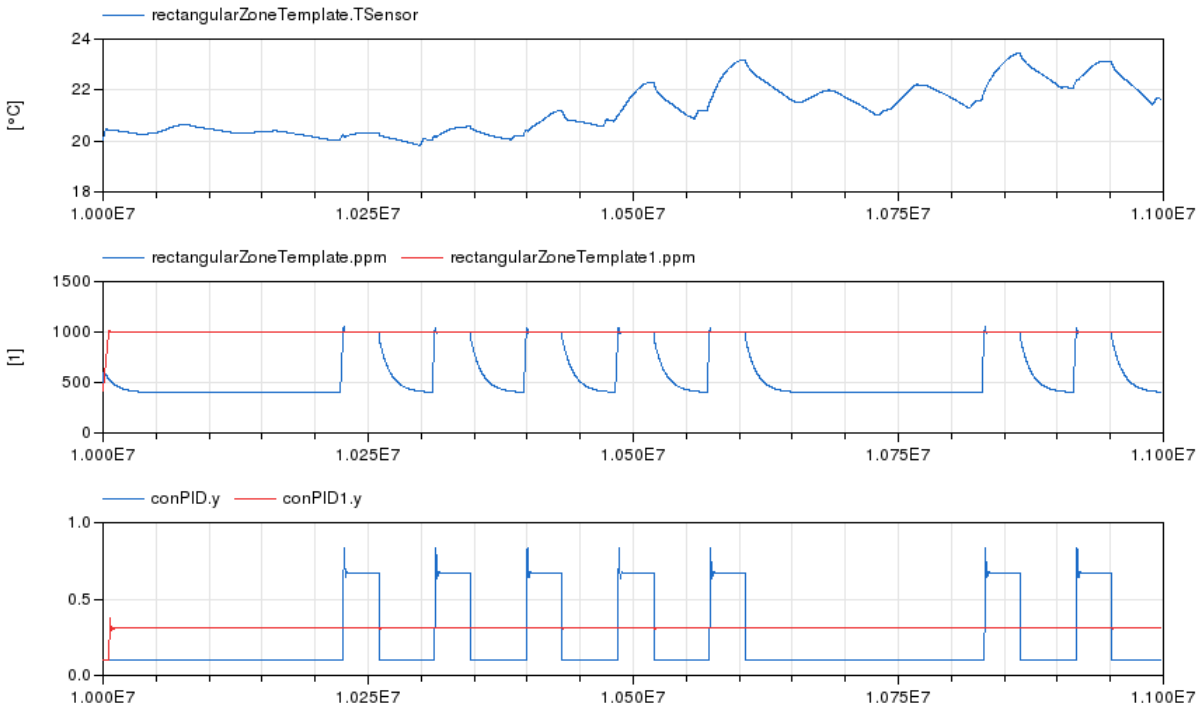


Figure 4: Zone temperature, CO_2 concentrations and PI control signals.

References

[1] F. Jorissen, M. Wetter, and L. Helsen. Simulation Speed Analysis and Improvements of Modelica Models for Building Energy Simulation. In 11th International Modelica Conference, 59–69, Paris, 2015. doi: 10.3384/ecp1511859

[2] F. Jorissen, M. Wetter, and L. Helsen. Simplifications for hydronic system models in modelica, Journal of Building Performance Simulation, 11:6, 639-654, 2018, doi: 10.1080/19401493.2017.1421263

[3] F. Jorissen. *Toolchain for Optimal Control and Design of Energy Systems in Buildings*. Phd thesis, Arenberg Doctoral School, KU Leuven, April 2018.