APPENDIX D

Simulation Models

Abstract

This appendix gives an overview and a description of the Modelica models developed in the OpSys project (supplemented with screenshots from Dymola). The models are based on the house model presented in Appendix C and consists of different reusable subcomponents organized in Modelica packages. Modelica enables export of functional mockup units (FMU's) for simulation in other environments such as Python, which is utilized in the OpSys testrig setup.

Kasper Vinther

Aalborg University

Table of Contents

D.1 Overall Simulation Model Structure

The models are split into two main Modelica packages “OpSys_Modelon” and “OpSys_Experiments (see Package Browser view in Figure D.1).

The package OpSys_Modelon consist of prebuild example models of different house types:

- BR10_house
- BR1970_house
- BR1970_house_mod (the floor heating in room 1 and 2 is split up into 4 and 3 circuits respectively in this model).

These models are based on room models, which are interconnected with component models of Ceiling, LayeredWall, LayeredWall_Window, and Door. The Room and component models are further described in Section D.2. A house model is given an interchangeable record with parameters for a given house type (House_1970, House_BR10, House_2015), which means that a model with a BR15 house specification can easily be built from existing examples. The parameter values for each house type is presented in Appendix C.

The package OpSys_Experiments contain full simulation models loaded with input data for an entire year, see Section D.3. One of these models is BR1970_house_simulation_with_data, which is prepared for use together with the Testrig. A top layer view of this simulation model is also shown in Figure D.2. The boundary inputs to the model (inputs from the Testrig) are forward temperature and floor heating water mass flows in each of the four floor zones. Note that zone 1 has two flow inputs, which are added together, because the Testrig has two valves and two flow measurements for this zone. Additionally, each flow input is filtered with a second order Modelica Standard Library (MSL) filter to ensure faster and more robust simulation (gives smooth continuous input signal and derivatives). The filter cutoff frequency can be chosen fast compared to the dynamics of the flow inputs.

MSL combiTimeTables are used to load a txt data file with ambient temperature, solar heat input into each window, and heat input from people and appliances. The air ventilation rate for each room is fixed in the simulations and set as described in Appendix C. No measures for preventing overheating during

the summer have been included in the present work as the focus is on the heat and power demand during the heating season.

D.2 Sub-component Model Description

Figure D.3 provides an overview of the room model. The central heat capacity represents the air volume and associated room temperature. The heat capacity is connected with the surroundings through Modelica heat ports. Q_delta is the heat input from ventilation calculated based on the inputs Ventilation_Flow and T_ambient. The InternalHeatFlowPort input is the heat input from people and appliances and the InternalHeatFlowPort2 input is the heat input from the sun to the air and floor (divided by a gain set to 0.5 – half of the sun heat input goes directly to the air and the other half flows into the floor).

Each room model also consists of a floor heating model with a heat connection to the room air heat capacity. The content of the floor model is shown in Figure D.4. In the project, it was decided to discretize the floor along the floor heating pipe length in 10 zones with vertical heat conduction up through the floor (no heat conduction horizontally along the discretization). Additionally, each zone has four heat resistances; water to floor capacity, water to ground temperature, floor to upper floor, and upper floor to air. This gives the resistive-capacitive network shown in Figure D.4. The dynamic floor heating pipe model (blue cylinder) is from MSL and with Fluid.Pipes.BaseClasses.FlowModels.Turbulent-PipeFlow as flow model (Quadratic turbulent flow in circular tubes and roughness 0.7e-5 m similar to floor heating PEX tubes). The diameter of the pipe is set to 0.012 m for the pipes in room 1, 2, and 4 (wood flooring) and 0.02 m for the pipe in Room 3 (concrete flooring).

Figure D.5 shows the LayeredWall_Window model consisting of a wall and a window heat capacity. These capacities are connected to the surroundings through thermal resistances and heat ports. The input sun_to_wall provides the possibility of adding heat directly from the sun into the wall capacity (not used in the simulation models in this project). The LayeredWall and Ceiling models are similar just without windows.

The Door model provides the possibility to add heat transfer between rooms caused by ventilation of air through door openings. The input to the model is the ventilation rate and the room temperature on each side of the door. The output heat transfer rate can be added to the internal heat gain of each room (note that one of the rooms should have heat transfer added with opposite sign).

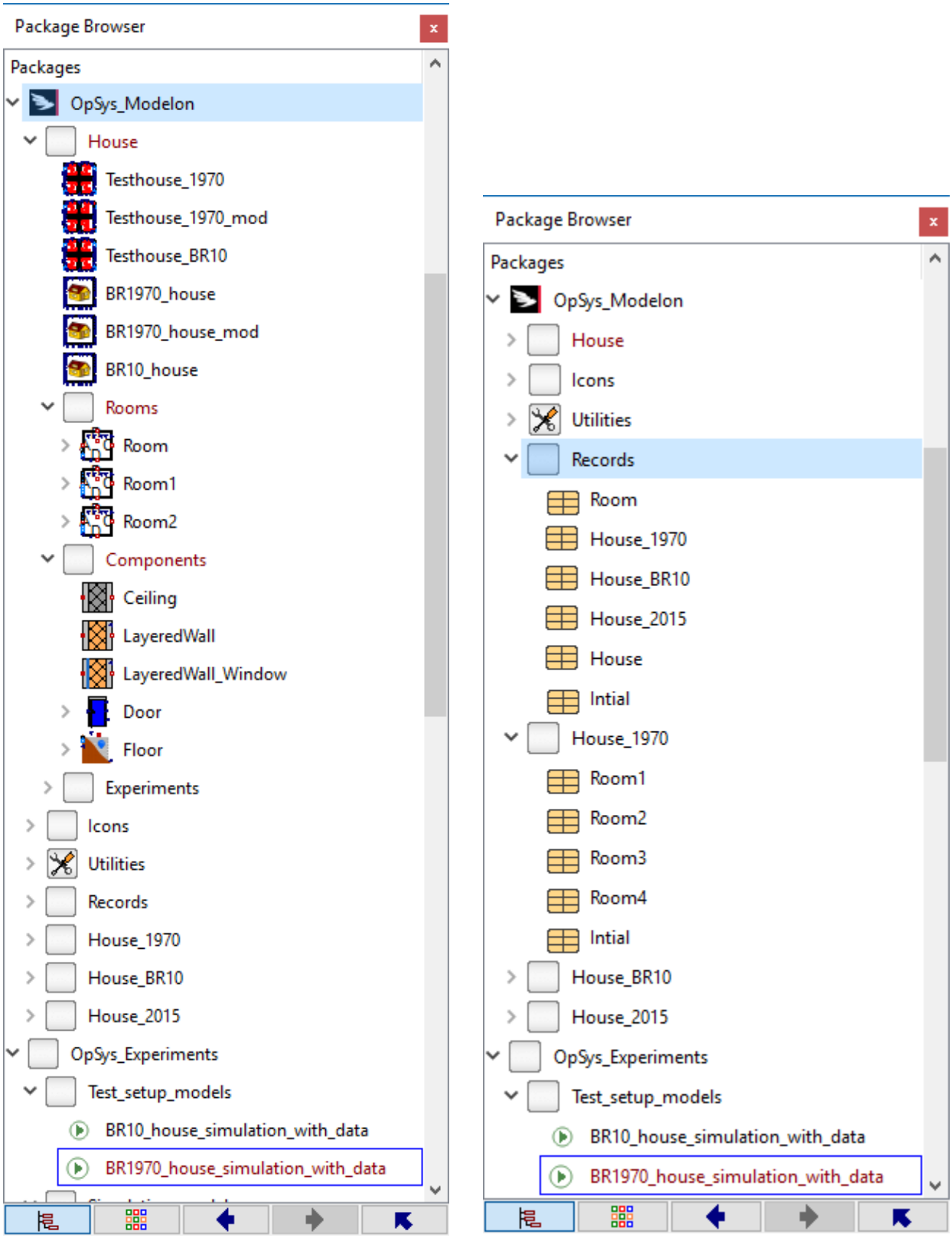

Figure D.1: Package browser overviews of the OpSys_Modelon package and parts of the OpSys_Experiments package with highlighting of the main simulation file for the Testrig (under Test_setup_models).

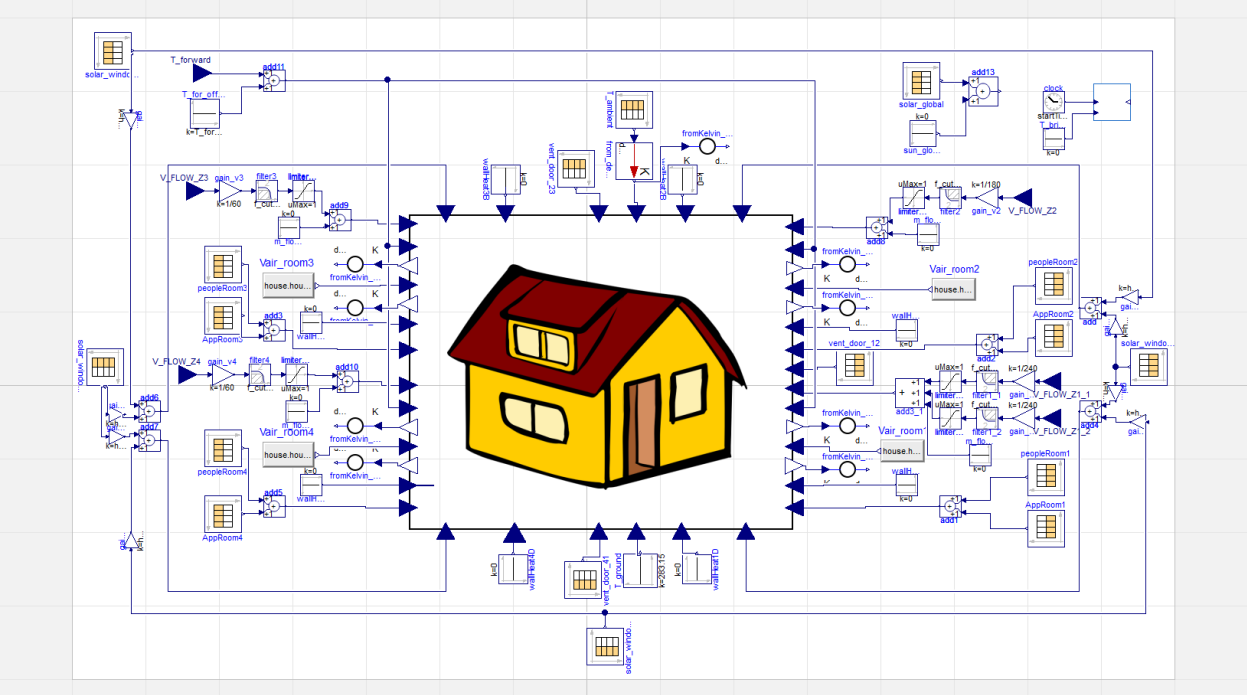

Figure D.2: Top layer view of house simulation model used for the Testrig (model "BR1970_house_simulation_with_data").

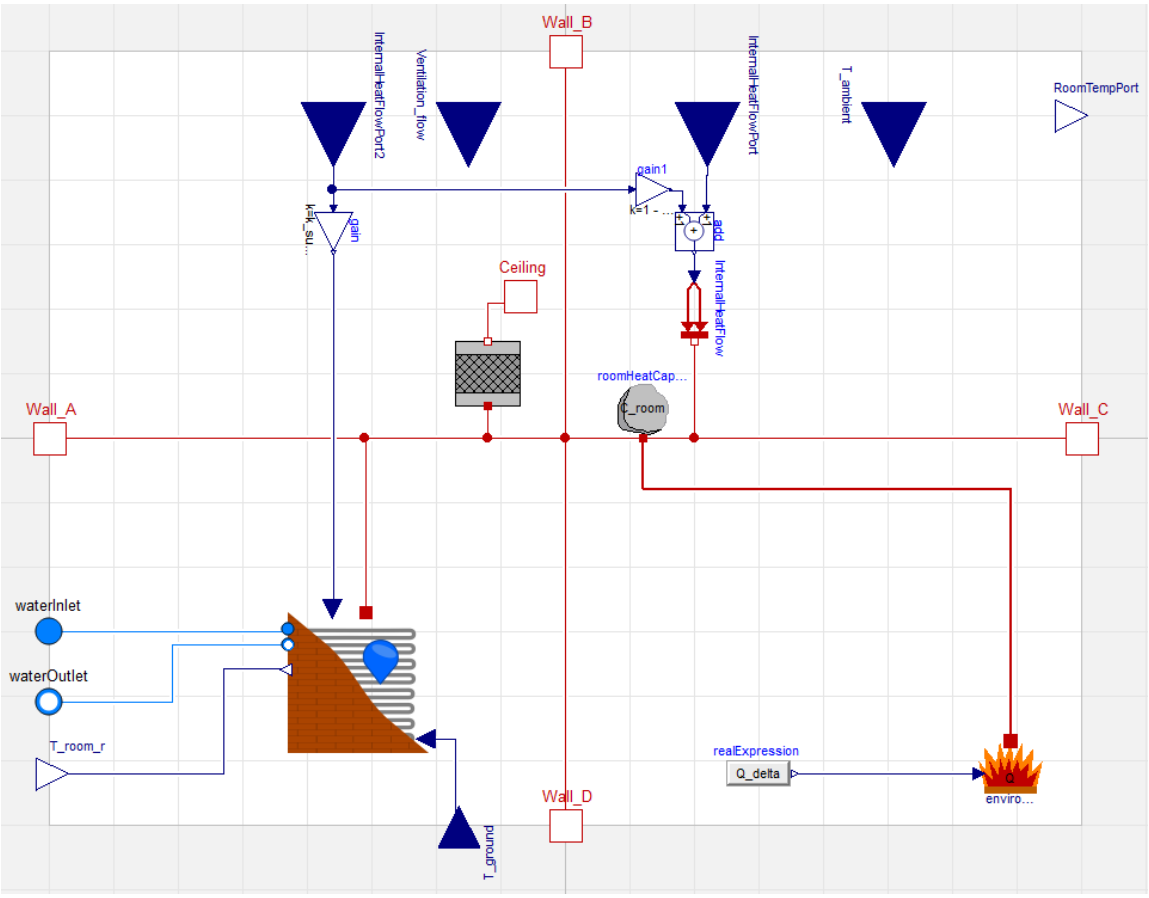

Figure D.3: Room model with a central heat capacity and connections to walls, ceiling, floor, and internal heat gains.

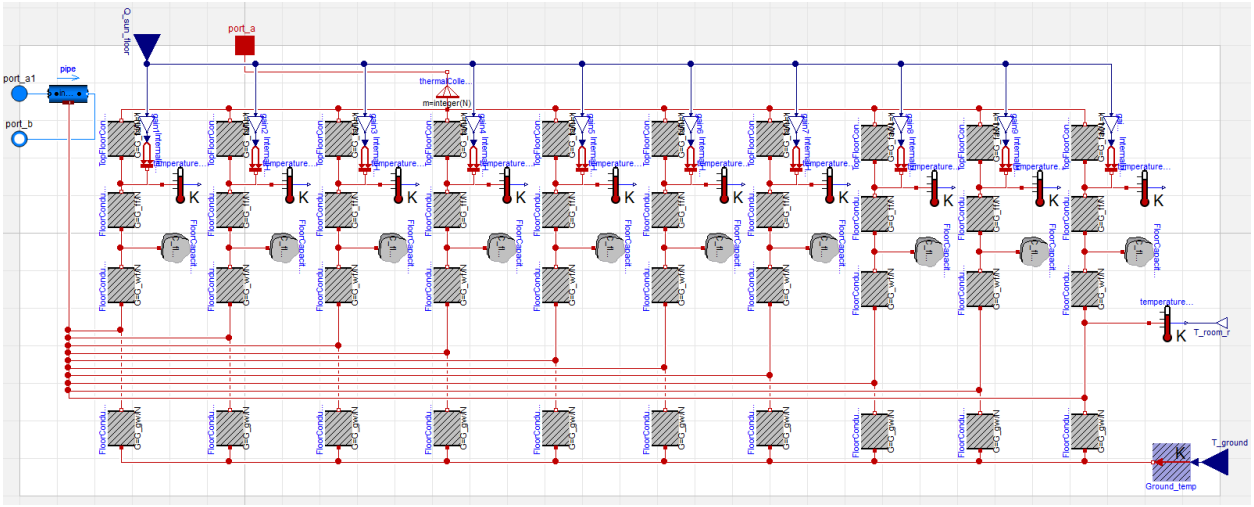

Figure D.4: Floor model consisting of an MSL dynamic pipe model with interchangeable fluid media model (blue cylinder) and a resistive-capacitive network representing a discretized floor.

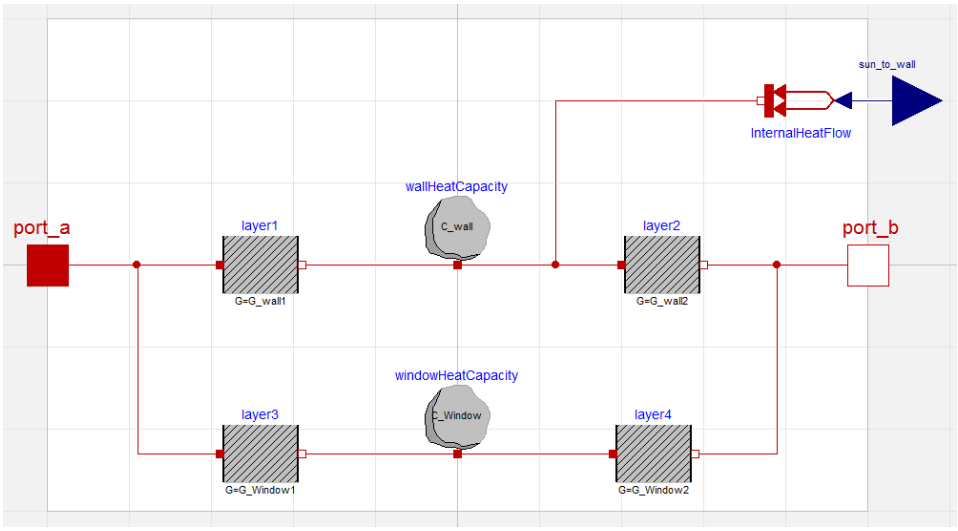

Figure D.5: LayeredWall_Window model consisting of a resistive-capacitive network.

D.3 Load Data File Description

Three txt files with tabulated data are generated with hourly load values for a reference year (see description of data in Appendix C):

- all_data_1970.txt
- all_data_BR10.txt
- all_data_BR15.txt

Three additional files are generated, where normally distributed noise is added to each value in the schedules for internal heat gains from people and appliances, since they are expected to exhibit a stochastic behavior:

- all_data_1970_noisy.txt
- all_data_BR10_noisy.txt
- all_data_BR15_noisy.txt

A standard deviation of 10% of the value is chosen for the normally distributed noise and it is added to provide data, which is not entirely predictable just by looking at previous days. More elaborate randomness/noise studies could be conducted by changing the content of the above-mentioned files. A parameter in the simulation model determines which file is in use (should correspond to the house type used in the simulation).

Example of a part of the txt data file content (first column is time in seconds and second column is value):

double equipment_living_room(8761,2)
0,152.2166;
3600,170.94047;
7200,111.82096;
10800,156.90409;
…
double people_living_room(8761,2)
0,0;
3600,0;
7200,0;
10800,0;
14400,0;
18000,0;
21600,146.05664;
…

Note that weather data and brine temperature are also included in the load data files.

D.4 Simulation Without Hardware

Models similar to the entire test rig setup are also provided in the OpSys_Experiments package for simulation without hardware in the loop (see package overview in Figure D.6). This provides the possibility of running long simulations faster than real time and makes it possible to test control algorithms before implementation on the Testrig.

The subfolder Simulation_models consist of various prebuilt models:

- **BR1970_house_without_hp** – This model consists of a house using a 1970's house parameter record, models for the manual adjustment valves (fixed opening degree for balancing of pressure drops in floor heating pipes), valve models with on/off hysteresis type control for room temperature control (controllable thermal wax valves), and a circulation pump model with variable rotational speed for control of the temperature drop across the floor heating pipes (e.g., more flow gives lower temperature drop). The heating source is not included and the forward temperature is therefore set directly.

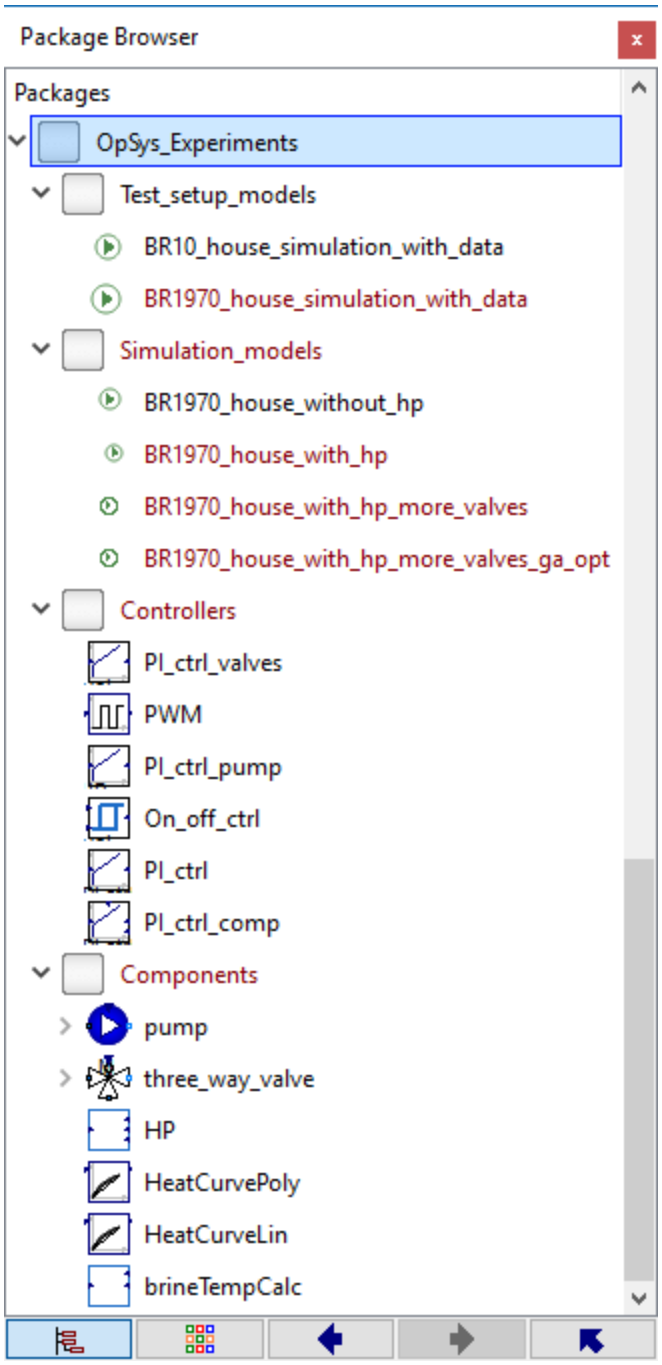

Figure D.6: Overview of the OpSys_Experiments package with models for Testrig experiments, models for simulation without hardware, different controllers, and specific components for simulation without hardware (e.g., heat pump and heat curves).

- **BR1970_house_with_hp** – This model is similar to the previous model with the addition of a simple static heat pump model that provides heat input to a dynamic MSL pipe model. The circulation pump control is replaced by a fixed speed setting and the heat pump is instead supplemented with PI control of the forward temperature. The forward temperature setpoint is generated by a heating curve function based on the current outdoor ambient temperature.

- **BR1970_house_with_hp_more_valves** – This model is again similar to the previous model, but with a split of the floor heating circuits to room 1 and 2. Room 1 and 2 has four and three identical individual floor heating loops, respectively. The length of each loop is 90 m in Room 1 and 80 m in Room 2. This ensures that the length of each individual loop does not exceed 100 m. Additionally night setback is added to the room temperature control between the hours 22.00 – 5.00 (from 22°C to 19°C).
- **BR1970_house_with_hp_more_valves_ga_opt** – This model does not have room temperature control, but is prepared for implementation of more advanced control of the floor heating valves in, e.g, Python. This means that the valve opening degrees are inputs to the model along with the possibility of offsetting the ambient temperature measurement used in the heat pump heat curve. The model is specifically used in OpSys in connection with design of neural networks and genetic algorithm optimization. A screenshot of the model in Dymola is shown in Figure D.7.

D.4.1 Floor Heating System Model Overview

A floor heating system is shown in Figure D.7, with the center square box representing the house model with four rooms.

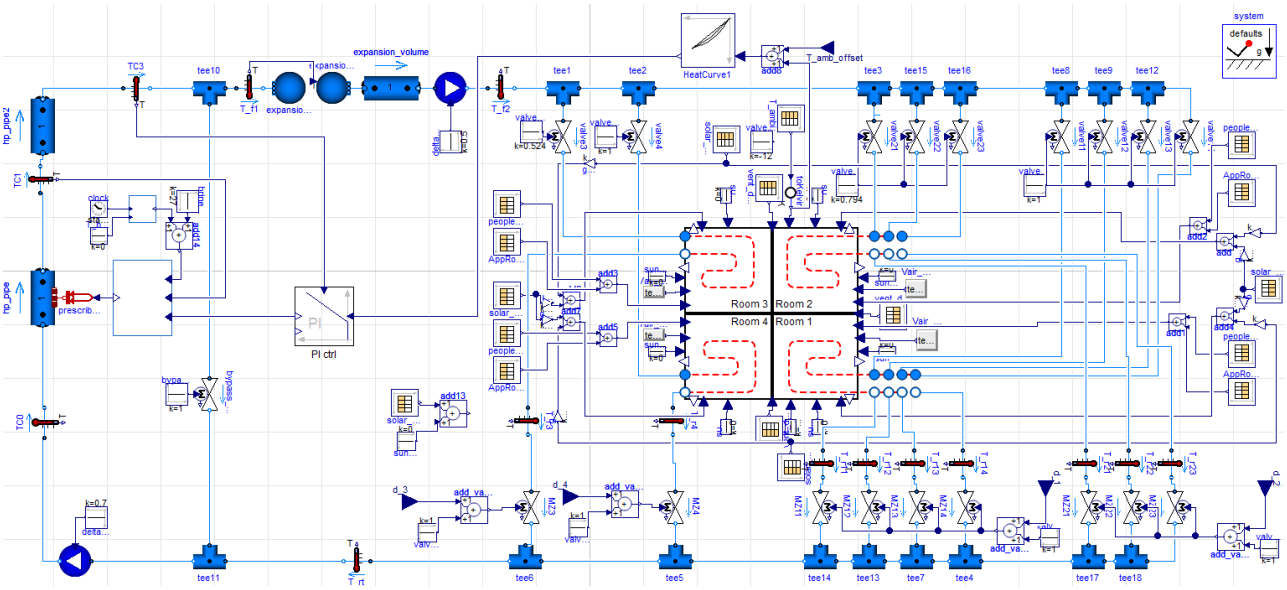

Figure D.7: Overview of the simulation model BR1970_house_with_hp_more_valves_ga_opt. d_1, d_2, d_3, d_4, and T_amb_offset are the five external inputs to the simulation (e.g., determined in Python and set as input to the FMU of the simulation model).

Load data is provided through combiTimeTables in the same manner as when running simulations on the Testrig. The blue connections/lines represent the incompresible fluid flow in the network. Starting from the left, the fluid goes through the dynamic pipe model hp_pipe, where heat is applied from the heat pump model through the prescripbed heat flow interface (red component). The additional pipe hp_pipe2 represent a volume, which can be seen as the volume in pipings and the heat exhanger. The volume of the two pipe sections are in OpSys FMU's set to 5 liters each. A bypass valve and a heat pump circulation pump ensures a constant circulation of water even with all floor heating valves closed. The pump is set to a fixed speed, but this can be upgraded with more advanced control if needed. Another

circulation pump is used to supply the floor heating loops (also fixed speed) and an expansion volume is placed before this pump with a volume of 32 liters in OpSys FMU's. The water flow is then distributed among the nine manual adjustment valves shown in the top part of Figure D.7. The specific distribution is determined by the on/off state of the bottom nine control valves and the particular setting on the manual adjustment valves (set to provide similar flows as in the Testrig setup). The dynamic tee junctions on each side of the floor heating valves, which splits and collects all the flows, are set to have a volume of 0.2 liters.

D.4.2 Sub-component Model Choices and Default Parameters

- Dynamic valve and pipe models are from the MSL and the nine control valves have input filters on the opening degree, with a rise time of 300 seconds, to emulate the slow dynamics of the thermal wax valves.
- The heat pump model is based on results from the IPower project (see Jensen, S.Ø., Christiansen, C.H., Jørgensen, D.M. and Huet, J., 2016. Smart Meter Case Study. Danish Technological Institute) and provides a simple static function that calculates the heat pump heat input to the floor heating system Q_hp from the current brine temperature, the forward temperature, and the input power:

```
deltaT = T_forward - T_brine;
COP_carnot = T_forward/deltaT;
eta = -0.02623*Power/1000 + 0.0010993*deltaT + 0.4016;
COP_hp = eta*COP_carnot;
Q_hp = Power*COP_hp;
```

  The input power is limited to between 500 and 2500 W for reasonable dimensioning of the heat pump to the particular house and floor heating system (can be changed by setting the parameters PI_comp.ymin and PI_comp.ymax). Below 500 W the heat pump is on/off controlled, while between 500 and 2500 W the power to the heat pump is continuously controlled in order to fit the needed heat input to the house
- The starting time of the simulation in seconds from 1st of January (time_offset) and the current simulation time (time_sim) is used to determine the brine temperature used in the heat pump calculations (see Appendix C);

```
day = (time_sim - time_offset)/(24*60*60);
Temp_brine = 0.8358 - 8.854e-3*day - 1.349e-3*day*day + 3.463e-5*day*day*day
             - 2.154e-7*day*day*day*day + 5.162e-10*day*day*day*day*day
             - 4.328e-13*day*day*day*day*day*day;
```

- The heat pump heat curve is implemented with the following equations:

```
TSup_nonsat = TSup_min + (TSup_nominal - TSup_min)/(Tamb_nominal - Tamb_min)*(Tamb
              - Tamb_min);
TSup = max(TSup_min, min(TSup_max, TSup_nonsat));
```

  The desired saturated forward supply temperature is TSup and parameter values are set to TSup_nominal = 323.15 K, Tamb_nominal = 258.15 K, TSub_min = 298.15 K, TSub_max = 323.15 K, and Tamb_min = 293.15 K in the prebuilt simulation models (can be changed, e.g., using the parameter hP.TSup_nominal) .
- The pump models are based on a quadratic map for the Grundfos UPM2 25-60 180 pump:

```
H_pump = -0.183*abs(port_a.m_flow*3.6)*(port_a.m_flow*3.6) -
          0.1804*(port_a.m_flow*3.6)*speed + 7.0579*speed*speed;
```

```
P_pump = -3.3768*abs(port_a.m_flow*3.6)*(port_a.m_flow*3.6)*speed +
          44.6567*(port_a.m_flow*3.6)*speed*speed +
          35.7518*speed*speed*speed + 0.0698*speed + 1.0575;
```

H_pump is the pump head in meters (converted pressure by multiplying with 1e4) and P_pump is the pump power consumption. The speed should be in the interval 0.32-1 or 0. The flow through the pump is equal to port_a.m_flow.

D.4.3 Baseline Control and Default Parameters

- Hysteresis control is used on each of the floor heating valves to determine if they should be on or off in the next time interval (set to 10 min) based on the current temperature in each room. The hysteresis threshold, around the desired room temperature, is set to +/-0,5. This means that the valve needs to be in the same state for at least 10 min and the temperature needs to change at least 1 K to limit the number of switches. The thresholds can be changed through the parameters valve1_ctrl.offset_upper and valve1_ctrl.offset_lower and the sampling time is set using the parameter valve1_ctrl.t_s (likewise for the remaining three valves).
- The heat pump is set to control the forward temperature and is implemented using a discrete PI controller with anti-windup. The sampling time is set to 1 min (parameter PI_comp.t_s) and the gain and integral time constant are set to 50 and 120, respectively (parameters PI_comp.K and PI_comp.Ti). The heat pump is automatically switched off if the forward temperature exceeds the upper threshold set to 323.15 K (parameter PI_comp.PVmax). The heat pump is also set to switch off if the requested heat pump power drops below the lower limit PI_comp.ymin in order to be able to emulate on/off control of the heat pump. The heat pump is only allowed to switch on again after a waiting period of 10 min, again to limit the number of switches (parameter PI_comp.backoff_wait).

D.4.4 Python Interface for more Advanced Control

An FMU of the Dymola models can be generated and simulated in, e.g., Python (PyFMI package needs to be installed). A simple example of FMU simulation is provided in the following. Note that good results in terms of simulation speed and robustness have been obtained by choosing the solver Radau IIa and exporting the FMU using co-simulation with Dymola solvers.

```
1.  import numpy as np
2.  import os as O
3.  from pyfmi import load_fmu
4.
5.  def run_sim():
6.
7.      # Load the dynamic library and XML data
8.      curr_dir = O.path.dirname(O.path.abspath(__file__));
9.      fmu_name = O.path.join(curr_dir,'OpSys_0Experiments_Simulation_0models_BR1970_0hous
    e_0with_0hp_0more_0valves.fmu')
10.     model=load_fmu(fmu_name,kind='CS',log_level=0)
11.
12.     # Setup FMU simulation
13.     Ts = 60*10 # sample time
14.     Tstop = 24*60*60*2 # simulation stop time
15.     opts = model.simulate_options()
16.     opts["ncp"] = 1 # number of simulation steps to save in the simulation result
17.     opts["filter"] = ['testhouse.T_room1',
18.     'testhouse.T_room2',
```

```
    'testhouse.T_room3',
    'testhouse.T_room4',
    "hP.Power"] # filter is optional, but can be used to limit size of the result
    opts["initialize"] = False # false is a requirement for continued simulation
    model.set("start_time_sim", -24*60*60*30) # how to change a parameter
    model.set("T_brine_time_offset.k",-24*60*60*30) # start sim 30. january
    model.setup_experiment(start_time=0)
    model.initialize()

    # Go through all iterations
    t = 0.0
    for k in range(1,Tstop/Ts):

        # Implement a more fancy advanced controller than this
        T_amb_offset = 0
        d_1 = 1
        d_2 = 1
        d_3 = 1
        d_4 = 1

        # Propagate simulation model with chosen input
        u_t = np.linspace(t,t+Ts,2)
        u_traj = np.transpose(np.vstack((u_t,
        [T_amb_offset,T_amb_offset],
        [d_1,d_1],
        [d_2,d_2],
        [d_3,d_3],
        [d_4,d_4])))
        input_object = (['T_amb_offset','d_1','d_2','d_3','d_4'],u_traj)
        res = model.simulate(start_time=t,final_time=t+Ts, input=input_object, options=
opts)

        # Extract sim result
        T_a1 = res['testhouse.T_room1'][-1] # [-1] to get last sample
        T_a2 = res['testhouse.T_room2'][-1]
        T_a3 = res['testhouse.T_room3'][-1]
        T_a4 = res['testhouse.T_room4'][-1]
        Power = res['hP.Power'][-1]

        t = t+Ts
```

Note that simulation model saved as an FMU in this case needs to be located in the same folder as the Python code (remember to also have the data file, e.g., all_data_1970_noisy.txt, located here). A more extensive list of potential variables in the simulation result is listed here;

Variable name in FMU	Description	Unit
TC3.T	Forward temp. before bypass valve	K
T_f2.T	Forward temp. after bypass valve	K
HeatCurve1.TSup	Reference forward temp. from heat curve calc.	K
add9.y	Ref. temp. Room 1	K
add10.y	Ref. temp. Room 2	K
add11.y	Ref. temp. Room 3	K
add12.y	Ref. temp. Room 4	K
testhouse.T_ambient	Ambient temp.	K
add8.u2	Ambient temp. offset (used in heat curve calc.)	K
testhouse.Q_sun_1	Solar radiation into Room 1	W

testhouse.Q_sun_2	Solar radiation into Room 2	W
testhouse.Q_sun_3	Solar radiation into Room 3	W
testhouse.Q_sun_4	Solar radiation into Room 4	W
testhouse.Q_int_1	Gain from people and appl. Room 1	W
testhouse.Q_int_2	Gain from people and appl. Room 2	W
testhouse.Q_int_3	Gain from people and appl. Room 3	W
testhouse.Q_int_4	Gain from people and appl. Room 4	W
testhouse.T_room1	Temp. Room 1	K
testhouse.T_room2	Temp. Room 2	K
testhouse.T_room3	Temp. Room 3	K
testhouse.T_room4	Temp. Room 4	K
testhouse.T_room1_r	Return temp. floor heating Room 1	K
testhouse.T_room2_r	Return temp. floor heating Room 2	K
testhouse.T_room3_r	Return temp. floor heating Room 3	K
testhouse.T_room4_r	Return temp. floor heating Room 4	K
T_rt.T	Total return temp. floor heating	K
valve1_ctrl.y_state	Control signal valve Room 1 (range 0-1)	-
valve2_ctrl.y_state	Control signal valve Room 2 (range 0-1)	-
valve3_ctrl.y_state	Control signal valve Room 3 (range 0-1)	-
valve4_ctrl.y_state	Control signal valve Room 4 (range 0-1)	-
MZ11.opening_filtered	Position of valves Room 1 (range 0-1)	-
MZ21.opening_filtered	Position of valves Room 2 (range 0-1)	-
MZ3.opening_filtered	Position of valve Room 3 (range 0-1)	-
MZ4.opening_filtered	Position of valve Room 4 (range 0-1)	-
MZ11.m_flow	Mass flow through valve 1 Room 1	kg/s
MZ12.m_flow	Mass flow through valve 2 Room 1	kg/s
MZ13.m_flow	Mass flow through valve 3 Room 1	kg/s
MZ14.m_flow	Mass flow through valve 4 Room 1	kg/s
MZ21.m_flow	Mass flow through valve 1 Room 2	kg/s
MZ22.m_flow	Mass flow through valve 2 Room 2	kg/s
MZ23.m_flow	Mass flow through valve 3 Room 2	kg/s
MZ3.m_flow	Mass flow through valve Room 3	kg/s
MZ4.m_flow	Mass flow through valve Room 4	kg/s
bypass_valve.m_flow	Mass flow through bypass valve	kg/s
testhouse.V_door_12	Ventilation through door between Room 1 and 2	m^3/s
testhouse.V_door_23	Ventilation through door between Room 2 and 3	m^3/s
testhouse.V_door_41	Ventilation through door between Room 4 and 1	m^3/s
add13.y	Global solar radiation	W
hP.Power	Heat pump power consumption	W
hP.Q_hp	Delivered heat from heat pump to floor heating system	W
time	Simulation time	s

The Python package neurolab can be used to implement a neural network in Python for prediction purposes. Here is some example code to initialize the network from parameters derived in Matlab:

```
import neurolab as nl
import numpy as np
import scipy.io as sio

# Load neural network parameters (one hidden layer)
mat_contents = sio.loadmat('NN_pars.mat') # a matlab file with parameters
input_w = np.array(mat_contents['input_w'])
input_b = np.array(mat_contents['input_b'])
output_w = np.array(mat_contents['output_w'])
output_b = np.array(mat_contents['output_b'])
range_u = mat_contents['range_u']
range_y = mat_contents['range_y']
N_inputs = len(input_w[0])
N_neurons = len(input_b)
N_outputs = len(output_b)
input_b = np.reshape(input_b,N_neurons)
output_b = np.reshape(output_b,N_outputs)

# Create network with random initialization and insert pretrained network parameters
net = nl.net.newff([[-
1,1]]*N_inputs,[N_neurons,N_outputs],[nl.trans.TanSig(),nl.trans.PureLin()])
net.layers[0].np['w'][:] = input_w
net.layers[1].np['w'][:] = output_w
net.layers[0].np['b'][:] = input_b
net.layers[1].np['b'][:] = output_b
```

The network can then be used by calling the function net.sim([input_vec]), where input_vec is an array containing all the inputs to the neural network at a given discrete time instance and the output is, e.g., the four predicted room temperatures on step ahead in time.

D.5 Simulation Example

A few simulation result plots from Dyomla is shown in Figure D.8 – D.10. The chosen simulation model is BR1970_house_with_hp_more_valves and the first two days of January is plottet.

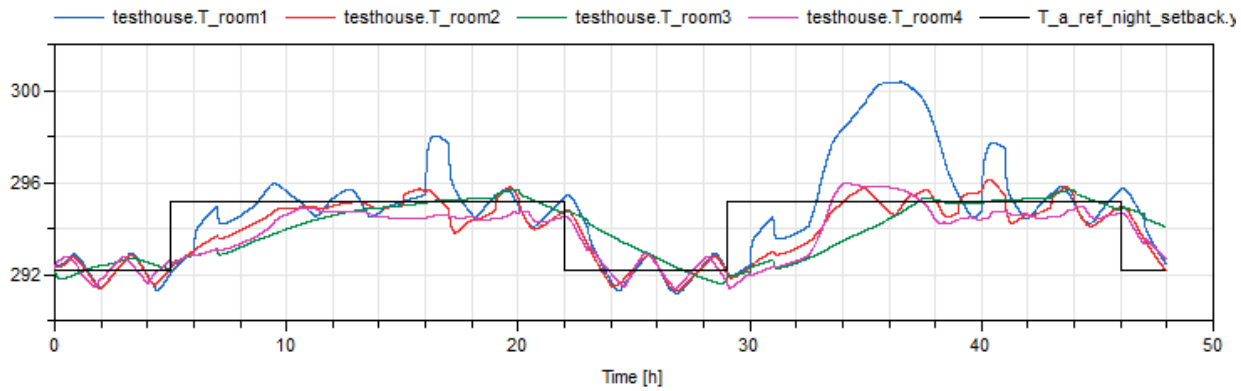

Figure D.8: Room temperatures in Kelvin and desired room temperature with night setback from 22.00-05.00 (black line). The temperatures are controlled with hysteresis type control (on/off), which gives the "sawtooth" behavior. Room 3 has concrete flooring and is dynamically slower in terms of tracking the reference setpoint. The temperature in Room 1 is disturbed by solar heat during day two, due to large window sections, and has dinner cooking peaks from increased internal heat gain from appliances (kitchen part of Room 1).

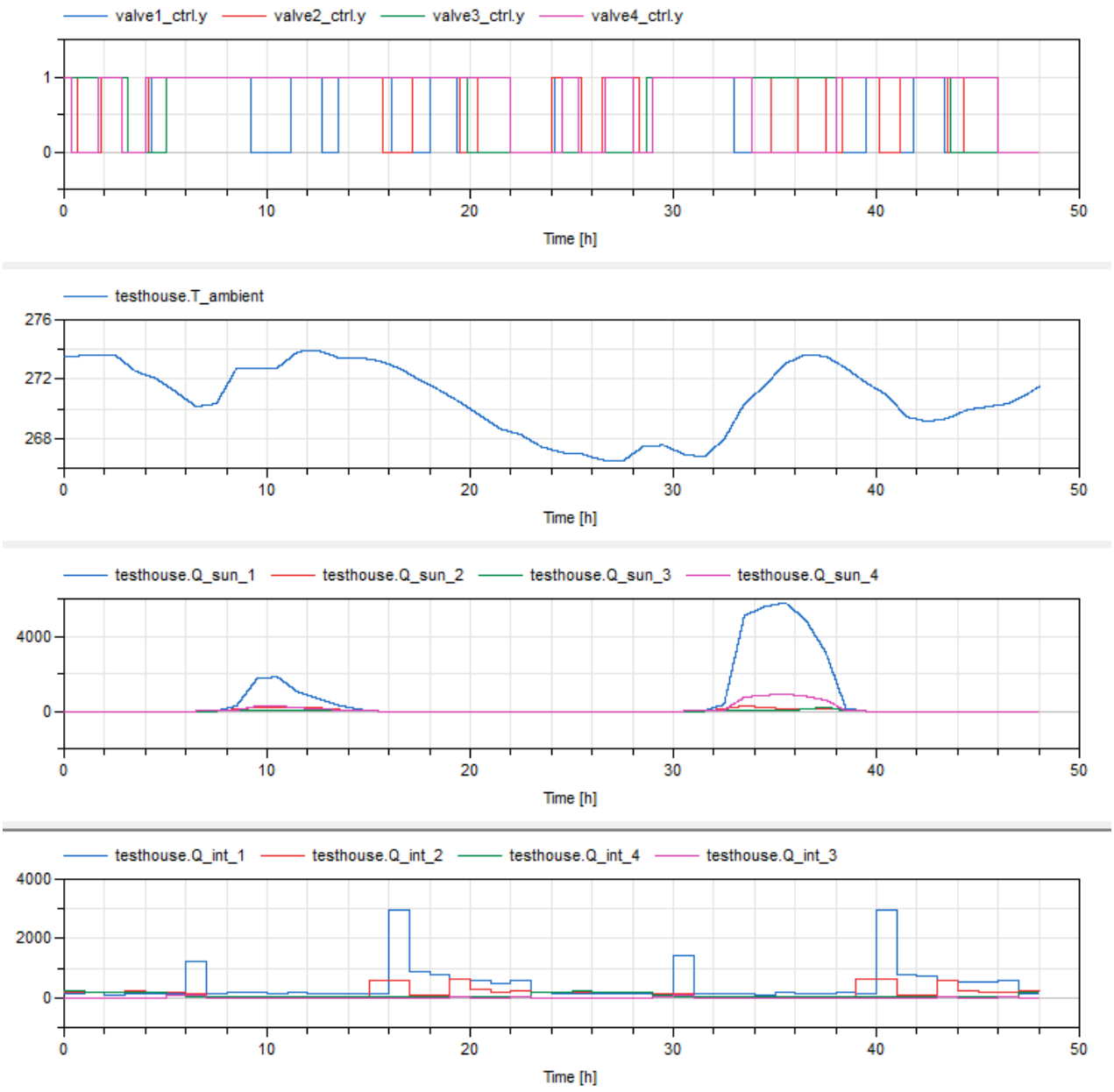

Figure D.9: Valve on/off control signal, ambient temperature, solar heat gains into each room, and internal heat gains from people and appliances.

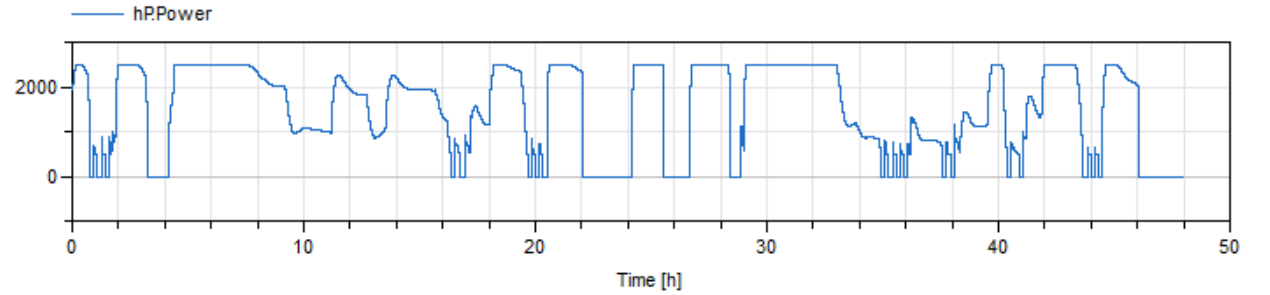

Figure D.10: Heat pump power consumption. The heat pump is heavily saturated when the house needs to be warmed up after night-setback. Saturation also occurs due to synchronization of the heat demand from each room. On/off operation of the heat pump typically occurs if only one of the rooms has a heat demand.