ACT Project Number:
271498

Project acronym:
ELEGANCY

Project full title:
Enabling a Low-Carbon Economy via Hydrogen and CCS

ERA-Net ACT project

Starting date: 2017-08-31
Duration: 36 months

D4.5.2
Model documentation and user manual for the dynamic operational tool

Issue 1

Actual DELIVERY date: 2020-10-20

Organization name of lead participant for this deliverable:
Imperial College London

ACT ELEGANCY, Project No 271498, has received funding from DETEC (CH), BMWi (DE), RVO (NL), Gassnova (NO), BEIS (UK), Gassco, Equinor and Total, and is cofunded by the European Commission under the Horizon 2020 programme, ACT Grant Agreement No 691712.

Dissemination Level		
PU	Public	X
CO	Confidential, only for members of the consortium (including the Commission Services)	

Deliverable number:	D4.5.1
Deliverable title:	Model documentation and user manual for the operational toolkit – final version.
Work package:	WP4
Lead participant:	Imperial College London (ICL)

Authors		
Name	**Organisation**	**E-mail**
Edward Graham	ICL	edward.graham10@imperial.ac.uk
Nixon Sunny	ICL	nixon.sunny13@imperial.ac.uk
Nilay Shah	ICL	n.shah@imperial.ac.uk
Julian Straus	SINTEF	Julian.straus@sintef.no

Keywords
Modelling tool-kit, Operational, Components, OpenModelica

Executive Summary
This document describes the dynamic models for units within a H_2 + CCS distribution network and shows how users can build and analyse distribution chains using these tools, developed as part of work package 4 of the ERA-Net ACT ELEGANCY project. The principal software used is the OpenModelica Connection editor, which is an open-source graphical interface that provides users with an intuitive way to develop unit models (e.g., pipes, storage facilities) and provide connections between various types of ports (e.g., fluid and heat ports). If there are any comments or general enquires, please contact edward.graham10@imperial.ac.uk .

TABLE OF CONTENTS

1 INTRODUCTION

This document details how users can build and analyse distribution chains using the tools developed as part of work package 4 of the ERA-Net ACT ELEGANCY project. The principal software used is the OpenModelica Connection editor, which is an open-source graphical interface that provides users with an intuitive way to develop unit models (e.g., pipes, storage facilities) and provide connections between various types of ports (e.g., fluid and heat ports).

We first describe the installation of software. Next, we describe the development and implementation of pure component physical property models within the Modelica language. We then give a brief overview of the model equations and assumptions used to model the various units within the distribution chain, along with how to specify initialization procedures. Finally, we show specific examples of how to build simple networks and define control strategies, and move on to an example of a more complex network. These examples are included within the toolkit to show the user precisely how to formulate different problems and the numerical method used for their solution.

If there are any comments or general enquires, please contact edward.graham10@imperial.ac.uk .

2 INSTALLATION

2.1 OPENMODELICA

We recommend that the latest version of OpenModelica is downloaded from https://www.openmodelica.org/download/download-windows . In case of compatibility issues, the specific version of OMEdit used to develop the operational toolkit is the official release of the Windows version at the time of writing: OpenModelica1.14.1-64bit.

The components of the OpenModelica toolkit are included in the package: “Operational_Toolkit_v2.mo”. Upon loading this library, the unit models and various examples shown in this user guide will be displayed in the library browser as follows:

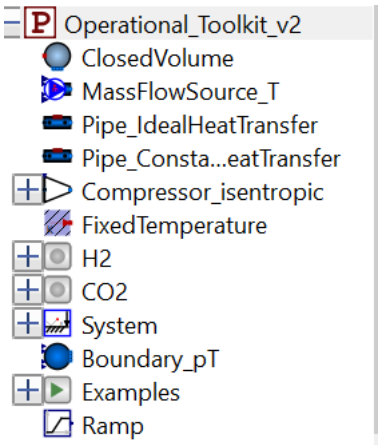


Figure 1 – Overview of models in the OpenModelica toolkit

3 PHYSICAL PROPERTY MODELS

In this section we describe how users should create a custom medium model that is compatible with the Modelica.Fluid library, we describe here how this should be done along with some specific examples.

3.1 GENERATING A CUSTOM MEDIUM MODEL

Developing custom medium models in the rigorous format required by the Modelica library is extremely useful since the medium model may be used in conjunction with the wealth of robust models available to Modelica. In this section we describe the steps required to do this, since the current documentation is relatively sparse.

The OpenModelica toolkit consists of custom medium models for pure hydrogen (package "H_2") and pure carbon dioxide (package "CO_2").

The user may wish to generate their own custom medium model for different components or models that are valid over a different range of thermodynamic conditions. This should be done as follows:

1. A new package should be loaded, and named according to the component or mixture, in this case, we are modelling "H2".
2. The independent variables should be specified: in this case we set the thermodynamic states to have independent variables p, T and x.
3. The minimum and maximum values of the pressure and temperature should then be set. Note that if the thermodynamic variables exceed these bounds during simulation, an assertion error is generated during run time, but this does not stop the simulation.
4. The molecular weight of the component should be declared.

The top of the medium package should therefore be defined as follows for hydrogen:

```
package H2
  // 6th order taylor model from consumet
  extends Modelica.Media.Interfaces.PartialPureSubstance(ThermoStates =
Modelica.Media.Interfaces.Choices.IndependentVariables.pTX, singleState =
false, Temperature(min = 273.15, max = 303.15, start = 300),
AbsolutePressure(min = 1e6, max = 40e6, start = 6e6));
  constant MolarMass MM_const = 2.016e-3 "Molar mass";
  // kg/mol
```

The 'BaseProperties' class should be redeclared and functions for the various thermodynamic variables inputted. Since this is a T,p model, we write the density and enthalpy as a function of temperature and pressure. The output of CONSUMET is in terms of the reduced variables, thus, we define new reduced variables Tr and Pr at the top of the function and write d(Tr, Pr) and h(Tr, Pr). Furthermore, we require the relation for internal energy (u = h-p/d) and statements defining temperature, pressure, molecular weight and the ideal gas constant on a mass basis. The base properties model is then simply:

```
  redeclare model extends BaseProperties(T(stateSelect = if preferredMediumStates then
StateSelect.prefer else StateSelect.default), p(stateSelect = if preferredMediumStates
then StateSelect.prefer else StateSelect.default)) "Base properties of medium"
      Real Pr, Tr;

    equation
      Pr = (p * 1e-6 - 1) / (40 - 1);
      Tr = (T - 273.15) / (303.15 - 273.15);
//d = p/R/T;
      d = MM * (437.623 + Pr ^ 1.0 * 16961.04 + Pr ^ 2.0 * (-4056.23) + Pr ^ 3.0 *
207.1778 + Pr ^ 4.0 * 433.718 + Pr ^ 5.0 * (-170.5648) + Pr ^ 6.0 * 12.28372 + Tr ^ 1.0
* (-48.09765) + Tr ^ 1.0 * Pr ^ 1.0 * (-1854.212) + Tr ^ 1.0 * Pr ^ 2.0 * 714.9401 + Tr
^ 1.0 * Pr ^ 3.0 * 107.2221 + Tr ^ 1.0 * Pr ^ 4.0 * (-219.7232) + Tr ^ 1.0 * Pr ^ 5.0 *
66.36184 + Tr ^ 2.0 * 6.403281 + Tr ^ 2.0 * Pr ^ 1.0 * 200.4548 + Tr ^ 2.0 * Pr ^ 2.0 *
(-101.5664) + Tr ^ 2.0 * Pr ^ 3.0 * (-2.253024) + Tr ^ 2.0 * Pr ^ 4.0 * 11.53657 + Tr ^
3.0 * (-1.839775) + Tr ^ 3.0 * Pr ^ 1.0 * (-11.80001) + Tr ^ 3.0 * Pr ^ 2.0 * 3.362076
+ Tr ^ 4.0 * Pr ^ 1.0 * (-2.709656) + Tr ^ 4.0 * Pr ^ 2.0 * 2.924457);
      h = (7213.956 + Pr ^ 1.0 * 281.7351 + Pr ^ 2.0 * 176.4006 + Pr ^ 4.0 *
(-38.61002) + Pr ^ 6.0 * 8.269283 + Tr ^ 1.0 * 861.3027 + Tr ^ 1.0 * Pr ^ 1.0 *
72.27537 + Tr ^ 1.0 * Pr ^ 2.0 * (-38.51922) + Tr ^ 1.0 * Pr ^ 4.0 * 5.739548 + Tr ^
2.0 * 3.226658 + Tr ^ 2.0 * Pr ^ 1.0 * (-2.796348) + Tr ^ 3.0 * Pr ^ 1.0 * (-1.343903)
+ Tr ^ 3.0 * Pr ^ 3.0 * 1.150659) / MM;
      u = h - p / d;
      p = state.p;
      T = state.T;
      MM = MM_const;
      R = 8.3144 / MM;
  end BaseProperties;
```

Other medium properties may then be defined by redeclaring a number of functions. For example, the dynamic viscosity:

```
  redeclare function extends dynamicViscosity "Return specific heat capacity at
constant volume"
    protected
      Real Tr, Pr;

    algorithm
      Pr := (state.p * 1e-6 - 0.01) / (70 - 0.01);
      Tr := (state.T - 273.15) / (303.15 - 273.15);
      eta := 1e-6 * (8.376351 + Pr ^ 1.0 * 0.4960503 + Pr ^ 2.0 * 4.32647 + Pr ^ 3.0 *
(-4.093603) + Pr ^ 4.0 * 2.328414 + Pr ^ 5.0 * (-0.7374059) + Pr ^ 6.0 * 0.09694489 +
Tr ^ 1.0 * 0.6374942 + Tr ^ 1.0 * Pr ^ 1.0 * (-0.1703525) + Tr ^ 1.0 * Pr ^ 2.0 *
(-0.711275) + Tr ^ 1.0 * Pr ^ 3.0 * 0.9779357 + Tr ^ 1.0 * Pr ^ 4.0 * (-0.5781216) + Tr
^ 1.0 * Pr ^ 5.0 * 0.1318107 + Tr ^ 2.0 * (-0.01372378) + Tr ^ 2.0 * Pr ^ 1.0 *
0.03534364 + Tr ^ 2.0 * Pr ^ 2.0 * 0.06870629 + Tr ^ 2.0 * Pr ^ 3.0 * (-0.101565) + Tr
^ 2.0 * Pr ^ 4.0 * 0.04035988 + Tr ^ 3.0 * 0.007522214 + Tr ^ 3.0 * Pr ^ 1.0 *
(-0.01337785) + Tr ^ 3.0 * Pr ^ 2.0 * 0.007104198 + Tr ^ 3.0 * Pr ^ 3.0 * 0.0008715189
+ Tr ^ 4.0 * (-0.01006183) + Tr ^ 4.0 * Pr ^ 1.0 * 0.00413409 + Tr ^ 4.0 * Pr ^ 2.0 *
(-0.003308206) + Tr ^ 5.0 * 0.007695032 + Tr ^ 6.0 * (-0.002408013));
  end dynamicViscosity;
end H2;
```

Note that ‘protected’ is required, and the reduced variables must be defined at the top of the algorithm section.

As part of the medium package, we also include the function ‘setState_phX’, which allows us to determine medium properties with p and h as independent variables. This is done by determining temperature as an explicitly function of enthalpy and pressure and including this in the ThermodynamicSate class:

```
  redeclare function extends setState_phX "Set the thermodynamic state record from p
and h (X not needed)"
    protected
      Real pr, hr;

    algorithm
      pr := (p * 1e-6 - 1) / (40 - 1);
      hr := (h * 2.016e-3 - 7214.060) / (8542.957 - 7214.060);
      state := ThermodynamicState(p = p, T = 273.1497 + pr ^ 1.0 * (-9.818327) + pr ^
2.0 * (-4.822566) + pr ^ 3.0 * (-2.528819) + pr ^ 4.0 * 5.963713 + pr ^ 5.0 *
(-4.083106) + pr ^ 6.0 * 1.037863 + hr ^ 1.0 * 46.32792 + hr ^ 1.0 * pr ^ 1.0 *
(-4.115097) + hr ^ 1.0 * pr ^ 2.0 * 3.076093 + hr ^ 1.0 * pr ^ 3.0 * (-0.8564101) + hr
^ 1.0 * pr ^ 4.0 * (-0.4089363) + hr ^ 1.0 * pr ^ 5.0 * 0.2853492 + hr ^ 2.0 *
(-0.3475331) + hr ^ 2.0 * pr ^ 1.0 * 0.8080364 + hr ^ 2.0 * pr ^ 2.0 * (-0.9458613) +
hr ^ 2.0 * pr ^ 3.0 * 0.5194099 + hr ^ 2.0 * pr ^ 4.0 * (-0.1275396) + hr ^ 3.0 *
0.03261113 + hr ^ 3.0 * pr ^ 1.0 * (-0.09893147) + hr ^ 3.0 * pr ^ 2.0 * 0.1350867 + hr
^ 3.0 * pr ^ 3.0 * (-0.04398887) + hr ^ 4.0 * 0.0514136 + hr ^ 5.0 * (-0.03577434));
  end setState_phX;
```

For thermodynamic consistency, one should compute T(p,h) analytically from the function h(T,p). However, and analytic solution is not always available. In this case, one should be careful using the medium model within a closed loop cycle to avoid enthalpy generation/loss.

3.2 GENERATING MODELS FROM THE OUTPUT OF CONSUMET

The file named “CONSUMET_to_Modelica.py” shows an example of how to automatically generate the Modelica functions to be used within the medium package. This function reads the output of CONSUMET (“regression.csv”) and prints the corresponding Taylor series function to the command line which may be copied and pasted into the Modelica script. This function is relatively simple, where the user only needs to specify the property type required (by changing imodel), and naming the reduced variables (Tr and Pr).

```python
import numpy as np

data_regression = np.genfromtxt('regression.csv', delimiter=',')

#input_lb    = [273.15,  1.0]
#input_ub    = [303.15, 40.0]
 #   modelString = ['Density', 'Enthalpy', 'Entropy', 'Viscosity']

# Specify the index of the property type
imodel = 3;

string = str()
for row in data_regression:
    if row[0] == imodel:
        if row[-1] != 0.0:
            tterm = "Tr^" + str(row[1]) + "*" if row[1] > 0 else str() # Tr is the first variable
            pterm = "Pr^" + str(row[2]) + "*" if row[2] > 0 else str() # Pr is the second variable
            coeff = "(" + str(row[3]) + ")"
            if row[3] != 0:
                string = string + tterm + pterm + coeff + "+"

string = string[:-1]
print(string)
```

4 UNIT MODELS

In this section we give a brief overview of the various unit models and how they may be connected via fluid and heat ports.

4.1 NOMENCLATURE

Table 1: Description of variables

x	Independent spatial coordinate (flow is along the x coordinate)
t	Time
$v(x,t)$	Mean velocity
$p(x,t)$	Mean pressure
$T(x,t)$	Mean temperature
$\rho(x,t)$	Mean density
$u(x,t)$	Specific internal energy
$z(x,t)$	Height above ground
$A(x)$	Area perpendicular to the x coordinate
g	Gravitational constant
f	Fanning friction factor
S	Circumference of pipe
W_c	Compressor work
η	Isentropic efficiency

4.2 FLUID CONNECTORS

The transfer of mass between various process units is modelled by defining fluid connectors and fluid ports. In Figure 2 we detail the code used for fluid connectors. A fluid is defined by its medium package (for accessing physical properties via, e.g., an equation of state). The mass flowrate is defined by m_flow ($\dot{m}$, a **flow** variable). The thermodynamic state of the fluid is then fully defined by the **stream** variables (intrinsic properties): pressure (p), specific enthalpy (h), and a vector of mass fractions ($\boldsymbol{m}$).

The use of fluid connectors is useful for the robust numerical solution of the Modelica model. In particular, discontinuities are avoided when flow reversal occurs, (e.g., discontinuities in the flow of enthalpy). The use of connector and ports is also useful for intuitively building different distribution networks, since various units (storage tanks, pipes, compressors …) can be connected in a 'drag/ drop' manner within the OpenModelica interface.

```
connector FluidPort
  "Interface for quasi one-dimensional fluid flow in a piping network
   (incompressible or compressible, one or more phases, one or more substances)"

  replaceable package Medium = Modelica.Media.Interfaces.PartialMedium
    "Medium model" annotation (choicesAllMatching=true);

  flow Medium.MassFlowRate m_flow
    "Mass flow rate from the connection point into the component";
  Medium.AbsolutePressure p "Thermodynamic pressure in the connection point";
  stream Medium.SpecificEnthalpy h_outflow
    "Specific thermodynamic enthalpy close to the connection point if m_flow < 0";
  stream Medium.MassFraction Xi_outflow[Medium.nXi]
    "Independent mixture mass fractions m_i/m close to the connection point if m_flow < 0";
  stream Medium.ExtraProperty C_outflow[Medium.nC]
    "Properties c_i/m close to the connection point if m_flow < 0";
end FluidPort;
```

Figure 2 - Modelica code for connectors for specification of fluid flow between units.

4.3 DYNAMIC PIPE

In this section we outline the dynamic model for pipelines in Modelica. This model extends from the standard Modelica library (Modelica.Fluid.Pipes.DynamicPipe). The equations and variables used in this model are detailed here. In Table 1 we define the variables used in this section. All variables are in S.I. units.

4.4 MODEL EQUATIONS

The pipe model is derived from distributed mass, energy and momentum balances. The flow is taken as one-dimensional, i.e., the governing variables are considered to depend on only one spatial coordinate, x, and in general depend on time, t. The mass balance across a differential volume leads to the partial differential equation

$$\frac{\partial(\rho A)}{\partial t} + \frac{\partial(\rho A v)}{\partial x} = 0 \tag{1}$$

The momentum balance is given by

$$\frac{\partial(\rho v A)}{\partial t} + \frac{\partial(\rho v^2 A)}{\partial x} = -A\frac{\partial p}{\partial x} - F_F - A\rho g\frac{\partial z}{\partial x} \tag{2}$$

The energy balance is given by

$$\frac{\partial(\rho u A)}{\partial t} + \frac{\partial\left(\rho v\left(u + \frac{p}{\rho}\right)A\right)}{\partial x} = vA\frac{\partial p}{\partial x} + vF_F + \frac{\partial}{\partial x}\left(kA\frac{\partial T}{\partial x}\right) + \dot{Q}_e \tag{3}$$

where the frictional loss term is given as a function of the Fanning friction factor, f.

$$F_F = \frac{1}{2}\rho v|v| f S \tag{4}$$

4.4.1 PIPE WITH HEAT TRANSFER

Heat transfer to the environment may be imposed by connecting the heat port in each discretized pipe section to an external heat port representing the environment. The code to make these connections is shown in Figure 4.

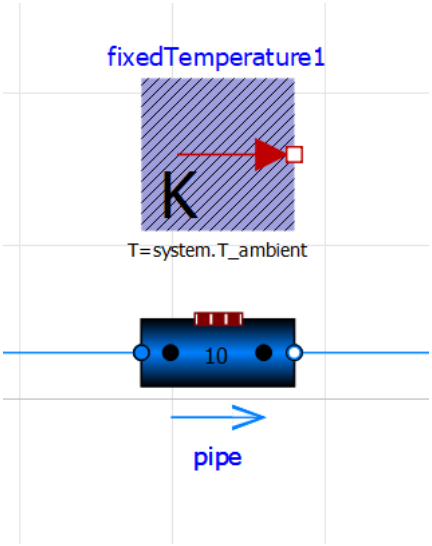


Figure 3 - Schematic for the dynamic pipe model as viewed in the OpenModelica interface. The heat port connections are not shown in the interface, but connections are made within the code.

```
for i in 1:pipe.nNodes loop
  connect(fixedTemperature1.port, pipe.heatPorts[i]);
end for;
```

Figure 4 - Code for connecting heat ports to pipe sections for modelling an isothermal pipe

Two models are included in the operational toolkit:

- Pipe_IdealHeatTransfer: Ideal heat transfer is assumed such that the temperature of each section in the pipe is equal to that of the environment.
- Pipe_ConstantFlowHeatTransfer: A constant overall heat transfer coefficient is imposed. In this case the user should specify the alpha_0 value in the code as follows:

```
  extends Modelica.Fluid.Pipes.DynamicPipe(redeclare model HeatTransfer =
Modelica.Fluid.Pipes.BaseClasses.HeatTransfer.ConstantFlowHeatTransfer(alpha0 = 0.34));
```

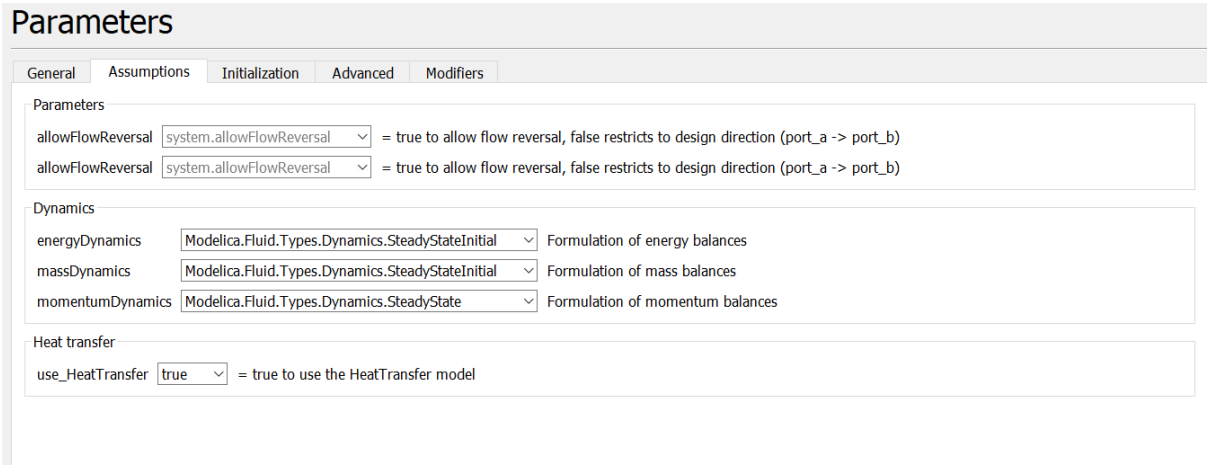
In this case, $\dot{Q}_e$ in equation (3) is modelled as:

$$\dot{Q}_e^i = \alpha_0 S^i (T^i - T_{amb}) \tag{3}$$

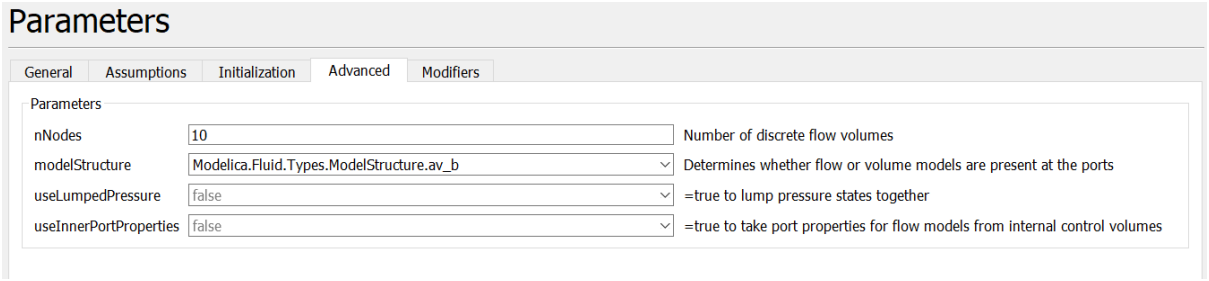
Where S^i is the surface area of the i^{th} section of the pipe.

4.4.2 INITIALIZATION OF THE PIPE

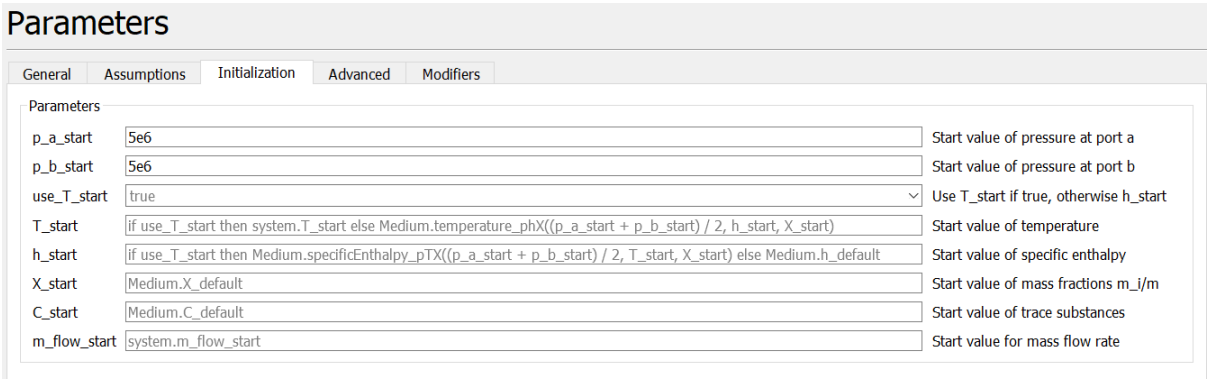
It is important to properly initialize the Modelica model, particularly for large systems. Initialization procedures for the pipes may be chosen on the “Assumptions” tab. Note that a useful feature of the dynamic pipe model is that the energy, mass and momentum balances are decoupled, so a different initialization procedure may be used for each equation. In the following, the energy and mass dynamics are initially at steady state, and converge to the dynamic solution using the homotopy method available to Modelica. The momentum dynamics are assumed to be steady state in the example below, which is a fairly good assumption for most modelling conditions:



Furthermore, the number of discretized pipe sections used in the numerical integration should be specified. Note that a larger the number of nodes leads to a more accurate integration but the simulation time is significantly increased by increasing this parameter. The model structure should be chosen carefully, depending on whether the pipe ports are connected to volume (e.g., storage), or flow (e.g., valve) units. These considerations should be specified in the advanced tab:



Good initial guesses for the thermodynamic variables should also be specified on the dynamic tab:



We found that a good initial guess is to specify that the inlet and outlet pressures are equal, leading to a zero mass flowrate at the start of the simulation.

4.5 STORAGE FACILITIES

To model the storage facilities, the 'ClosedVolume' model should be used. This extends from 'Modelica.Fluid.ClosedVolume. The model is shown in Figure 5 as viewed in the OpenModelica interface. This model represents an ideally mixed storage facility of constant volume. Heat transfer with the environment may be imposed similar to the dynamic pipe in the section above. This connector is shown by the red line in Figure 5. The blue lines in Figure 5 represent the connections between fluid ports (representing input and output streams in either direction) via the fluid connector as detailed in section 8.

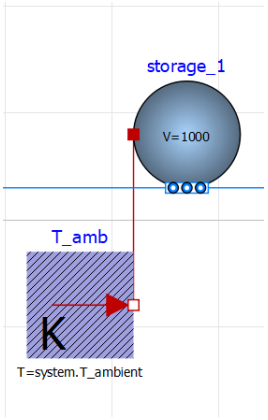


Figure 5 - Model for storage facilities. The blue lines represent fluid connectors. The red line represents the transfer of heat to the environment.

4.6 COMPRESSOR MODEL

Compressors are required in order to control the mass flow through the pipes by adjusting the pressure driving force. The model considers an isentropic compression stage whilst the work is calculated by applying an efficiency factor. Note that a multi-stage compressor may be modelled by connecting these stages in series with volumes and heat sinks inbetween, as shown in the example section entitled "defining control strategies".

The work required to drive the compressors is adjusted for the isentropic efficiency, η. Subscripts 'in' and 'out' represent variables associated with input and output streams respectively.

Assuming no accumulation of mass within the compressor.

$$\dot{m}_{in} + \dot{m}_{out} = 0 \tag{5}$$

The specific entropy of the fluid entering and exiting the compressor is held constant.

$$s_{in}(p_{in}, h_{in}) + s_{out}(p_{out}, h_{out}) = 0 \tag{6}$$

We define the compressor ratio, p^r (equivalently we could specify the compressor work). This variable may be used within a control scheme.

$$\frac{p_{out}}{p_{in}} = p^r \tag{7}$$

The compressor work is given by the energy balance

$$W_c = \frac{\dot{m}_{in}(h_{out} - h_{in})}{\eta} \tag{8}$$

where we apply an isentropic efficiency to deal with entropy losses. This parameter may be specified in the parameter dialogue box.

The above equations are implemented using the following Modelica code:

```
model Compressor_isentropic
  replaceable package Medium = Modelica.Media.Interfaces.PartialPureSubstance;
  Modelica.Fluid.Interfaces.FluidPort_a port_a(redeclare package Medium = Medium) annotation( ...);
  Modelica.Fluid.Interfaces.FluidPort_b port_b(redeclare package Medium = Medium) annotation( ...);
  Modelica.Blocks.Interfaces.RealInput P_ratio annotation( ...);
  Medium.ThermodynamicState state1, state2;
  parameter Real eta = 1 "compressor efficiency";
  Modelica.SIunits.Power W;
equation
// Mass balance
  port_a.m_flow + port_b.m_flow = 0;
// Get thermodynamic states from ports. Note that for the stream flowing out of the compressor, inStream is
needed for flow in the design direction
  state1 = Medium.setState_ph(port_a.p, port_a.h_outflow);
  state2 = Medium.setState_ph(port_b.p, port_b.h_outflow);
// Assume isentropic compression
  Medium.specificEntropy(state1) = Medium.specificEntropy(state2);
// Fix compression ratio
  if port_a.m_flow > 0 then
    port_b.p / port_a.p = P_ratio;
  else
    port_b.p / port_a.p = 1;
// don't decompress
  end if;
// Equation for flow in reverse direction (no work in the reverse direction)
  port_a.h_outflow = inStream(port_b.h_outflow);
// Equation for compressor work
//  W = port_a.m_flow * (inStream(port_b.h_outflow) - port_a.h_outflow) / eta;
  W = port_a.m_flow * (port_b.h_outflow - port_a.h_outflow) / eta;
// W = 10000;
  annotation( ...);
end Compressor_isentropic;
```

Note that if fluid flows in the reverse direction (port_b to port_a), the pressure ratio defaults to 1. The states of the ports are defined using p, h as independent variables, for which functions are defined explicitly in the medium model.

5 USER GUIDE

In this section we will show how users may use the toolkit to build and control distribution networks.

5.1 SPECIFICATION OF H_2 SUPPLY

- Right click on the project > New Modelica Class > Specialization: Model

This opens a blank screen on the diagram view. A mass flow source can be specified by dragging and dropping the 'Modelica.Fluid.Sources.MassFlowSource_T' icon onto the canvas. Parameters can be set by double clicking on the icon. In Figure 6 we specify 'use_m_flow_in', meaning that the production rate of H_2 is taken from a real input connector (triangles to the left of the H2_source icon). nPorts is set to 1, i.e., the H_2 source has a single output stream. The temperature is set to system.T_ambient. Global system variables may be set by dragging the Modelica.Fluid.System block onto the canvas.

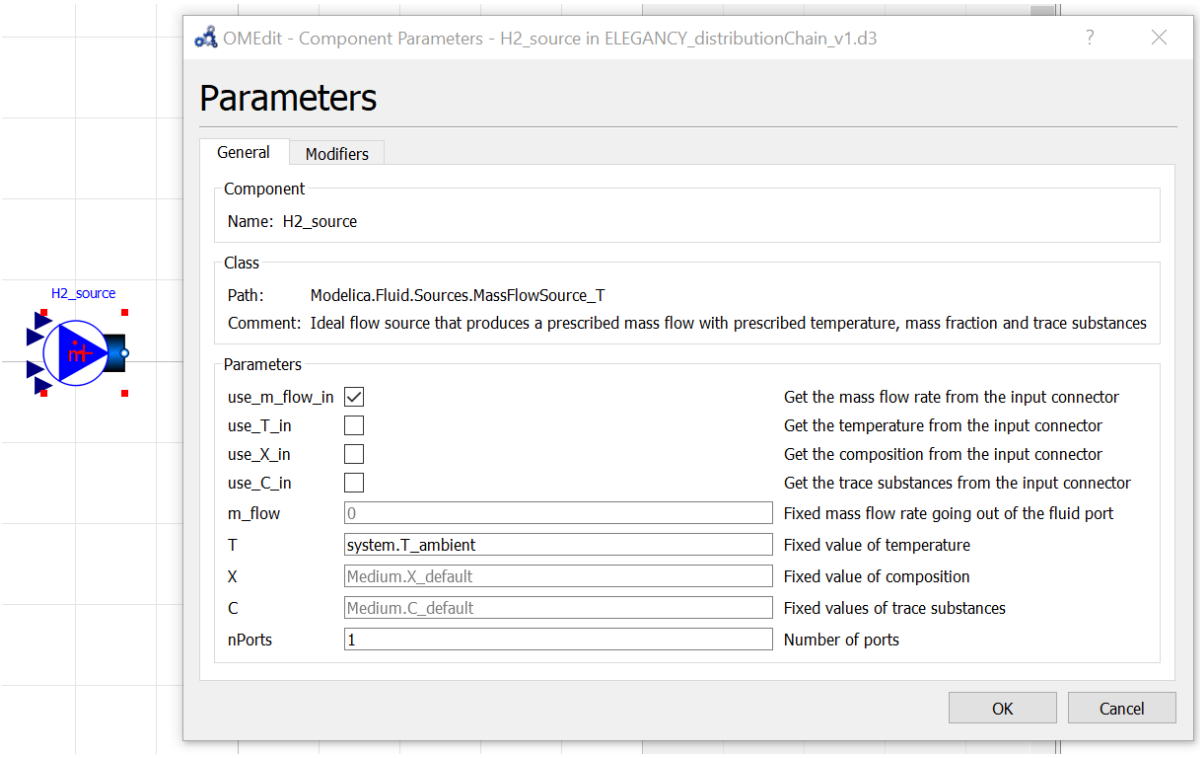


Figure 6 - Specification of unit parameters for H_2 source

The next step is to specify the Medium model to be used. This step is required for specifying the physical property method for all unit models. Within the 'Text view' tab for the distribution network, the user should specify the medium model as a replaceable package, and then redeclare this package within each unit model. An ideal gas property model for H2 can be specified with the following code:

```
model distributionNetwork
  replaceable package Medium = Modelica.Media.IdealGases.SingleGases.H2;
  Modelica.Fluid.Sources.MassFlowSource_T H2_prod(redeclare package Medium = Medium, T = system.T_ambient,
nPorts = 1, use_m_flow_in = true) annotation(
    Placement(visible = true, transformation(origin = {-96, 6}, extent = {{-10, -10}, {10, 10}}, rotation =
0)));
```

The custom medium model developed in this work may be used by swapping 'Modelica.Media.IdealGases.SingleGases.H2' by 'H2' in the project folder.

5.2 MAKING FLUID CONNECTIONS BETWEEN UNITS

Units can be connected by connecting the fluid ports in the OpenModelica interface. A simple example is shown in Figure 7, where we connect the H2 source (H2_prod) with a storage facility (storage_1). The blue line connecting the two units represents a fluid connection, where the variables that are transferred are the mass flowrate, specific enthalpy and the pressure of the stream. Connections can be made by dragging the mouse between fluid connectors.

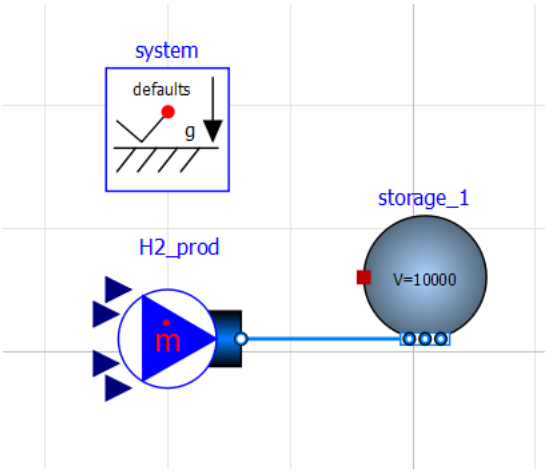
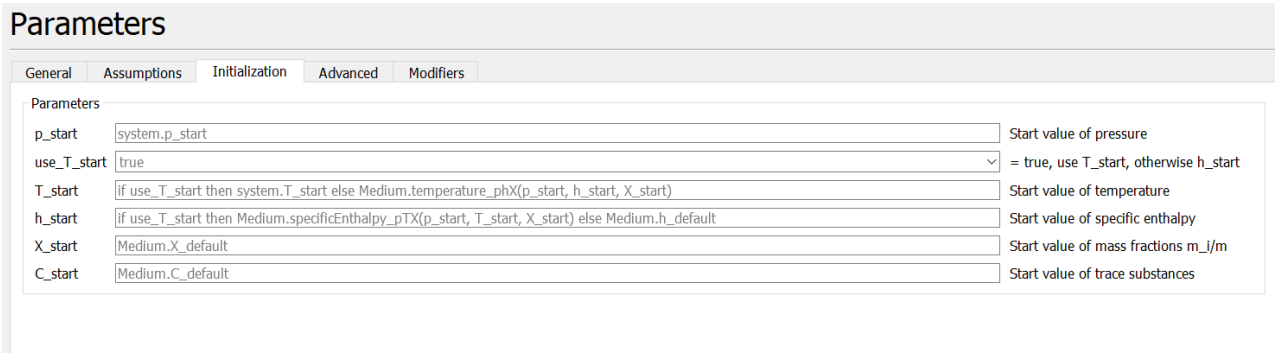


Figure 7 - Connecting H_2 source with storage facility

The model presented in Figure 7 consists of a flow source (H2_prod) which is fed into a closed volume representing the storage facility (Modelica.Fluids.Vessels.ClosedVolume). storage_1 has a heat port (red square). Since this is not connected, storage_1 is adiabatic. In this model, mass and energy accumulates in the storage facility, leading to an increased pressure and temperature within the storage tank. After specifying the parameters and connecting ports, OpenModelica generates the following code:

```
model distributionNetwork2
  replaceable package Medium = Modelica.Media.IdealGases.SingleGases.H2;
Modelica.Fluid.Sources.MassFlowSource_T H2_prod(redeclare package Medium =
Medium, T = system.T_ambient, m_flow = 1, nPorts = 1, use_m_flow_in = false)
annotation(
    Placement(visible = true, transformation(origin = {-38, 2}, extent =
{{-10, -10}, {10, 10}}, rotation = 0)));
Modelica.Fluid.Vessels.ClosedVolume storage_1(redeclare package Medium =
Medium, V = 10000, nPorts = 1, use_portsData = false)  annotation( ...);
inner Modelica.Fluid.System system annotation(
    Placement(visible = true, transformation(origin = {-40, 36}, extent =
{{-10, -10}, {10, 10}}, rotation = 0)));
equation
  connect(H2_prod.ports, storage_1.ports) annotation(
    Line(points = {{-28, 2}, {0, 2}, {0, 2}, {2, 2}}, color = {0, 127, 255},
thickness = 0.5));
end distributionNetwork2;
```

Note that the Medium package is redeclared for each unit. The equation 'use_portsData = false' is a required parameter to set for the storage tank. This means that pressure losses and kinetic energy terms are neglected to model the ports (otherwise, relations need to be defined explicitly). The number of ports (nPorts) and the volume (V) also need to be specified. Furthermore, since this is a dynamic model, the initial pressure and temperature should be set for storage_1. This can be done by specifying initial variables within the 'Initialization' tab of the interface. By default, initial values are taken from those defined in the system block.



An isothermal storage facility (i.e., ideal heat transfer to ambient) can be modelled by simply connecting a fixed temperature source, Modelica.Thermal.HeatTransfer.Sources.FixedTemperature, to the storage tank heat port, as shown in Figure 8.

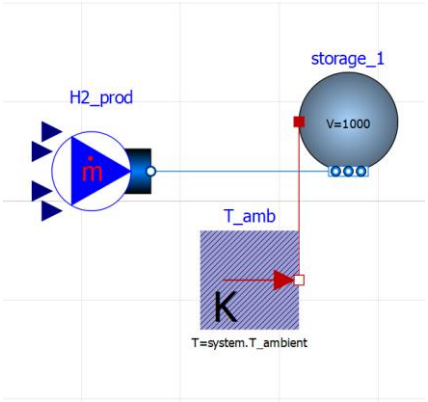


Figure 8 - Isothermal storage facility

5.3 ADDING COMPRESSORS

A compressor unit can be added to the model by dragging and dropping 'compressor' in the interface and connecting the fluid ports. The isentropic efficiency, eta, should be specified as a parameter. The compression ratio is defined as a real input such that it can be incorporated in control strategy. In the example shown in Figure 9, the compression ratio is modelled as a linear ramp using the 'Modelica.Blocks.Sources.Ramp' block.

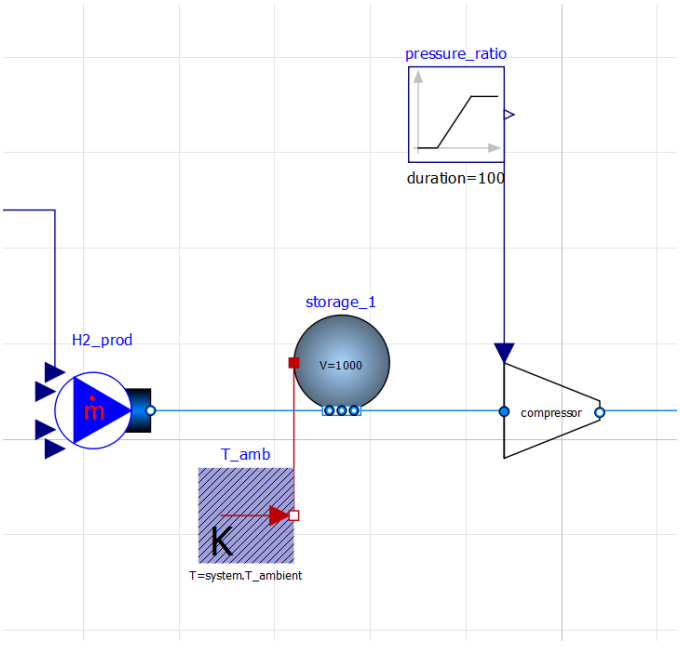


Figure 9 - Model for H_2 source, isothermal storage and compression

5.4 CONNECTING A DYNAMIC PIPE

In Figure 10 we show how a dynamic pipe can be incorporated the chain model. This is done by dragging the Modelica.Fluid.Pipes.DynamicPipe block into the interface. And connecting the fluid ports.

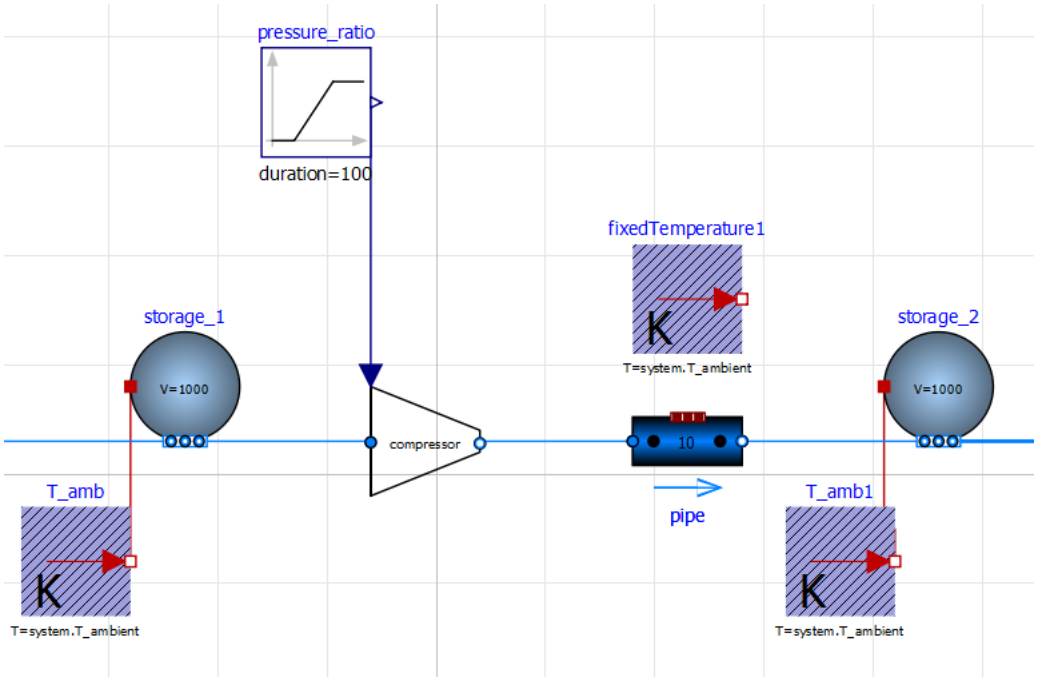


Figure 10 - Incorporating a dynamic pipe model

To model an isothermal pipe, the fixedTemperature1 source should be connected to all discretized nodes of the pipe using the following code

```
for i in 1:pipe.nNodes loop
  connect(fixedTemperature1.port, pipe.heatPorts[i]);
end for;
```

The pipe length, diameter and roughness may be set within the General tab as shown in Figure 11. The number of discretized sections of the pipe used for numerical solution can be specified in the Advanced tab (Figure 12). It was found that nNodes=10 provides a good balance between accuracy and speed for a 10 km pipe. A suitable model structure should be specified in the advanced tab (av_vb, a_v_b, av_b or a_vb) to avoid difficulties in numerical solution such as nonlinear algebraic equations. The reader is referred to [1] for details on choosing the model structure.

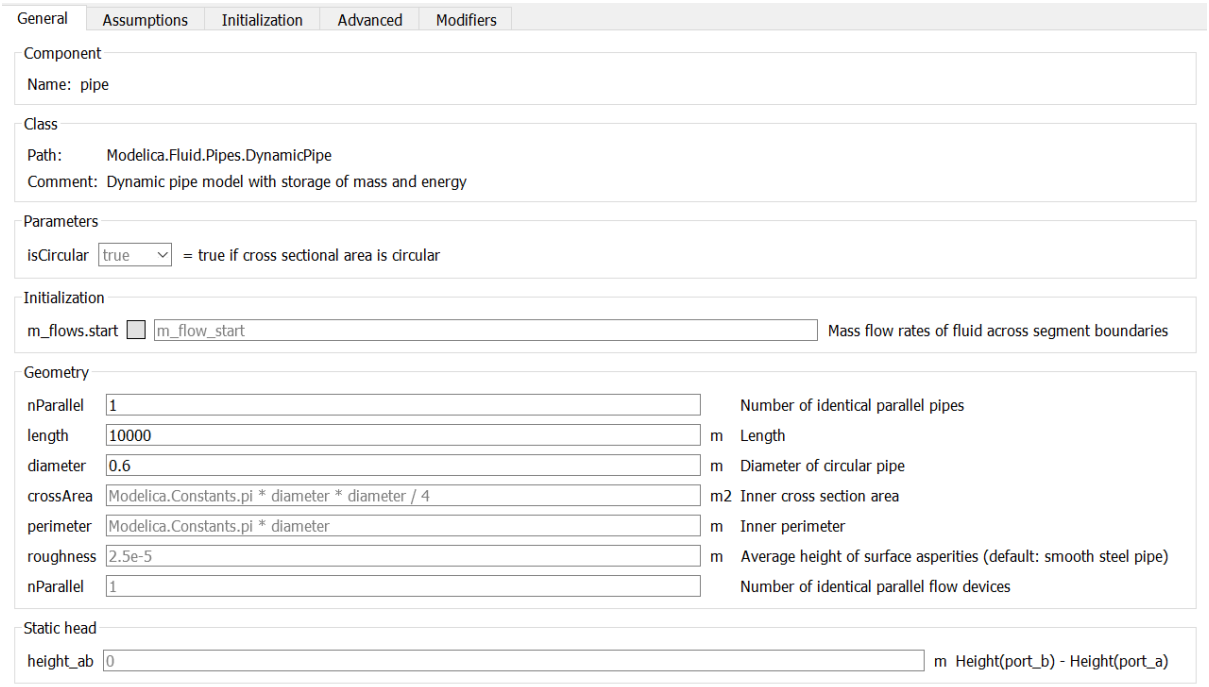


Figure 11 - Specification of pipe parameters

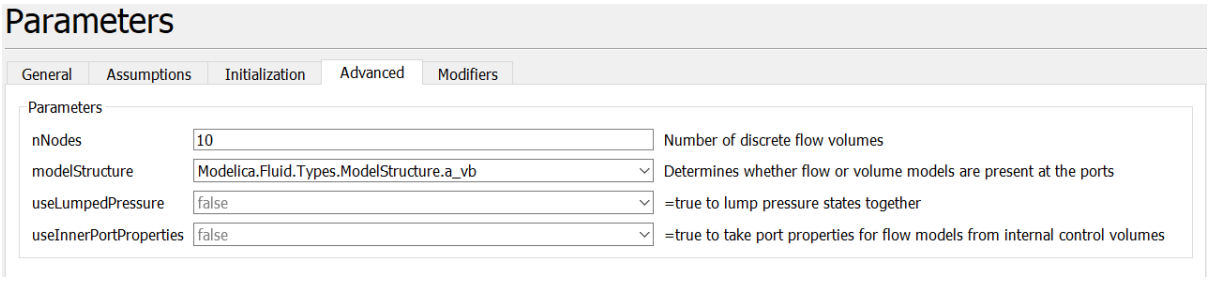


Figure 12 - Advanced pipe parameters

5.5 SPECIFICATION OF A TIME VARYING DEMAND SIGNAL

5.5.1 FROM A LOOK-UP TABLE

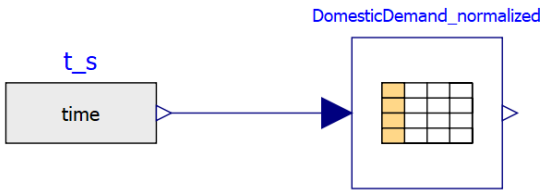
This example is shown in the model “Examples.DomesticDemand”.

A time-varying hydrogen demand signal may be incorporated into the Modelica simulation by reading data from an external .txt file and interpolating the data such that it has a smooth continuous derivative. Note that this method computationally efficient since it avoids discontinuities during simulation time, and memory is dynamically allocated so that the whole data set does not need to be loaded during compilation. The following steps should be taken:

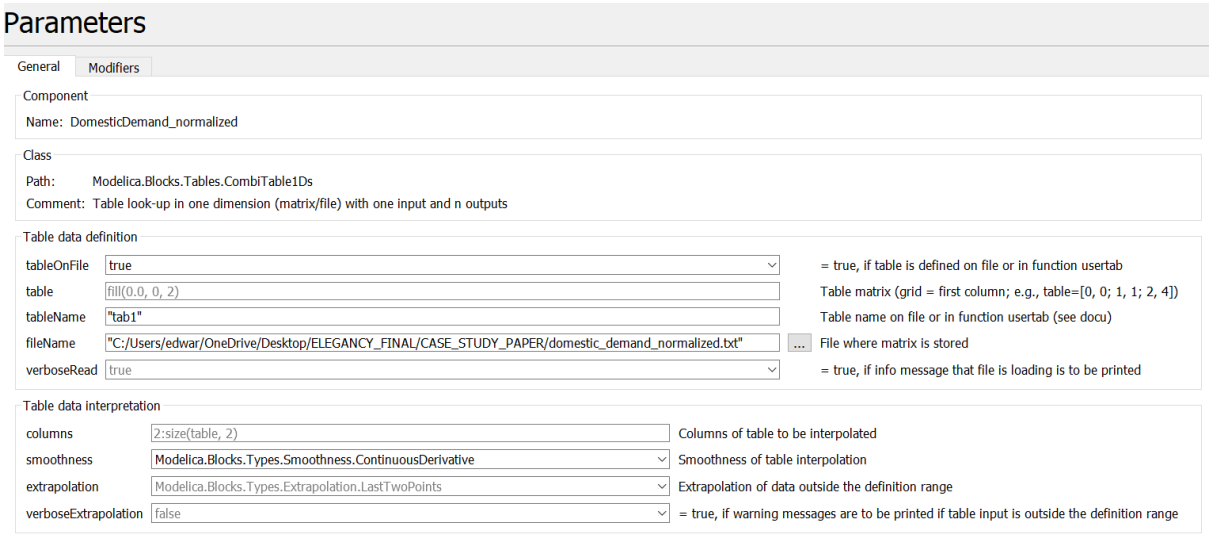
1. Drag the ‘Modelica.Blocks.Tables.CombiTable1Ds’ block into the interface

2. Drag the 'Modelica.Blocks.Sources.RealExpression' block into the interface – this is to input time as the independent variable to determine the corresponding H_2 demand at a given time.
3. Connect the time signal to the input of the table.

After renaming the component blocks (right click > attributes), the following should be displayed:



For continuous spline interpolation of the data and a signal with continuous derivatives, the user should specify the following parameters for the table:



This loads data from the .txt file titled "domestic_demand_normalized.txt" from the given path. This file should include a table named "tab1". The format of this file is as follows:

```
#1
double tab1(17520,2)   # comment line
0       0.06900712
1800    0.048730715
3600    0.050668324
5400    0.041691251
7200    0.04027165
9000    0.037854813
```

The first column is the time in seconds, the second column is the H_2 demand in kg/s. The first argument to tab1 is the number of rows of data, and the second is the number of columns. The file "domestic_demand_normalized" gives a domestic heating demand signal which is

normalized to have a maximum value of 1. Multiple demand sources with the same functional form may then be included in the distribution chain by scaling this signal. This avoids having to read from multiple data sets which could be memory intensive.

5.5.2 FROM AN ANALYTIC FUNCTION

The demand signal can be specified with analytical function by adding a 'Modelica.Blocks.Sources.RealExpression' block to the interface and connecting it to the Real input connector on the H_2 sink model. Note that this expression should be the negative of the actual demand since flow is defined to be positive for flow out of the H2_demand unit. In Figure 13 we show how the H_2 demand can be modelled with a sine function.

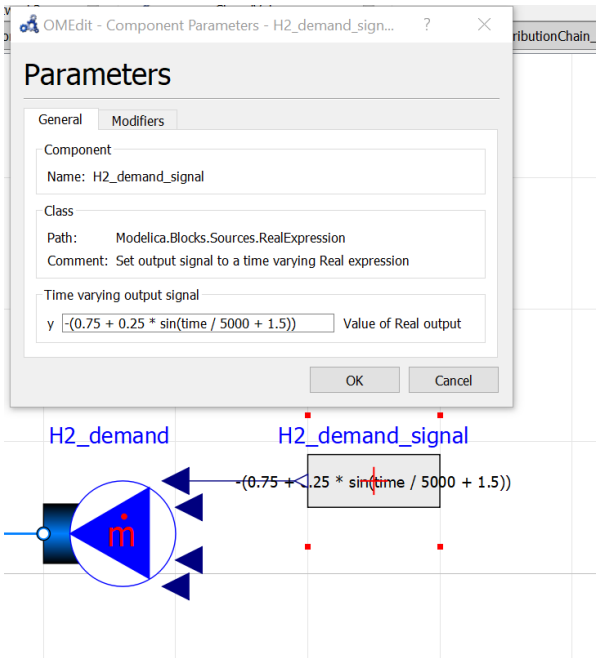


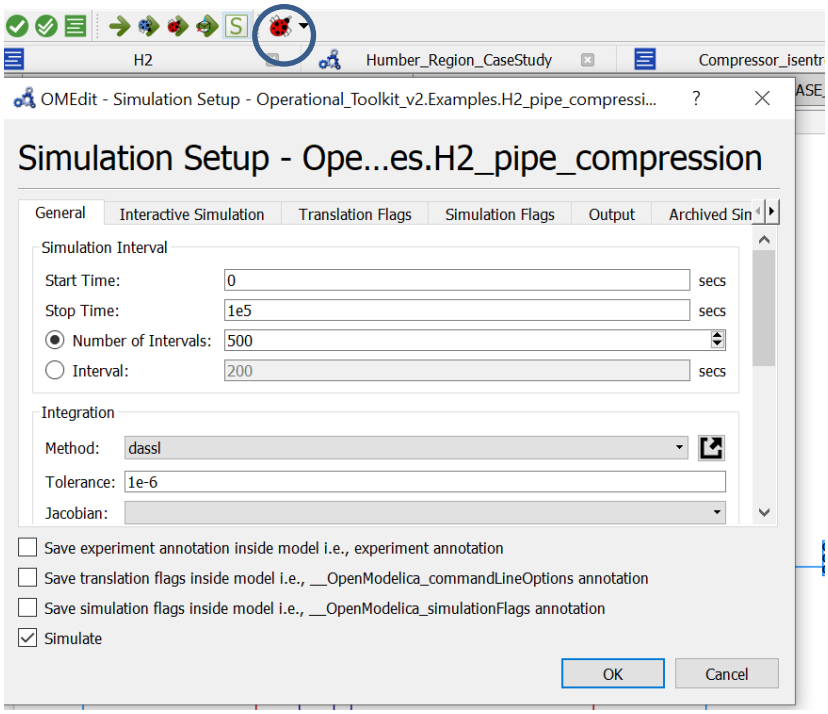
Figure 13 - Specifying an analytical demand signal for H_2

5.6 SPECIFYING A SUPPLY SIGNAL

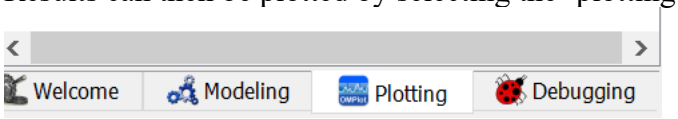
The specification of the flowrates from various production facilities is best shown in the examples section 7, where we show how one can control an SMR production facility with linear ramps.

6 SIMULATING AND PLOTTING RESULTS

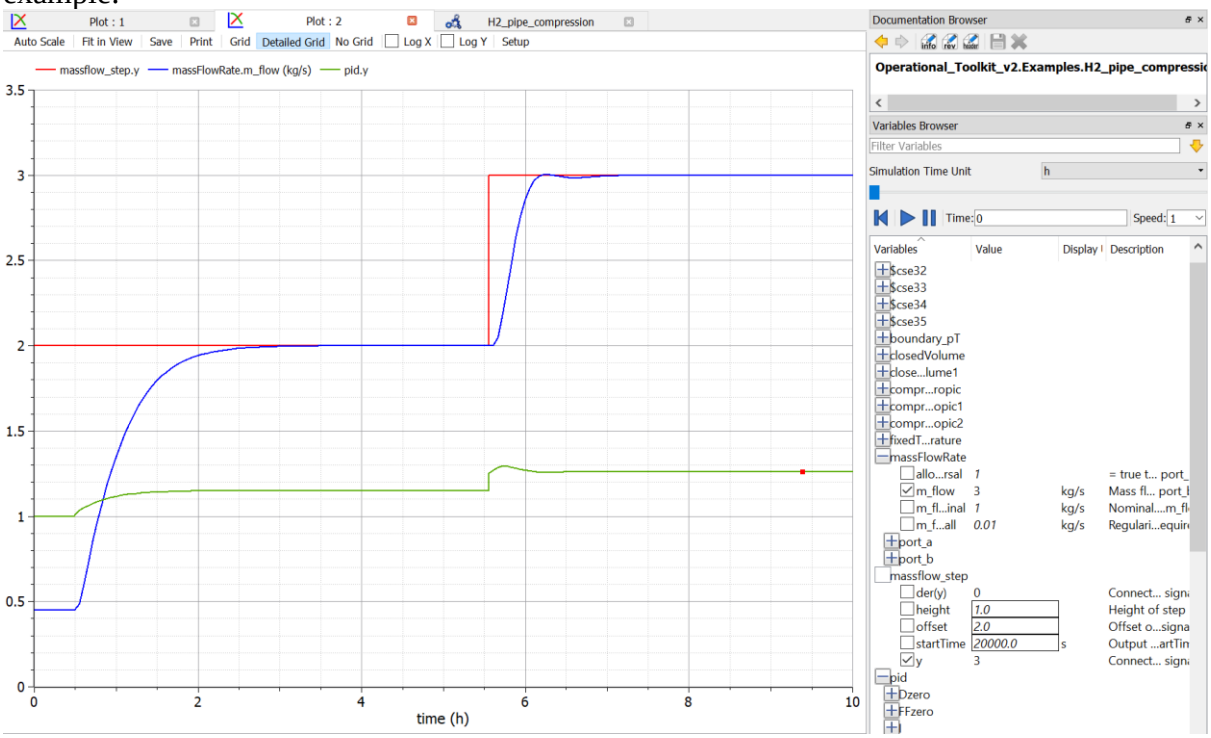
Clicking the symbol marked with an “S” loads the simulation set-up dialogue box. Here, the start time and stop time should be specified, and the number of reporting intervals. Note that the number of intervals does not affect the simulation result. In this box one can also specify the integration method and tolerance values. Clicking ‘OK’ will run the simulation. The simulation set-up options are saved until OMEdit is restarted. Subsequent simulations in the current session may simply be run by pressing the green arrow.



Results can then be plotted by selecting the ‘plotting’ tab:

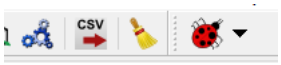


And selecting the desired variables to plot with a check mark in the variables browser, for example:



The same plots will be generated in subsequent simulations of the model.

The results may be exported to a .csv file for plotting or further analysis by clicking the csv icon in the toolbar at the top:

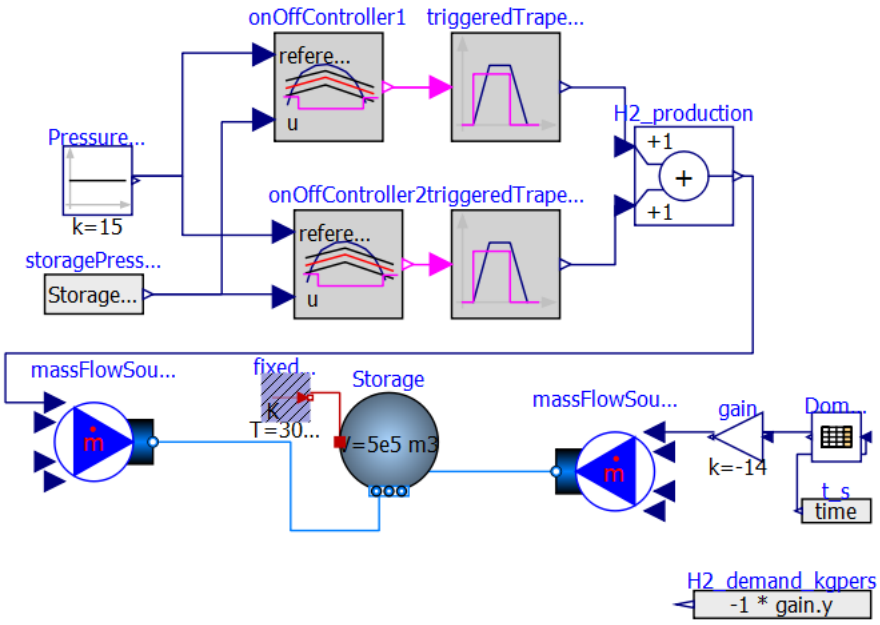


7 EXAMPLES

7.1 SATISFYING H2 DEMAND WITH AN SMR PRODUCTION FACILITY AND A STORAGE UNIT

In Examples.Production_SMR we consider a scenario where a storage unit is fed by an SMR production facility, and H_2 is removed from the storage unit at a rate needed to satisfy a typical domestic heating demand signal. Compression and piping is not considered in this example, it is a simplified scenario to show how the storage pressure may be controlled by ramping production, where the rate of change in production of H_2 is constricted to rates typical of an SMR production facility.

The OMEdit interface shows the following schematic for this process.

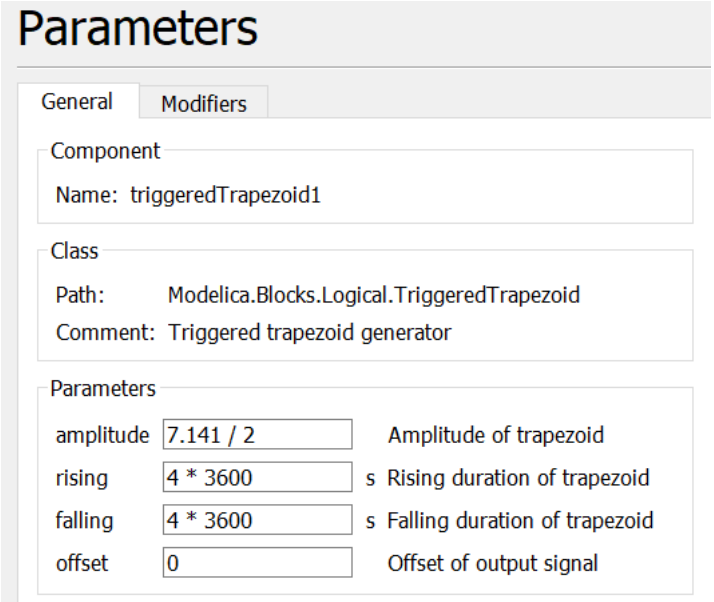


The model employs the following heuristics:

- The storage pressure is controlled around a set point of 15 MPa.
- Ideal heat transfer is assumed between the storage and the environment.
- The SMR facility takes 4 hours to ramp up and down between all sequential production points (0%, 50% and 100%) production, and linear ramping is assumed.

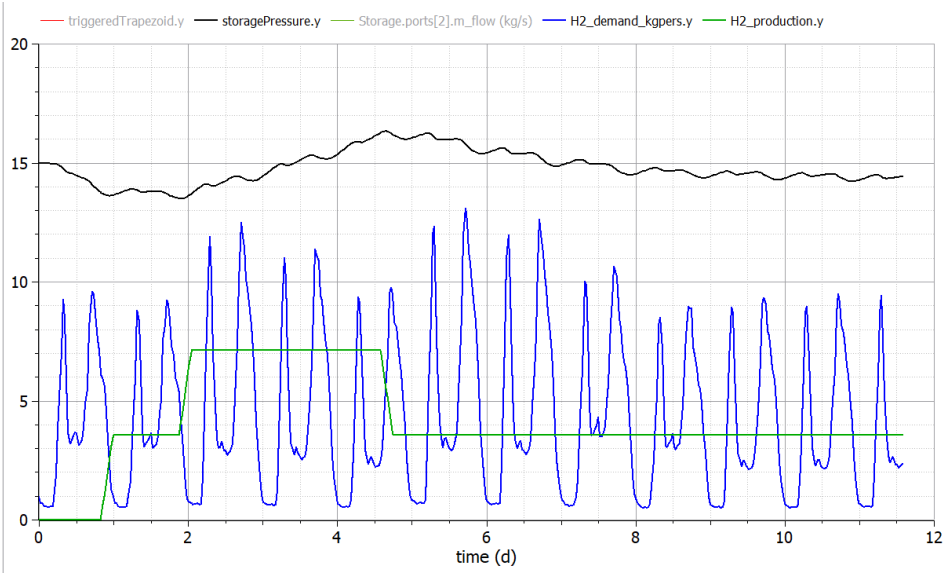
The change in SMR production is triggered by two on/off controllers which provide Boolean values depending on whether the storage pressure falls above and below a certain bandwidth around the set point. Different bandwidths are used for the on/off controllers. The trapezoids, representing the production rates, are triggered as follows: when the Boolean input changes from false to true, the production increases to the given amplitude. When it changes from true

to false, the production decreases to the lower bound. By combining two trapezoids as in this example, we can model a production facility with 3 different operating points. The SMR production unit can thus be modelled using the two trapezoids with the following parametrizations:



Where the maximum production rate is 7.141 kg/s. The more flexible operation of ATR/GHR CCS may be considered by reducing the rising and falling durations of the trapezoid.

Running the example produces the following result for the storage pressure, production rate and H_2 demand:

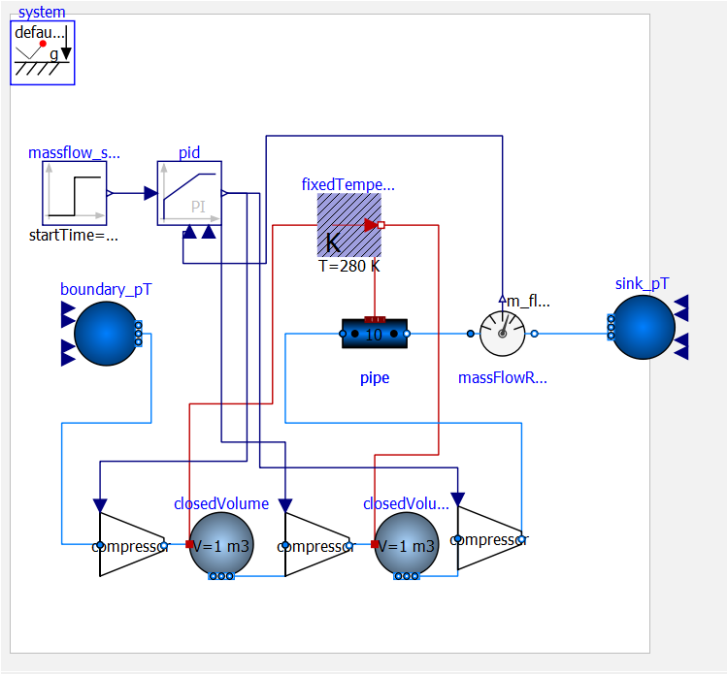


Note that the production facility operates at 3 different operating points over the 11 day period, and the storage pressure is controlled around 15 MPa.

7.2 CONTROL EXAMPLE WITH COMPRESSOR

The Modelica library contains a number of blocks that are useful for the control strategies. The toolkit contains an example named: "H2_pipe_compression", where a stream of hydrogen at a fixed temperature and pressure is delivered at a certain location by controlling a series of compressors.

The schematic for the process is the following:



Note that three idealised isentropic compression stages are connected in series, and a closedVolume is used in-between as a heat sink to the environment. This means that the temperature of hydrogen does not exceed the bounds specified in the CONSUMET tool, which could lead to numerical failures (e.g., a negative temperature solution may be found).

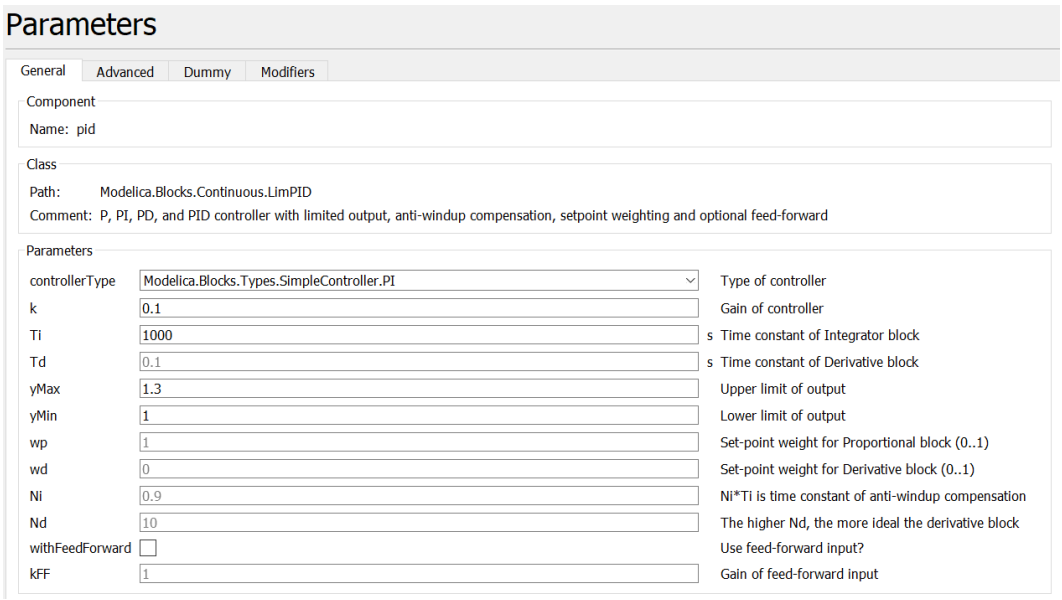
A proportional-integral controller is used to determine the compression ratio in each compressor to meet the set-point mass flow rate. A step change in the mass flowrate is given at a particular time. This model makes use of the massFlowRate sensor within the Modelica library.

The following steps may be used to tune the PID controller, as detailed in the "Modelica.Blocks.Continuous.LimPID" information:

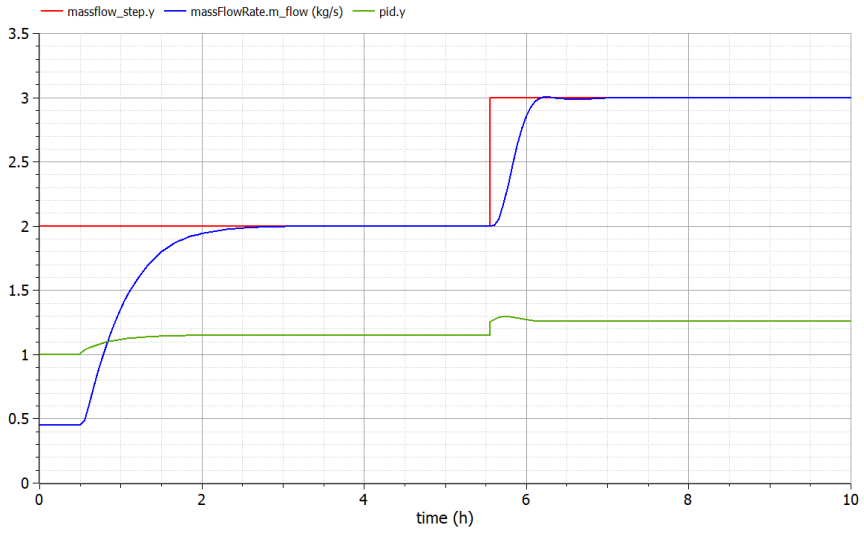
1. Set very large limits, e.g., yMax = Modelica.Constants.inf
2. Select a **P**-controller and manually enlarge parameter **k** (the total gain of the controller) until the closed-loop response cannot be improved any more.
3. Select a **PI**-controller and manually adjust parameters **k** and **Ti** (the time constant of the integrator). The first value of Ti can be selected, such that it is in the order of the time constant of the oscillations occurring with the P-controller. If, e.g., vibrations in the order of T=10 ms occur in the previous step, start with Ti=0.01 s.

4. If you want to make the reaction of the control loop faster (but probably less robust against disturbances and measurement noise) select a **PID**-Controller and manually adjust parameters **k**, **Ti**,**Td** (time constant of derivative block).
5. Set the limits yMax and yMin according to your specification.
6. Perform simulations such that the output of the PID controller goes in its limits.
   Tune **Ni** (Ni*Ti is the time constant of the anti-windup compensation) such that the input to the limiter block (= limiter.u) goes quickly enough back to its limits. If Ni is decreased, this happens faster. If Ni=infinity, the anti-windup compensation is switched off and the controller works bad.

This methodology leads to the following parameterization of the PI controller:



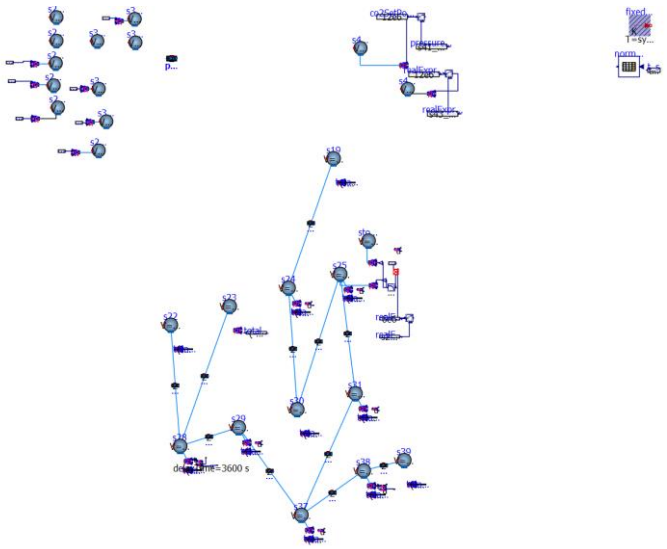
Running this simulation produces the following result:



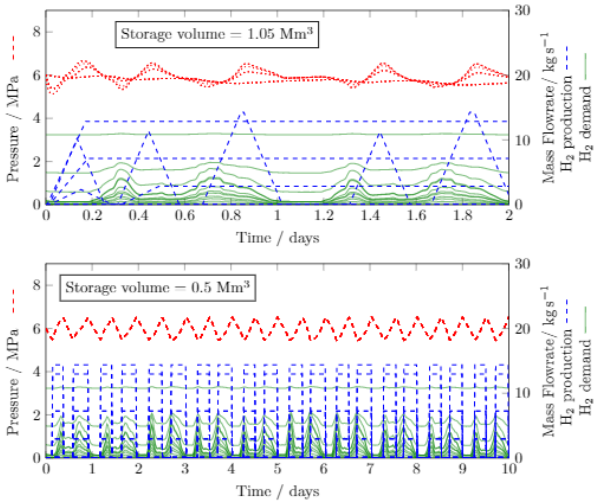
The step change in the mass flowrate set point between 2 and 3 kg/s at around 6h leads to an increase in compression ratio. For specific pipe considered here (200mm, 100km), the dynamic response time is approximately 0.5 h.

7.3 SIMULATING A LARGE-SCALE DISTRIBUTION SYSTEM WITH H_2 AND CO_2

In the final example entitled 'Humber_Region_CaseStudy' we consider a large-scale distribution network with various production facilities and demand signals in several locations.



Simulation of this model allows us to see how the given infrastructure may be operated to satisfy hydrogen demand in the Humber Region, with limitations on production ramping and Hydrogen/ CO_2 injection rates. The figure below shows an example analysis of the interplay between production, demand, and storage capacity.



8 CONCLUSIONS

In this document we have provided the information necessary for building and analysing various models applicable to H_2 + CCS distribution networks based on the operational toolkit developed as part of the ELEGANCY project. We have detailed the equations used to model

the various units and have shown how these units may be connected, parametrized, simulated and controlled within the OpenModelica interface.

REFERENCES

[1] https://doc.modelica.org/om/Modelica.Fluid.Types.ModelStructure.html