Summer Fellowship Report

On

Modeling of Unit Operations & Thermodynamic Functions in OpenModelica

Submitted by

Aditi Jain

Under the guidance of

Prof.Kannan M. Moudgalya
Chemical Engineering Department
IIT Bombay

July 12, 2019

Acknowledgment

First and foremost, I would like to thank Prof. Kannan Moudgalya for establishing this fellowship, which I believe was an excellent introduction to me on open-source software and technologies. His suggestions greatly improved the quality of the project.

I would also like to thank my mentors Mr. Priyam Nayak, Pravin Dalve and Rahul A S for providing valuable insight and expertise, as well as assisting me in overcoming the several difficulties I faced during the course of this project. Their advice on the modeling and simulation of unit operations significantly improved the accuracy in results. I would never be able to finish the project without their support.

I would also like to show my gratitude to my fellow interns and peers, who were constantly there to clarify my doubts and recommend improvements to the project. Their help and support was of immense value to me.

Contents

List of Tables

Chapter 1

Introduction

OpenModelica is a free/libre and open-source modeling and simulation environment. It is built on top of Modelica language. It employs equation-oriented approach in solving a given set of equations.

OpenModelica contains a very exhaustive library called Modelica Standard Library (MSL) which is a collection of different libraries from different domains such as electrical, mechanical, hydraulic, mathematics, etc. Since MSL doesn't have any chemical library, it is not of much use for chemical engineers if users intend to use OpenModelica for chemical process simulation.

FOSSEE has been developing the chemical library called "OMChemSim" in OpenModelica so that OpenModelica can also be used for chemical process simulation. The existing library already contains many thermodynamic functions & packages, component database and unit operations. The work which is going to be presented in this report will be to add a few more missing unit operations and thermodynamic function to OMChemSim.

The report is organised as following: Chapter 2 will be about the modeling of thermodynamic function viscosity. The results generated through the model will be compared with DWSIM. Chapter 3 & 4 will demonstrate modeling of unit operations orifice plate & valve respectively. The obtained results will be compared with that of DWSIM. Chapter 5 will demonstrate modeling of batch reactor. Since batch reactor doesn't exist in DWSIM, the results will be compared with some simple reaction examples solved through hand calculation. Chapter 6 is about conclusions. Chapter 7 contains the OpenModelica code of all the models developed for unit operations and thermodynamic function.

Chapter 2

Viscosity

2.1 Introduction

Viscosity can be defined as the ratio of local shear stress per unit area at any point to the velocity gradient. In other words, viscosity is also a measure of the internal fluid friction, which tends to oppose any dynamic change in the fluid motion.
If a shearing stress is applied to any portion of a confined fluid, the fluid will move with a velocity gradient with its maximum velocity at the point where the stress is applied. Viscosity is ordinarily referred to as a non-equilibrium property.

Viscosity is an important thermodynamic function used in the modeling and simulation of orifice plate.

2.2 Modeling

2.2.1 Liquid Viscosity

For a pure component, dependence of viscosity with temperature is given by equation

$$\eta = exp(A + \frac{B}{T} + ClogT + DT^E) \tag{2.1}$$

where A, B, C, D, E are the experimental coefficients available in ChemSep database, T is temperature in K and η is dynamic viscosity of the pure component in Pa.s

In calculating liquid viscosity, the effect of pressure is taken into the account using the method devised by Lucas .The routine computes the correction factor as given by equation

$$\frac{\eta_{corr}}{\eta} = \frac{1 + D(\frac{\delta P_r}{2.118})^A}{1 + C * w * \delta P_r} \tag{2.2}$$

where,
η_{corr} is the correction factor calculated from the Lucas routine
η is the liquid viscosity of pure component calculated using equation 2.1
δP_r is the change in reduced pressure which can be calculated using the equation

2.3
and A, C, D are empirical constants that can be calculated from equation 2.4, 2.5 and 2.6 respectively.

$$\delta P_r = \frac{P - P_{sat}}{P_c} \tag{2.3}$$

$$A = 0.9991 - \frac{0.0004674}{(1.0523T_r^{-0.03877}) - 1.0513} \tag{2.4}$$

$$C = -0.07921 + 2.1616T_r - 13.404T_r^2 + 44.1706T_r^3 - 84.8291T_r^4 + 96.1209T_r^5 - 59.8127T_r^6 + 15.6719T_r^7 \tag{2.5}$$

$$D = \frac{0.3257}{(1.0039 - T_r^{2.573})^{0.2906}} - 0.2086 \tag{2.6}$$

The corrected liquid viscosity is given by the equation

$$\eta_{corr} = \frac{\eta_L}{\eta} \tag{2.7}$$

where,
η_L is the corrected viscosity
η_{corr} is viscosity correction factor calculated using equation 2.2
η is liquid viscosity calculated using equation 2.1

The liquid phase viscosity can be calculated from

$$\eta_{mix}^L = exp(\sum_{i=1}^{n} x_i \ln \eta_i^L) \tag{2.8}$$

where,
η_{mix}^L is the liquid phase viscosity in Pa.s
x_i is the liquid phase mole fraction of i^{th} component
η_i^L is corrected liquid viscosity of i^{th} component in Pa.s

2.2.2 Vapor Viscosity

For a pure component, dependence of viscosity with temperature is given by equation

$$\eta = A + \frac{BT^C}{1 + \frac{D}{T} + \frac{E}{T^2}} \tag{2.9}$$

where A, B, C, D, E are the experimental coefficients available in ChemSep database, T is temperature in K and η is dynamic viscosity of the pure component in Pa.s.

When the experimental coefficients are not available, the viscosity can be calculated as a function of temperature given by Lucas

$$\eta\xi = 0.807T_r^{0.618} - 0.357exp^{-0.449T_r} + 0.34exp^{-4.058T_r} + 0.018 \tag{2.10}$$

where,
η is viscosity in μP
ξ is reduced inverse viscosity in μP^{-1}
T_r is reduced Temperature

The reduced inverse viscosity can be calculated from

$$\xi = 0.176(\frac{T_c}{MM^3P_c^4})^{\frac{1}{6}} \tag{2.11}$$

where,
T_c is critical temperature in K
P_c is critical pressure in bar
MM is molecular weight in g/mol

The experimental or calculated viscosity is corrected to take into account the effect of pressure, by the Jossi-Stiel-Thodos method,

$$[(\eta - \eta_0)(\frac{T_c}{MM^3P_c^4}) + 1]^{\frac{1}{4}} = 1.023 + 0.23364\rho_r + 0.58533\rho_r^2 - 0.40758\rho_r^3 + 0.093324\rho_r^4 \tag{2.12}$$

where,
reduced density, $\rho_r = \frac{\rho}{\rho_c} = \frac{V_c}{V}$
η_0 is Lucas viscosity in μP
η is corrected viscosity in μP

The vapor phase viscosity can be calculated from

$$\frac{1}{\eta_{mix}^V} = \sum_{i=1}^{n} \frac{x_i}{\eta_i^V} \tag{2.13}$$

where,
η_{mix}^V is the vapor phase viscosity in Pa.s
X_i is the vapor phase mole fraction of i^{th} component
η_i^V is the corrected viscosity of i^{th} component in Pa.s

2.3 Comparison of Results

2.3.1 Liquid Viscosity Comparison

Sl No	Component System	Temperature, (K)	Pressure, (Pa)	Flow rate, (mol/s)	Composition	Liquid Viscosity	
						DWSIM	OpenModelica
1	Methanol - Water	315	101456	60	0.1-0.9	0.000508016	0.0006035
2	Methanol - Water	310	101654	60	0.2-0.8	0.000641729	0.000641573
3	Acetic acid-ethanol- propanol	373	101325	60	0.6-0.2-0.2	0.00994628	0.0004323
4	Acetic acid-Ehanol-Propanol	350	101325	60	0.4-0.3-0.3	0.0005677	0.000571872
5	Acetylene - ethylene- ethylchloride- acetone	200	101325	60	0.25-0.25-0.25-0.25	0.0046039	0.000318279
6	Acetylene - ethylene- ethylchloride- acetone	185	101325	60	0.1-0.2-0.1-0.6	0.0050761	0.00084

Table 2.1: Comparison of Liquid Viscosity in DWSIM and OpenModelica

2.3.2 Vapor Viscosity Comparison

Sl No	Component System	Temperature, (K)	Pressure, (Pa)	Flow rate, (mol/s)	Composition	Vapor Viscosity	
						DWSIM	OpenModelica
1	Methanol - Water	380	108000	60	0.5-0.5	1.30E-05	1.28E-05
2	Methanol - Water	450	108900	60	0.4-0.6	1.57E-05	1.51E-05
3	Acetic acid-ethanol- propanol	415	101325	60	0.7-0.1-0.2	1.12E-05	1.08E-05
4	Acetic acid-Ehanol-Propanol	435	102235	60	0.1-0.2-0.7	1.20E-05	1.61E-05
5	Acetylene-Ethylene-Ethylchloride-Aceton	315	201325	60	0.6-0.1-0.1-0.2	1.05E-05	1.00E-05
6	Acetylene-Ethylene-Ethylchloride-Aceton	355	101325	60	0.10.3-0.5-0.1	1.18E-05	1.1372

Table 2.2: Comparison of Vapor Viscosity in DWSIM and OpenModelica

Chapter 3

Orifice Meter

3.1 Introduction

What is orifice Plate?
A orifice meter or orifice plate is a flat plate having a central hole that is placed across the flow of a liquid, usually between flanges in a pipeline. The orifice plates are used for measuring the flow rates in pipes. when the fluid is single phase, well mixed, the fluid also occupies the entire pipe and well developed under these circumstances when the orifice plate is constructed and installed according to appropriate standards, flow rate can be easily calculated by the formulae based on substantial research. Orifice plates are also used to reduce pressure or restrict flow, in which case they are called restriction plates.

Advantages of orifice meter
Orifices are small plates and easy to install or remove, simple in construction, able to measures a wide range of flowrates, easily maintained, offer very little pressure difference, inexpensive, price does not increase dramatically with size and most suitable for most gases and liquids also easily fitted between flanges.

Disadvantages of orifice meter
Orifice require homogeneous fluid, accuracy is affected by density, pressure and viscosity of fluid also requires straight pipe to ensure accuracy is maintained and in gas application in order to be accurate it requires constant pressure and temperature.

Pressure tapping
There are three standard positions for pressure tapping:

1. Corner taps placed immediately upstream and downstream of the plate.
2. D and D/2 taps or radius taps placed one pipe diameter upstream and half a pipe diameter downstream of the plate.
3. Flange taps placed 25.4 mm (1 inch) upstream and downstream of the plate.

3.2 Modeling

By assuming steady-state, incompressible, laminar flow in a horizontal pipe (no change in elevation) with negligible frictional losses, Bernoulli's equation reduces to an equation relating the conservation of energy between two points on the same streamline.

Change in pressure in orifice can be calculated using the equation

$$q_m = \frac{C_d}{\sqrt{1-b^4}} \xi \frac{\pi}{4} d_2^2 \sqrt{2\rho\Delta P} \tag{3.1}$$

where,
q_m is mass flow rate in kg/s
C_d is dimensionless constant called coefficient of discharge.
b is also a dimensionless constant which is the ratio of orifice diameter d_2 to pipe diameter d_1

$$b = \frac{d_2}{d_1} \tag{3.2}$$

ξ is expansibility factor which is 1 for incompressible gases and most liquid
ρ is fluid density in kg/m^3
ΔP is differential pressure measured across the orifice in Pa

Coefficient of discharge, C_d, can be calculated using the equation 3.3 and 3.4 depending on constants defined for the pressure tappings.
for $L_1 < 0.433$

$$C_d = 0.5959 + 0.312b^{2.1} - 0.184b^8 + 0.0029b^{2.5}(\frac{10^6}{ReD}) + 0.09L_1(\frac{b^4}{1-b^4}) - 0.037L_2b^3 \tag{3.3}$$

for $L_1 > 0.433$

$$C_d = 0.5959 + 0.312b^{2.1} - 0.184b^8 + 0.0029b^{2.5}(\frac{10^6}{ReD}) + 0.039L_1(\frac{b^4}{1-b^4}) - 0.037L_2b^3 \tag{3.4}$$

where,
ReD is Reynolds number which can be calculated using the equation

$$ReD = \frac{q_m * d_2}{1000 * A_1 * \eta_{mix}} \tag{3.5}$$

L_1 and L_2 are constants depending upon pressure tappings as shown in table 3.1.
A_1 is the internal area of pipe that can be calculated using the equation

$$A_1 = 3.146\frac{d_1^2}{1000 * 4} \tag{3.6}$$

η_{mix} is the viscosity of the mixture which can be calculated using the equation

$$\frac{1}{\eta_{mix}} = [\frac{\beta}{\eta_{vap}} + \frac{\alpha}{\eta_{liq}}] \tag{3.7}$$

where,
β is the vapor phase mole fraction
α is the liquid phase mole fraction
η_{liq} is dynamic viscosity of pure liquid in Pa.s
η_{vap} is dynamic viscosity of pure vapor in Pa.s

The equation for calculating change in differential pressure across the orifice given in equation 3.1 can be rearranged as

$$\Delta P = (\frac{q_m}{C_d A_2})^2 (\frac{1 - b^4}{2\rho}) + \rho g (S_2 - S_1) \tag{3.8}$$

where,
A_2 is orifice area in m^2 can be calculated using the equation

$$A_2 = 3.146 \frac{d_2^2}{1000 * 4} \tag{3.9}$$

$(S_2 - S_1)$ is a constant depending upon pressure tapping as shown in table 3.1. Other constants like L_1 and L_2 are also defined.

Constant	Corner	Flange	Radius
$(S_2 - S_1)$	0	0.0508	$1.5d_2$
L_1	0	0.0254/(1000*d_2)	1
L_2	0	0.0254/(1000*d_2)	0.47

Table 3.1: Different constants defined for various pressure tappings

Overall pressure drop, ΔP_{fluid}, can be calculated using the equation

$$\Delta P_{fluid} = \Delta P * \frac{\sqrt{1 - b^4(1 - C_d^2)} - C_d b^2}{\sqrt{1 - b^4(1 - C_d^2)} + C_d b^2} \tag{3.10}$$

OpenModelica code for the orifice plate can be found in section **??**

3.3 Results

The OpenModelica code for the orifice plate was tested for various component systems at varying temperature and pressure conditions and the results for the same is reported in table 3.2, 3.3 and 3.4.

Input			
Component System		Methanol - Water	
Mole Flow, (mol/s)		60	
Composition, (-)		0.6 - 0.4	
Temperature, (K)		350	
Pressure, (Pa)		101456	
Pressure Tapping		Radius	
Phase		Mixed	
Results			
Orifice Pressure Drop, (Pa)		Overall Pressure Drop, (Pa)	
DWSIM	OpenModelica	DWSIM	OpenModelica
15544.3	15553.4	11012.3	11019.7

Table 3.2: Pressure drop comparison for a two-component system at radius tapping of orifice plate

Input			
Component System		Acetic acid - Ethanol - Propanol	
Mole Flow, (mol/s)		60	
Composition, (-)		0.4 - 0.3 - 0.3	
Temperature, (K)		350	
Pressure, (Pa)		101325	
Pressure Tapping		Flange	
Phase		Liquid	
Results			
Orifice Pressure Drop, (Pa)		Overall Pressure Drop, (Pa)	
DWSIM	OpenModelica	DWSIM	OpenModelica
633.02	644.525	446.72	454.85

Table 3.3: Pressure drop comparison for a three-component system at flange tapping of orifice plate

Input			
Component System		Acetylene - Ethylene- Ethylchloride- Acetone	
Mole Flow, (mol/s)		60	
Composition, (-)		0.6 - 0.1 - 0.1 - 0.2	
Temperature, (K)		315	
Pressure, (Pa)		201325	
Pressure Tapping		Corner	
Phase		Vapor	
Results			
Orifice Pressure Drop, (Pa)		Overall Pressure Drop, (Pa)	
DWSIM	OpenModelica	DWSIM	OpenModelica
29009.7	29012.6	20569.3	20571.6

Table 3.4: Pressure drop comparison for a four-component system at corner tapping of orifice plate

Chapter 4

Valve

4.1 Introduction

A valve is a device that regulates, directs or controls the flow of a fluid by opening, closing, or partially obstructing various passageways.

Valves have many uses, including controlling water for irrigation, industrial uses for controlling processes. Valves are found in almost every industrial process, small valves fitted to washing machines and dishwashers including water and sewage processing, mining, power generation, processing of oil, gas and petroleum, food manufacturing, chemical and plastic manufacturing and many other fields.

4.2 Modeling

In valve, it is desired to calculate either the outlet pressure or pressure drop by mentioning different input parameters depending on the user. In the current model, three different modes of operation are available which are

1. Outlet pressure
2. Pressure drop
3. Kv (valve flow coefficient)

Depending on the mode selected, user can provide input and calculate the output as indicated in the table 4.1

Modes	**Outlet Pressure**	**Pressure Drop**	**Valve Flow Coefficient**
Input Parameters	Outlet Pressure	Pressure Drop	Kv_{max}, Opening Percentage
Outlet Parameters	Pressure Drop	Outlet Pressure	Pressure Drop, Outlet Pressure

Table 4.1: Inlet and Outlet parameters for different calculation mode of valve

If the calculation mode is pressure drop or outlet pressure, then the output parameter calculated using simple pressure balance equation

$$P_{in} - P_{out} = \Delta P \tag{4.1}$$

If the calculation mode is valve flow coefficient, the pressure drop can be calculated using the equation

$$K_v = \frac{3600 * w * 10}{\sqrt{\Delta P * \rho}} \tag{4.2}$$

where,
K_v is valve flow coefficient
ΔP is the pressure drop across the valve in Pa
ρ is the density of the fluid in kg/s

K_v can be calculated for two different cases. One, where valve opening percentage relationship is specified and the other, where it is not specified.
If any opening percentage relationship is not specified, K_v can be calculated using the equation

$$K_v = K_{vmax} \tag{4.3}$$

where K_{vmax} is the maximum value of valve flow coefficient to be specified by the user.
When any valve opening percentage relationship is specified, K_v can be calculated using the equation

$$\frac{K_v}{K_{vmax}} = f(OP) \tag{4.4}$$

where $f(OP)$ is the valve opening expression which can be specified in form of

$$f = a * OP + b * OP^2 + c * OP^3 \tag{4.5}$$

where OP is the valve opening to be specified as input.

OpenModelica code for the valve can be found in section **??**

4.3 Results

The OpenModelica code for the valve was tested for various component systems at varying temperature and pressure conditions and the results for the same is reported in table 4.2, 4.3 and 4.4.

Input			
Component System		Methanol-Water	
Mole Flow, (mol/s)	60	Composition	0.7-0.3
Temperature, (K)	345	Pressure, (Pa)	101325
Results			
Outlet Pressure, (Pa)		Pressure Drop, (Pa)	
DWSIM	OpenModelica	DWSIM	OpenModelica
31939.7	31915.4	69385	69409.6

Table 4.2: Comparison of outlet pressure and pressure drop across valve for methanol-water system.

Input			
Component System		Ethylene -Aceticacid	
Mole Flow, (mol/s)	60	Composition	0.8-0.2
Temperature, (K)	160	Pressure, (Pa)	161325
Results			
Outlet Pressure, (Pa)		Pressure Drop, (Pa)	
DWSIM	OpenModelica	DWSIM	OpenModelica
160591	160641	733.943	684.47

Table 4.3: Comparison of outlet pressure and pressure drop across valve for ethylene-acetic acid system.

Input			
Component System		Ethylene-Ethanol	
Mole Flow, (mol/s)	60	Composition	0.4-0.6
Temperature, (K)	160	Pressure, (Pa)	188745
Results			
Outlet Pressure, (Pa)		Pressure Drop, (Pa)	
DWSIM	OpenModelica	DWSIM	OpenModelica
188201	188241	544.436	503.554

Table 4.4: Comparison of outlet pressure and pressure drop across valve for ethylene-ethanol system.

Chapter 5

Ideal Batch Reactor

5.1 Introduction

5.1.1 Characteristics of a Batch Reactor

• Each batch is a closed system.
• The total mass of each batch is fixed.
• The reaction (residence) time t for all elements of fluid is the same.
• It is assumed that, at any time, the batch is uniform (e.g., in composition, temperature, etc.), because of perfect mixing.

5.1.2 Advantages:

• High conversions can be obtained
• Versatile, used to make many products
• Good for producing small amounts
• Easy to Clean

5.1.3 Dis-advantages:

• High cost of labor per unit of production
• Difficult to maintain large scale production
• Long Charging and Discharging times

5.2 Modeling

The Rate Equation
Suppose a single-phase reaction $aA + bB \longrightarrow rR + sS$. Then the rate of reaction for reactant A is then

$$-r_A = k * C_A^a * C_B^b \tag{5.1}$$

Where
$-r_A$ is rate of disappearance of A
dN_A/dt is change in number of moles of A with respect to time

k is the rate constant
C_A and C_B are the outlet concentration of components A and B.

The rate constant, k, can be calculated using a general expression as a function of temperature, as given in equation 5.2

$$k = k_0 + k_1 * T + k_2 * T^2 + k_3 * T^3 + \frac{k_4}{T} + k_5 * exp^{\frac{k_6}{T}} \tag{5.2}$$

Performance Equation of Batch Reactor

Overall mole balance equation is given by

$$Input = Output + Disappearance + Accumulation \tag{5.3}$$

In a batch reactor, there is no constant flow of input and output. Therefore,

$$Input = Output = 0 \tag{5.4}$$

Therefore equation 5.3 reduces to

$$(Disappearance) = -(Rate of accumulation) \tag{5.5}$$

Assuming A to be the base component, it's disappearance and accumulation can be given by

Disappearance of A in reaction (moles/time) = $(-r_A)V$, where V is the volume of the reactor

Accumulation of A (moles/time) = $\frac{dN_A}{dt} = \frac{d[N_{A0}(1-X_A)]}{dt} = -N_{A0}\frac{dX_A}{dt}$

Applying these over equation 5.5, we get

$$(-r_A)V = -N_{A0}\frac{dX_A}{dt} \tag{5.6}$$

dividing throughout by V, we get

$$-r_A = -C_{A0}\frac{dX_A}{dt} \tag{5.7}$$

Conversion of the base component, A, can be given by

$$X_A = \frac{(C_{A0} - C_A)}{C_{A0}} \tag{5.8}$$

OpenModelica code for the reaction manager can be found in section 7.6.

5.3 Results

Reaction : $N_2 + 3H_2 \longrightarrow 2NH_3$
Rate equation : $-r_{N_2} = kC_{N_2}$

Flow rate (moles/sec)		60
Pressure (Pa)		101325
Temperature (K)		410
Mole Fraction (-)		
Nitrogen	Hydrogen	Nitric acid
0.25	0.75	0
Conversion	OpenModelica	Hand calculation
N_2	0.393469	0.393469
H_2	0.393469	0.393469

Table 5.1: Comparison of conversion of reactant for formation of nitric acid at 410 k temperature in vapor phase

Reaction : $C_2H_5OH + CH_3COOH \longrightarrow CH_3COOC_2H_5 + H_2O$
Rate equation : $-r_{C_2H_5OH} = kC_{C_2H_5OH}C_{CH_3COOH}$

Flowrate (kg/sec)			60
Pressure (Pa)			101325
Temperature (K)			300
Mole fraction			
C_2H_5OH	CH_3COOH	$CH_3COOC_2H_5$	H_2O
0.5	0.5	0	0
Conversion		OpenModelica	Hand Calculation
C_2H_5OH		0.452329	0.452313
CH_3COOH		0.452329	0.452313

Table 5.2: Comparison of conversion of reactant for formation of Ethyl acetate at 300 k temperature in Liquid phase

Chapter 6

Conclusions

This work summarizes the effort undertaken to add a few missing unit operations and thermodynamic function to the existing chemical library called OMChemSim in OpenModelica.

Firstly, thermodynamic function called viscosity was modelled for calculation in vapor and liquid phase. Correction factor was also added to the model to take into account the effect of pressure. The generated results were compared with that of DWSIM. In some cases, deviation in the DWSIM and OpenModelica results of viscosity was observed where the pressure of any component was above the saturated vapor pressure. For such cases, the viscosity correction factor is supposed to be neglected and the calculated viscosity is to be taken for further calculation. Whereas in DWSIM, though the viscosity correction factor was neglected but the calculated viscosity was taken to be zero and therefore the viscosity for such cases didn't match with OpenModelica. This mistake in the calculation routine of DWSIM was reported to its developer, Daniel Wagner.

Next, unit operations like orifice plate and valve were modelled for calculation. The generated results were compared with that of DWSIM. In all cases for orifice plate, the results were nearly same for DWSIM and OpenModelica. The reported deviation was due to the error in viscosity function as reported earlier. For valve, the results were found to be nearly same for DWSIM and OpenModelica.

Finally, batch reactor was also modelled for calculation. Some simple reactions were taken and the results were found by hand caculation. Generated OpenModelica results were compared with hand calculation results and found to be same.

Chapter 7

OpenModelica Code

7.1 Lucas Viscosity Correction Factor

```
function Lucas_Viscosity_CorrectionFactor
input Integer NOC "Number of components";
input Real Pc[NOC];
input Real Tc[NOC];
input Real AF[NOC];
input Real P;
input Real T;
input Real Psat[NOC];
output Real Cf[NOC];
protected
Real A[NOC];
Real Tr[NOC];
Real Pr[NOC];
Real D[NOC];
Real dPr[NOC];
Real C[NOC];
Real s[NOC];
Real t[NOC];
algorithm
for i in 1:NOC loop
Tr[i]:= T/Tc[i];
Pr[i]:= P/Pc[i];
A[i]:=0.9991-(0.0004674/(1.0523*Tr[i]^(-0.03877)-1.0513));
D[i]:=(0.3257/(1.0039-Tr[i]^(2.573))^0.2906)-0.2086;
C[i]:=-0.07921+(2.1616*Tr[i])-(13.404*Tr[i]^2)+(44.1706*Tr[i]^3)-(84.8291*
    Tr[i]^4)+(96.1209*Tr[i]^5)-(59.8127*Tr[i]^6)+(15.6719*Tr[i]^7);
dPr[i]:=(P-Psat[i])/Pc[i];
s[i]:=dPr[i]/2.118;
if s[i]<0 then
t[i]:=(-abs(s[i])^A[i]);
else
t[i]:=(s[i]^A[i]);
end if;
Cf[i]:=(1+D[i]*t[i])/(1+C[i]*AF[i]*dPr[i]);
end for;
end Lucas_Viscosity_CorrectionFactor;
```

7.2 Vapor Viscosity

```
function vapor_viscosity
```

```
2  input Real MW"Molecular weight";
3  input Real Pc"Critical pressure";
4  input Real Tc"Critical temperature";
5  input Real T "Temperature";
6  input Real P"Pressure ";
7  input Real Vc"Critical volume";
8  input Real Rhor" Reduced density";
9  output Real Vo"Corrected viscosity";
10 protected
11 Real C"empirical constant";
12 Real e"empirical constant";
13 Real Tr"Reduced temperature";
14 Real ve"empirical constant";
15 Real v"empirical constant";
16 Real V"Viscosity";
17 algorithm
18 Tr:=T/Tc;
19 e := 0.176*((Tc/((MW^3)*((Pc/100000)^4)))^(1/6));
20 ve := ((0.807*Tr^0.618)-(0.357*exp(-0.449*Tr))+(0.34*exp(-4.058*Tr))+0.018)
       ;
21 v := ve/e;
22 C:=((1.023+(0.23364*(Rhor))+(0.58533*(Rhor^2))-(0.40758*(Rhor^3))
       +(0.093324*(Rhor^4))));
23 Vo:= ((((C^4)-1)*0.176)/e)+v;
24 end vapor_viscosity;
```

7.3 Orifice Meter

```
1  model Orifice_Meter
2
3   //
       ================================================================================

4    //Variables to link material stream properties with orifice
5    Real F_in(min = 0, start = 100, unit = "moles/s") "Inlet mixture molar
         flow rate";
6    Real F_out(min = 0, start = 100, unit = "moles/s") "Outlet mixture molar
         flow rate";
7    Real P(min = 0, start = 101325, unit = "Pa") "Inlet pressure";
8    Real outP(min = 0, start = 101325, unit = "Pa") "Outlet pressure";
9    Real T(min = 0, start = 273.15, unit = "K") "Inlet Temperature";
10   Real outT(min = 0, start = 273.15, unit = "K") "Outlet Temperature";
11   Real beta_in(min = 0, max = 1, start = 0.5) "Inlet vapor phase mole
         fraction";
12   Real beta_out(min = 0, max = 1, start = 0.5) "Outlet vapor phase mole
         fraction";
13   Real H_in"inlet enthalpy";
14   Real H_out"outlet enthalpy";
15   Real inCompMolFrac[3, NOC](each min = 0, each max = 1, each start = 1 / (
       NOC + 1)) "Inlet component mole fraction";
16   Real outCompMolFrac[1, NOC](each min = 0, each max = 1, each start = 1 /
         (NOC + 1)) "Outlet component mole fraction";
17   Real totMolFlo[3]"total molar flow rate ";
18   Real MW[3]"molecular weight ";
19   Real totMasFlo[3]"total mass flow rate";
20   Real inCompMasFrac[3,NOC]"inlet mass fraction";
21
22
23   //
       ================================================================================

24   //Parameters for Orifice Plate
```

```
25    parameter Real d1(unit = "m") "Internal Diameter of pipe";
26    parameter Real d2(unit = "m") "Diameter of orifice";
27    Real q(unit = "kg/s") "Mass flow rate";
28    parameter Real e "Expansibility factor";
29    Real Rho(unit = "kg/m3") " Density of mixture";
30     Real muv(unit = "Pa s") " Dynamic viscosity of vapour phase";
31     Real mul(unit = "Pa s") " Dynamic viscosity of liquid phase";
32
33
34    //
          ==========================================================================

35    //Variables for Orifice Plate
36    Real cd "Coefficient of discharge";
37    Real b "Ratio of orifice diameter to pipe diameter";
38    Real mum(unit = "Pa s") "viscosity of mixture";
39    Real a1(unit = "m2") "Area of pipe";
40    Real a2(unit = "m2") "Area of orifice plate";
41    Real s2_s1(unit = "m") "difference in height";
42    Real l1(unit = "m") "constant";
43    Real l2(unit = "m") "constant";
44    Real Red "Reynolds Number";
45    Real DP(unit = "Pa") "orifice pressure difference ";
46    Real fluidDP(unit = "Pa") "overall pressure difference";
47    /*Visc_Liq[NOC]"dynamic viscosity of liquid phase", Visc_Vap[NOC]"dynamic
          viscosity of vapour phase", muv(unit = "m2/s")" Kinematic viscosity
        of vapour phase", mul(unit = "m2/s") " Kinematic viscosity of liquid
        phase"*/
48
49    parameter Integer NOC "Number of components";
50    parameter Simulator.Files.Chemsep_Database.General_Properties comp[NOC] "
        Array of components";
51    import Modelica.Constants.*;
52    import Modelica.SIunits.*;
53    import Simulator.Files.*;
54    parameter String calcMode = "corner_taps" "corner_taps, flange_taps,
        radius_taps; choose the required operation";
55
56    Simulator.Files.Connection.matConn inlet(connNOC = NOC)annotation(
57      Placement(visible = true, transformation(origin = {-90, 0}, extent =
            {{-10, -10}, {10, 10}}, rotation = 0), iconTransformation(origin =
            {-120, 0}, extent = {{-20, -20}, {20, 20}}, rotation = 0)));
58    Simulator.Files.Connection.matConn outlet(connNOC = NOC) annotation(
59      Placement(visible = true, transformation(origin = {90, 0}, extent =
            {{-10, -10}, {10, 10}}, rotation = 0), iconTransformation(origin =
            {120, 0}, extent = {{-20, -20}, {20, 20}}, rotation = 0)));
60
61
62  //================================================
63        Real LiqDens[NOC];
64        Real Liquid_Phase_Density;
65        Real VapDensity[NOC];
66       // Real VapDensity[NOC](unit = "kg/m^3");
67        Real Vapour_Phase_Density;
68        Real Density_Mixture;
69        parameter Real Zv = 1;
70        Real Visc_Liq[NOC];
71        Real Visc_Vap[NOC];
72
73
74  //Connector equations
75  equation
76    inlet.P = P;
77    inlet.T = T;
```

```
78    inlet.mixMolFlo = F_in;
79    inlet.mixMolEnth = H_in;
80    inlet.mixMolFrac[1, :] = inCompMolFrac[1, :];
81    inlet.mixMolFrac[2, :] = inCompMolFrac[2, :];
82    inlet.mixMolFrac[3, :] = inCompMolFrac[3, :];
83    inlet.vapPhasMolFrac = beta_in;
84    outlet.P = outP;
85    outlet.T = outT;
86    outlet.mixMolFlo = F_out;
87    outlet.mixMolEnth = H_out;
88    outlet.mixMolFrac[1, :] = outCompMolFrac[1, :];
89    outlet.vapPhasMolFrac = beta_out;
90  //
      ================================================================================

91  //material balance
92    F_in = F_out;
93    P - fluidDP = outP;
94    H_in = H_out;
95    outCompMolFrac[1, :] = inCompMolFrac[1, :];
96    totMolFlo[1] = F_in;
97    totMolFlo[1] = totMolFlo[2] + totMolFlo[3];
98    totMolFlo[3] = totMolFlo[1]*beta_in;
99    q = totMasFlo[1];
100
101 //Average Molecular Weights of respective phases
102   if beta_in <= 0 then
103     MW[1] = sum(inCompMolFrac[1,:].*comp[:].MW);
104     MW[2] = sum(inCompMolFrac[2,:].*comp[:].MW);
105     MW[3] = 0;
106     totMasFlo[1] = totMolFlo[1]*MW[1]*1E-3;
107     totMasFlo[2] = totMolFlo[2]*MW[2]*1E-3;
108     totMasFlo[3] = 0;
109     inCompMasFrac[1,:] = inCompMolFrac[1,:].*comp[:].MW/MW[1];
110     inCompMasFrac[2,:] = inCompMolFrac[2,:].*comp[:].MW/MW[2];
111     for i in 1:NOC loop
112     inCompMasFrac[3,i] = 0;
113   end for;
114 //Liquid_Phase_Density
115     LiqDens = Thermodynamic_Functions.Density_Racket(NOC, T, P, comp[:].Pc,
            comp[:].Tc, comp[:].Racketparam, comp[:].AF, comp[:].MW, Psat[:]);
116     Liquid_Phase_Density = 1 / sum(inCompMasFrac[2, :] ./ LiqDens[:]) / MW
            [2];
117 //Vapour Phase Density
118     for i in 1:NOC loop
119       VapDensity[i] = 0;
120     end for;
121     Vapour_Phase_Density = 0;
122 //Density of Inlet-Mixture
123     Density_Mixture = 1 / ((1 - beta_in) / Liquid_Phase_Density) * sum(
          inCompMolFrac[1, :] .* comp[:].MW);
124 //
      ================================================================================

125   elseif beta_in == 1 then
126     MW[1] = sum(inCompMolFrac[1,:].*comp[:].MW);
127     MW[2] = 0;
128     MW[3] = sum(inCompMolFrac[3,:].*comp[:].MW);
129     totMasFlo[1] = totMolFlo[1]*MW[1]*1E-3;
130     totMasFlo[2] = 0;
131     totMasFlo[3] = totMolFlo[3]*MW[3]*1E-3;
132     inCompMasFrac[1,:] = inCompMolFrac[1,:].*comp[:].MW/MW[1];
133     for i in 1:NOC loop
134     inCompMasFrac[2,i] = 0;
```

```
135       end for;
136       inCompMasFrac[3,:] = inCompMolFrac[3,:].*comp[:].MW/MW[3];
137
138   //Calculation of Phase Densities
139   //Liquid Phase Density-Inlet Conditions
140       for i in 1:NOC loop
141         LiqDens[i] = 0;
142       end for;
143       Liquid_Phase_Density = 0;
144   //Vapour Phase Density
145       for i in 1:NOC loop
146         VapDensity[i] = P / (Zv * 8.314 * T) * comp[i].MW * 1E-3;
147       end for;
148       Vapour_Phase_Density = 1 / sum(inCompMasFrac[3, :] ./ VapDensity[:]) /
              MW[3];
149   //Density of Inlet-Mixture
150       Density_Mixture = 1 / (beta_in / Vapour_Phase_Density) * sum(
              inCompMolFrac[1, :] .* comp[:].MW);
151     else
152       MW[1] = sum(inCompMolFrac[1,:].*comp[:].MW);
153       MW[2] = sum(inCompMolFrac[2,:].*comp[:].MW);
154       MW[3] = sum(inCompMolFrac[3,:].*comp[:].MW);
155       totMasFlo[1] = totMolFlo[1]*MW[1]*1E-3;
156       totMasFlo[2] = totMolFlo[2]*MW[2]*1E-3;
157       totMasFlo[3] = totMolFlo[3]*MW[3]*1E-3;
158       inCompMasFrac[1,:] = inCompMolFrac[1,:].*comp[:].MW/MW[1];
159       inCompMasFrac[2,:] = inCompMolFrac[2,:].*comp[:].MW/MW[2];
160       inCompMasFrac[3,:] = inCompMolFrac[3,:].*comp[:].MW/MW[3];
161   //Calculation of Phase Densities
162   //Liquid Phase Density-Inlet Conditions
163       LiqDens = Thermodynamic_Functions.Density_Racket(NOC, T, P, comp[:].Pc,
              comp[:].Tc, comp[:].Racketparam, comp[:].AF, comp[:].MW, Psat[:]);
164       Liquid_Phase_Density = 1 / sum(inCompMasFrac[2, :] ./ LiqDens[:]) / MW
              [2];
165   //Vapour Phase Density
166      for i in 1:NOC loop
167        VapDensity[i] = P / (Zv * 8.314 * T) * comp[i].MW * 1E-3;
168      end for;
169      Vapour_Phase_Density = 1 / sum(inCompMasFrac[3, :] ./ VapDensity[:]) /
           MW[3];
170   //Density of Inlet-Mixture
171      Density_Mixture = 1 / (beta_in / Vapour_Phase_Density + (1 - beta_in) /
           Liquid_Phase_Density) * sum(inCompMolFrac[1, :] .* comp[:].MW);
172    end if;
173
174    Rho = Density_Mixture;
175      for i in 1:NOC loop
176        Visc_Liq[i] = Transport_Properties.LiqVis(comp[i].LiqVis, T);
177      end for;
178     //calculation for viscosity
179    if beta_in <= 0.0 then
180    for i in 1:NOC loop
181    Visc_Vap[i] = 0;
182    end for;
183    mul= exp(sum(inCompMolFrac[2, :] .*log(Visc_Liq[:])));
184    muv = 0;
185    mum = mul;
186    elseif beta_in == 1 then
187    for i in 1:NOC loop
188    Visc_Vap[i] = Transport_Properties.VapVisc(comp[i].VapVis, T);
189    end for;
190    mul= 0 ;
191    muv = 1 / sum(inCompMolFrac[3, :] ./ Visc_Vap[:]);
192    mum = muv;
```

```
193     else
194     for i in 1:NOC loop
195     Visc_Vap[i] = Transport_Properties.VapVisc(comp[i].VapVis, T);
196     end for;
197     mul= exp(sum(inCompMolFrac[2, :] .*log(Visc_Liq[:])));
198     muv = 1 / sum(inCompMolFrac[3, :] ./ Visc_Vap[:]);
199     mum = 1 / (beta_in / muv + (1-beta_in) / mul);
200     end if;
201
202
203   //calculation for beta
204     b = d2 / d1;
205     a1 = pi * d1 ^ 2 / 4;
206     a2 = pi * d2 ^ 2 / 4;
207
208     Red = q * d2 / (a1 * mum);
209   //depends upon value of integer
210     if calcMode == "corner_taps" then
211       s2_s1 = 0;
212       l1 = 0;
213       l2 = 0;
214     elseif calcMode == "flange_taps" then
215       s2_s1 = 0.0508;
216       l1 = 1 / (d2 / 0.0254);
217       l2 = 1 / (d2 / 0.0254);
218     elseif calcMode == "radius_taps" then
219       s2_s1 = 1.5 * d2;
220       l1 = 1;
221       l2 = 0.47;
222     end if;
223   //calculation for coefficient of discharge based upon pressure tappings"
224     if l1 < 0.433 then
225       cd = 0.5959 + 0.312 * b ^ 2.1 - 0.184 * b ^ 8 + 0.0029 * b ^ 2.5 * (10
              ^ 6 / Red) ^ 0.75 + 0.09 * l1 * (b ^ 4 / (1 - b ^ 4)) - 0.0337 * l2
               * b ^ 3;
226     else
227       cd = 0.5959 + 0.312 * b ^ 2.1 - 0.184 * b ^ 8 + 0.0029 * b ^ 2.5 * (10
              ^ 6 / Red) ^ 0.75 + 0.039 * l1 * (b ^ 4 / (1 - b ^ 4)) - 0.0337 *
             l2 * b ^ 3;
228     end if;
229   // calculation for orifice pressure drop
230     DP = (q / (cd * a2)) ^ 2 * ((1 - b ^ 4) / (2 * Rho)) + Rho * g_n * s2_s1;
231   // calculation for overall pressure drop across orifice meter
232     if b >= 1 then
233       fluidDP = DP;
234     else
235       fluidDP = DP * ((1 - b ^ 4 * (1 - cd ^ 2)) ^ 0.5 - cd * b ^ 2) / ((1 -
              b ^ 4 * (1 - cd ^ 2)) ^ 0.5 + cd * b ^ 2);
236     end if;
237     annotation(
238       Icon(coordinateSystem(initialScale = 0.1), graphics = {Ellipse(
              fillColor = {85, 170, 255}, fillPattern = FillPattern.Sphere,extent
               = {{-100, 100}, {100, -100}}, endAngle = 360), Ellipse(fillColor =
               {255, 255, 255}, fillPattern = FillPattern.Solid, extent = {{-56,
              56}, {56, -56}}, endAngle = 360), Rectangle(origin = {0, 120},
              fillColor = {255, 255, 255}, fillPattern = FillPattern.Solid,
              extent = {{-20, 40}, {20, -20}})}),
239       Diagram);
240
241   end Orifice_Meter;
```

7.4 Valve

```
model kvalve

//=============================================================================

  //Variables to link material stream properties with orifice
  Real F_in(min = 0, start = 100, unit = "moles/s") "Inlet mixture molar flow rate";
  Real F_out(min = 0, start = 100, unit = "moles/s") "Outlet mixture molar flow rate";
  Real P(min = 0, start = 101325, unit = "Pa") "Inlet pressure";
  Real outP(min = 0, start = 101325, unit = "Pa") " calculated Outlet pressure";
  Real T(min = 0, start = 273.15, unit = "K") "Inlet Temperature";
  Real outT(min = 0, start = 273.15, unit = "K") "Outlet Temperature";
  Real beta_in(min = 0, max = 1, start = 0.5) "Inlet vapor phase mole fraction";
  Real beta_out(min = 0, max = 1, start = 0.5) "Outlet vapor phase mole fraction";
  Real H_in "inlet Molar enthalpy";
  Real H_out"Outlet Molar enthalpy";
  Real inCompMolFrac[3, NOC](each min = 0, each max = 1, each start = 1 / (NOC + 1)) "Inlet component mole fraction";
  Real outCompMolFrac[1, NOC](each min = 0, each max = 1, each start = 1 / (NOC + 1)) "Outlet component mole fraction";
  Real totMolFlo[3]"Total molar flow";
  Real MW[3]"Molecular weight of component";
  Real totMasFlo[3]"Total mass flow";
  Real inCompMasFrac[3,NOC]"inlet mass fraction of each phase";


//Variable link with density function
//=============================================================================
    Real LiqDens[NOC];
    Real Liquid_Phase_Density;
    Real VapDensity[NOC];
    Real Vapour_Phase_Density;
    Real Density_Mixture;
    parameter Real Zv = 1;


//Variable link with valve
//=============================================================================

parameter Real OutPressure(min = 0, start = 101325,unit ="Pa")=101325 "Outlet pressure";
parameter Real Pressuredrop(min = 0,start = 101325,unit = "Pa")=101325" Pressure drop";
Real Rho(unit = "kg/m3")"density of a mixture";
parameter Real Kvmax = 100  "valve flow coefficient";
Real w(unit= "kg/s")"Mass flow rate";
parameter Real valve_opening_percent= 25;
parameter Real a =5" a is cofficent of expression ( f = (a*OP)+(b*(OP^2))+(c*OP^3))";
parameter Real b =0" b is cofficent of expression ( f = (a*OP)+(b*(OP^2))+(c*OP^3))";
parameter Real c =0" c is cofficent of expression ( f = (a*OP)+(b*(OP^2))+(c*OP^3))";
parameter Integer Mode =1"MODE = 1 (Kv=Kvmax), MODE = 2 (Kv/Kvmax(%)=f(OP(%))expression";
parameter String calcMode = "Liquid_Service_Kv" "Outlet_Pressure, Pressure_Drop, Liquid_Service_Kv, Gas_Service_Kv, Steam_Service_Kv; choose the required operation";
Real pressDrop(unit = "Pa")"Calculated pressure drop" ;
```

```
46  Real OP"valve_opening_percent/100";
47  Real f"f = (a*OP)+(b*(OP^2))+(c*OP^3)";
48  Real Kv;
49  //Real Pg[NOC];
50  //(unit="kg/m3")"Density of gases at 0  C and 1013 mbar of each component
      ";
51  //Real PG ;
52  //
      ==========================================================================

53  //Real Pg;
54  /*Real Z[2,2],CV[3];
55  Real R[2],A;
56  //==========*///
      ==========================================================================
57   Simulator.Files.Connection.matConn inlet(connNOC = NOC)  annotation(
58     Placement(visible = true, transformation(origin = {-100, 0}, extent =
         {{-10, -10}, {10, 10}}, rotation = 0), iconTransformation(origin =
         {-90, 0}, extent = {{-10, -10}, {10, 10}}, rotation = 0)));
59   Simulator.Files.Connection.matConn outlet(connNOC = NOC)  annotation(
60     Placement(visible = true, transformation(origin = {96, 0}, extent =
         {{-10, -10}, {10, 10}}, rotation = 0), iconTransformation(origin =
         {90, 0}, extent = {{-10, -10}, {10, 10}}, rotation = 0)));
61
62  //
      ==========================================================================

63
64   parameter Integer NOC "Number of components";
65   parameter Simulator.Files.Chemsep_Database.General_Properties comp[NOC] "
       Array of components";
66   import Modelica.Constants.*;
67   import Modelica.SIunits.*;
68   import Simulator.Files.*;
69  //
      ==========================================================================

70
71  equation
72
73  //connector equation
74    inlet.P = P;
75    inlet.T = T;
76    inlet.mixMolFlo = F_in;
77    inlet.mixMolEnth = H_in;
78    inlet.mixMolFrac[1, :] = inCompMolFrac[1, :];
79    inlet.mixMolFrac[2, :] = inCompMolFrac[2, :];
80    inlet.mixMolFrac[3, :] = inCompMolFrac[3, :];
81    inlet.vapPhasMolFrac = beta_in;
82    outlet.P = outP;
83    outlet.T = outT;
84    outlet.mixMolFlo = F_out;
85    outlet.mixMolEnth = H_out;
86    outlet.mixMolFrac[1, :] = outCompMolFrac[1, :];
87    outlet.vapPhasMolFrac = beta_out;
88
89  //
      //==========================================================================

90    F_in = F_out;
91    H_in = H_out;
92    outCompMolFrac[1, :] = inCompMolFrac[1, :];
93    totMolFlo[1] = F_in;
94    totMolFlo[1] = totMolFlo[2] + totMolFlo[3];
```

```
95    totMolFlo[3] = totMolFlo[1]*beta_in;
96    w = totMasFlo[1];
97    P - pressDrop = outP;
98
99
100
101 //  ================================================================================
102
103  //Average Molecular Weights of respective phases
104    if beta_in <= 0 then
105      MW[1] = sum(inCompMolFrac[1,:].*comp[:].MW);
106      MW[2] = sum(inCompMolFrac[2,:].*comp[:].MW);
107      MW[3] = 0;
108       totMasFlo[1] = totMolFlo[1]*MW[1]*1E-3;
109       totMasFlo[2] = totMolFlo[2]*MW[2]*1E-3;
110       totMasFlo[3] = 0;
111       inCompMasFrac[1,:] = inCompMolFrac[1,:].*comp[:].MW/MW[1];
112       inCompMasFrac[2,:] = inCompMolFrac[2,:].*comp[:].MW/MW[2];
113       for i in 1:NOC loop
114       inCompMasFrac[3,i] = 0;
115    end for;
116 //Liquid_Phase_Density
117       LiqDens = Thermodynamic_Functions.Density_Racket(NOC, T, P, comp[:].Pc,
              comp[:].Tc, comp[:].Racketparam, comp[:].AF, comp[:].MW, Psat[:]);
118       Liquid_Phase_Density = 1 / sum(inCompMasFrac[2, :] ./ LiqDens[:]) / MW
            [2];
119 //Vapour Phase Density
120       for i in 1:NOC loop
121         VapDensity[i] = 0;
122       end for;
123       Vapour_Phase_Density = 0;
124 //Density of Inlet-Mixture
125       Density_Mixture = 1 / ((1 - beta_in) / Liquid_Phase_Density) * sum(
            inCompMolFrac[1, :] .* comp[:].MW);
126 //  ================================================================================
127    elseif beta_in == 1 then
128      MW[1] = sum(inCompMolFrac[1,:].*comp[:].MW);
129      MW[2] = 0;
130      MW[3] = sum(inCompMolFrac[3,:].*comp[:].MW);
131       totMasFlo[1] = totMolFlo[1]*MW[1]*1E-3;
132       totMasFlo[2] = 0;
133       totMasFlo[3] = totMolFlo[3]*MW[3]*1E-3;
134       inCompMasFrac[1,:] = inCompMolFrac[1,:].*comp[:].MW/MW[1];
135       for i in 1:NOC loop
136       inCompMasFrac[2,i] = 0;
137       end for;
138       inCompMasFrac[3,:] = inCompMolFrac[3,:].*comp[:].MW/MW[3];
139
140 //Calculation of Phase Densities
141 //Liquid Phase Density-Inlet Conditions
142       for i in 1:NOC loop
143         LiqDens[i] = 0;
144       end for;
145       Liquid_Phase_Density = 0;
146 //Vapour Phase Density
147       for i in 1:NOC loop
148         VapDensity[i] = P / (Zv * 8.314 * T) * comp[i].MW * 1E-3;
149       end for;
150       Vapour_Phase_Density = 1 / sum(inCompMasFrac[3, :] ./ VapDensity[:]) /
            MW[3];
```

```
151 //Density of Inlet-Mixture
152     Density_Mixture = 1 / (beta_in / Vapour_Phase_Density) * sum(
            inCompMolFrac[1, :] .* comp[:].MW);
153   else
154     MW[1] = sum(inCompMolFrac[1,:].*comp[:].MW);
155     MW[2] = sum(inCompMolFrac[2,:].*comp[:].MW);
156     MW[3] = sum(inCompMolFrac[3,:].*comp[:].MW);
157     totMasFlo[1] = totMolFlo[1]*MW[1]*1E-3;
158     totMasFlo[2] = totMolFlo[2]*MW[2]*1E-3;
159     totMasFlo[3] = totMolFlo[3]*MW[3]*1E-3;
160     inCompMasFrac[1,:] = inCompMolFrac[1,:].*comp[:].MW/MW[1];
161     inCompMasFrac[2,:] = inCompMolFrac[2,:].*comp[:].MW/MW[2];
162     inCompMasFrac[3,:] = inCompMolFrac[3,:].*comp[:].MW/MW[3];
163 //Calculation of Phase Densities
164 //Liquid Phase Density-Inlet Conditions
165     LiqDens = Thermodynamic_Functions.Density_Racket(NOC, T, P, comp[:].Pc,
            comp[:].Tc, comp[:].Racketparam, comp[:].AF, comp[:].MW, Psat[:]);
166     Liquid_Phase_Density = 1 / sum(inCompMasFrac[2, :] ./ LiqDens[:]) / MW
            [2];
167 //Vapour Phase Density
168   for i in 1:NOC loop
169     VapDensity[i] = P / (Zv * 8.314 * T) * comp[i].MW * 1E-3;
170   end for;
171   Vapour_Phase_Density = 1 / sum(inCompMasFrac[3, :] ./ VapDensity[:]) /
        MW[3];
172 //Density of Inlet-Mixture
173   Density_Mixture = 1 / (beta_in / Vapour_Phase_Density + (1 - beta_in) /
        Liquid_Phase_Density) * sum(inCompMolFrac[1, :] .* comp[:].MW);
174 end if;
175
176 Rho = Density_Mixture;
177
178
179 //   Pg[:] = 101325 / (Zv * 8.314 * 273.15) * comp[:].MW * 1E-3;
180
181 //   PG = 1/ sum(inCompMasFrac[3, :] ./ Pg[:]);
182
183
184 //   ================================================================================
185
186 if calcMode == "Outlet_Pressure" then
187   outP = OutPressure;
188   OP=100;
189   f=100;
190   Kv= 100;
191 elseif calcMode == "Pressure_Drop" then
192   pressDrop = Pressuredrop;
193   OP=100;
194   f=100;
195   Kv=100;
196 elseif calcMode == "Liquid_Service_Kv" then
197   if Mode == 1 then
198   OP=0;
199   f=0;
200   Kvmax = Kv;
201     elseif Mode == 2 then
202     OP = valve_opening_percent/100;
203     f = (a*OP)+(b*(OP^2))+(c*OP^3);
204     f = Kv/Kvmax;
205     end if;
206     Kv= ((3600*w*10)/(( pressDrop*Rho)^0.5));
207 elseif calcMode =="Gas_Service_Kv" then
```

```
208       if Mode == 1 then
209       OP=0;
210       f=0;
211       Kvmax = Kv;
212      elseif Mode == 2 then
213       OP = valve_opening_percent/100;
214       f = (a*OP)+(b*(OP^2))+(c*OP^3);
215       f = Kv/Kvmax;
216      end if;
217
218   //  ==========================================================================

219     //outP=101325;
220
221       // Kv= (((w*3600*(10^5))/519)*((T/(PG*abs(pressDrop)*outP))^0.5));
222      // (outP/100000) =  (P/100000)-(outP*T/10000*PG)*((w*3600)/(Kv*519))^2;
223  //     A=((T/PG)*(((3600*w*100000)/(Kv*519))^2));
224  //     // A=3154;
225  //      //outP*P-abs(outP*outP)=A;
226  //          CV[1]= -1;
227  //          CV[2]= P;
228  //          CV[3]= A;
229  //       Z= Modelica.Math.Vectors.Utilities.roots(CV);
230  //       R = {Z[2, i] for i in 1:2};
231  //       outP= max({R});
232      end if;
233
234      annotation(
235      Icon(graphics = {Line(origin = {-92, 94}, points = {{0, 0}}), Line(
             origin = {-80, 34}, points = {{0, 0}}), Line(origin = {-76, 48},
             points = {{0, 0}}), Line(origin = {-80, 80}, points = {{6, -40}}),
              Polygon(points = {{-82, 80}, {-82, 80}, {-82, 80}}), Polygon(
             origin = {-40.15, 0}, fillColor = {0, 85, 127}, fillPattern =
             FillPattern.HorizontalCylinder, points = {{-39.8536, 80},
             {-39.8536, -80}, {40.1464, 0}, {-39.8536, 80}}), Polygon(origin =
             {40.15, 0}, fillColor = {0, 85, 127}, fillPattern =
             FillPattern.HorizontalCylinder, points = {{39.8536, 80}, {39.8536,
             -80}, {-40.1464, 0}, {39.8536, 80}, {39.8536, 80}})},
             coordinateSystem(initialScale = 0)),
236      Diagram(coordinateSystem(initialScale = 0)),
237      version = "",
238      uses,__OpenModelica_commandLineOptions = "");
239
240      end kvalve;
```

7.5 Batch Reactor

```
1  model Batch_Reactor
2
3    //Variables to link material stream properties with Batch Reactor
4    Real F_in(min = 0, start = 100, unit = "moles/s") "Inlet mixture molar
         flow rate";
5    Real F_out(min = 0, start = 100, unit = "moles/s") "Outlet mixture molar
         flow rate";
6    Real P(min = 0, start = 101325, unit = "Pa") "Inlet pressure";
7    Real outP(min = 0, start = 101325, unit = "Pa") "Outlet pressure";
8    Real T(min = 0, start = 273.15, unit = "K") "Inlet Temperature";
9    Real outT(min = 0, start = 273.15, unit = "K") "Outlet Temperature";
10   Real beta_in(min = 0, max = 1, start = 0.5) "Inlet vapor phase mole
         fraction";
11   Real beta_out(min = 0, max = 1, start = 0.5) "Outlet vapor phase mole
```

```
        fraction";
12      Real H_in;
13      Real H_out;
14      Real inCompMolFrac[3, NOC](each min = 0, each max = 1, each start = 1 / (
            NOC + 1)) "Inlet component  mole fraction";
15      Real outCompMolFrac[1, NOC](each min = 0, each max = 1, each start = 1 /
            (NOC + 1)) "Outlet component mole fraction";
16      Real totMolFlo[3];
17      Real MW[3];
18      Real totMasFlo[3];
19      Real inCompMasFrac[3, NOC];
20      Real totVolFlo[3](each start = 30);
21      parameter Integer Phase = 1;
22      Real inCompMolFlo[3, NOC];
23      Real outCompMolFlo[1, NOC];
24      Real Flow_in[NOC, Nr];
25      Real Flow_out[NOC, Nr];
26      //=================================================
27      parameter Integer NOC "Number of components";
28      parameter Simulator.Files.Chemsep_Database.General_Properties comp[NOC] "
            Array of components";
29      import Modelica.Constants.*;
30      import Modelica.SIunits.*;
31      import Simulator.Files.*;
32      import ReactionManager.*;
33      // parameter link with Batch reactor
34      Real Con_in[NOC] "initial concentration of reactants";
35      parameter Integer Nr "no of reaction including forward or backward
            reaction";
36      Real r_base[Nr](unit = "moles/sec") "rate of the reaction wrt base
            component";
37      Real k[2, Nr] "Calculated rate constant at temp T";
38      Real conversion[Nr] "conversion of the component";
39      Real Con_out[NOC] " calculated final concentraion at time t";
40      Real X[NOC];
41      Real r[2, Nr];
42      Real cin[NOC, Nr];
43      Real cout[NOC, Nr];
44      Real x[NOC, Nr];
45      parameter String Mode = "Isothermal" "Isothermal, Outlet_temperature;
            Choose the required operating mode";
46      parameter String Rate_Calculation_Mode = "General_Rate" "General_Rate,
            Arrhenius; Choose the required rate calculation mode";
47      parameter Real Tdef;
48      //=========================
49      Real LiqDens[NOC];
50      Real Liquid_Phase_Density;
51      Real VapDensity[NOC](unit = "kg/m^3");
52      Real Vapour_Phase_Density;
53      Real Density_Mixture;
54      parameter Real Zv = 1;
55
56      extends ReactionManager.Reaction_Manager(NOC = NOC, comp = comp, Nr = 2,
            Bc = {1},Comp = 3, Sc = {{-1}, {-1}, {1}}, DO = {{1}, {0}, {0}}, RO =
             {{0}, {0}, {0}}, K0 = {{1, 1.5}, {0, 0}}, K1 = {{0, 0}, {0, 0}}, K2
            = {{0, 0}, {0, 0}}, K3 = {{0, 0}, {0, 0}}, K4 = {{0, 0}, {0, 0}}, K5
            = {{1.5, 1.5}, {0, 0}}, K6 = {{0, 0}, {0, 0}}, A = {{0.005, 0}, {0,
            0}}, E = {{0, 0}, {0, 0}});
57
58      //
            ==========================================================================

59      Simulator.Files.Connection.matConn inlet(connNOC = NOC) annotation(
60        Placement(visible = true, transformation(origin = {-92, 0}, extent =
```

```
        {{-10, -10}, {10, 10}}, rotation = 0), iconTransformation(origin =
        {-110, 70}, extent = {{-10, -10}, {10, 10}}, rotation = 0)));
61    Simulator.Files.Connection.matConn outlet(connNOC = NOC) annotation(
62      Placement(visible = true, transformation(origin = {92, 0}, extent =
        {{-10, -10}, {10, 10}}, rotation = 0), iconTransformation(origin =
        {110, -70}, extent = {{-10, -10}, {10, 10}}, rotation = 0)));
63    //
        ==========================================================================


64
65  algorithm
66    for j in 1:Nr loop
67      r[1, j] := k[1, j];
68      r[2, j] := k[2, j];
69      for i in 1:NOC loop
70        r[1, j] := r[1, j] * cout[i, j] ^ DO[i, j];
71        r[2, j] := r[2, j] * cout[i, j] ^ RO[i, j];
72      end for;
73      r_base[j] := r[1, j] - r[2, j];
74    end for;
75    outCompMolFrac[1, :] := outCompMolFlo[1, :] / F_out;
76
77  //Connector equations
78  equation
79    inlet.P = P;
80    inlet.T = T;
81    inlet.mixMolFlo = F_in;
82    inlet.mixMolEnth = H_in;
83    inlet.mixMolFrac[1, :] = inCompMolFrac[1, :];
84    inlet.mixMolFrac[2, :] = inCompMolFrac[2, :];
85    inlet.mixMolFrac[3, :] = inCompMolFrac[3, :];
86    inlet.vapPhasMolFrac = beta_in;
87    outlet.P = outP;
88    outlet.T = outT;
89    outlet.mixMolFlo = F_out;
90    outlet.mixMolEnth = H_out;
91    outlet.mixMolFrac[1, :] = outCompMolFrac[1, :];
92    outlet.vapPhasMolFrac = beta_out;
93  //=============================================
94  equation
95    F_out = sum(outCompMolFlo[1, :]);
96    P = outP;
97    totMolFlo[1] = F_in;
98    totMolFlo[1] = totMolFlo[2] + totMolFlo[3];
99    totMolFlo[3] = totMolFlo[1] * beta_in;
100 //Average Molecular Weights of respective phases
101   if beta_in <= 0 then
102     MW[1] = sum(inCompMolFrac[1, :] .* comp[:].MW);
103     MW[2] = sum(inCompMolFrac[2, :] .* comp[:].MW);
104     MW[3] = 0;
105     totMasFlo[1] = totMolFlo[1] * MW[1] * 1E-3;
106     totMasFlo[2] = totMolFlo[2] * MW[2] * 1E-3;
107     totMasFlo[3] = 0;
108     inCompMasFrac[1, :] = inCompMolFrac[1, :] .* comp[:].MW / MW[1];
109     inCompMasFrac[2, :] = inCompMolFrac[2, :] .* comp[:].MW / MW[2];
110     for i in 1:NOC loop
111       inCompMasFrac[3, i] = 0;
112     end for;
113 //Liquid_Phase_Density
114     LiqDens = Thermodynamic_Functions.Density_Racket(NOC, T, P, comp[:].Pc,
          comp[:].Tc, comp[:].Racketparam, comp[:].AF, comp[:].MW, Psat[:]);
115     Liquid_Phase_Density = 1 / sum(inCompMasFrac[2, :] ./ LiqDens[:]) / MW
          [2];
116 //Vapour Phase Density
```

```
117       for i in 1:NOC loop
118         VapDensity[i] = 0;
119       end for;
120       Vapour_Phase_Density = 0;
121   //Density of Inlet-Mixture
122       Density_Mixture = 1 / ((1 - beta_in) / Liquid_Phase_Density) * sum(
              inCompMolFrac[1, :] .* comp[:].MW);
123   //
          ================================================================================

124     elseif beta_in == 1 then
125       MW[1] = sum(inCompMolFrac[1, :] .* comp[:].MW);
126       MW[2] = 0;
127       MW[3] = sum(inCompMolFrac[3, :] .* comp[:].MW);
128       totMasFlo[1] = totMolFlo[1] * MW[1] * 1E-3;
129       totMasFlo[2] = 0;
130       totMasFlo[3] = totMolFlo[3] * MW[3] * 1E-3;
131       inCompMasFrac[1, :] = inCompMolFrac[1, :] .* comp[:].MW / MW[1];
132       for i in 1:NOC loop
133         inCompMasFrac[2, i] = 0;
134       end for;
135       inCompMasFrac[3, :] = inCompMolFrac[3, :] .* comp[:].MW / MW[3];
136   //Calculation of Phase Densities
137   //Liquid Phase Density-Inlet Conditions
138       for i in 1:NOC loop
139         LiqDens[i] = 0;
140       end for;
141       Liquid_Phase_Density = 0;
142   //Vapour Phase Density
143       for i in 1:NOC loop
144         VapDensity[i] = P / (Zv * 8.314 * T) * comp[i].MW * 1E-3;
145       end for;
146       Vapour_Phase_Density = 1 / sum(inCompMasFrac[3, :] ./ VapDensity[:]) /
              MW[3];
147   //Density of Inlet-Mixture
148       Density_Mixture = 1 / (beta_in / Vapour_Phase_Density) * sum(
              inCompMolFrac[1, :] .* comp[:].MW);
149     else
150       MW[1] = sum(inCompMolFrac[1, :] .* comp[:].MW);
151       MW[2] = sum(inCompMolFrac[2, :] .* comp[:].MW);
152       MW[3] = sum(inCompMolFrac[3, :] .* comp[:].MW);
153       totMasFlo[1] = totMolFlo[1] * MW[1] * 1E-3;
154       totMasFlo[2] = totMolFlo[2] * MW[2] * 1E-3;
155       totMasFlo[3] = totMolFlo[3] * MW[3] * 1E-3;
156       inCompMasFrac[1, :] = inCompMolFrac[1, :] .* comp[:].MW / MW[1];
157       inCompMasFrac[2, :] = inCompMolFrac[2, :] .* comp[:].MW / MW[2];
158       inCompMasFrac[3, :] = inCompMolFrac[3, :] .* comp[:].MW / MW[3];
159   //Calculation of Phase Densities
160   //Liquid Phase Density-Inlet Conditions
161       LiqDens = Thermodynamic_Functions.Density_Racket(NOC, T, P, comp[:].Pc,
              comp[:].Tc, comp[:].Racketparam, comp[:].AF, comp[:].MW, Psat[:]);
162       Liquid_Phase_Density = 1 / sum(inCompMasFrac[2, :] ./ LiqDens[:]) / MW
              [2];
163   //Vapour Phase Density
164       for i in 1:NOC loop
165         VapDensity[i] = P / (Zv * 8.314 * T) * comp[i].MW * 1E-3;
166       end for;
167       Vapour_Phase_Density = 1 / sum(inCompMasFrac[3, :] ./ VapDensity[:]) /
              MW[3];
168   //Density of Inlet-Mixture
169       Density_Mixture = 1 / (beta_in / Vapour_Phase_Density + (1 - beta_in) /
              Liquid_Phase_Density) * sum(inCompMolFrac[1, :] .* comp[:].MW);
170     end if;
171   //Component Molar Flow Rates in Phases
```

```
172   inCompMolFlo[1, :] = totMolFlo[1] .* inCompMolFrac[1, :];
173   inCompMolFlo[2, :] = totMolFlo[2] .* inCompMolFrac[2, :];
174   inCompMolFlo[3, :] = totMolFlo[3] .* inCompMolFrac[3, :];
175 //=================================
176 //Phase Volumetric flow rates
177   if Phase == 1 then
178     totVolFlo[1] = totMasFlo[1] / Density_Mixture;
179     totVolFlo[2] = totMasFlo[2] / (Liquid_Phase_Density * MW[2]);
180     totVolFlo[3] = totMasFlo[3] / (Vapour_Phase_Density * MW[3]);
181     Con_in[:] = inCompMolFlo[1, :] / totVolFlo[1];
182   elseif Phase == 2 then
183     totVolFlo[1] = totMasFlo[1] / Density_Mixture;
184     totVolFlo[2] = totMasFlo[2] / (Liquid_Phase_Density * MW[2]);
185     totVolFlo[3] = 0;
186     Con_in[:] = inCompMolFlo[2, :] / totVolFlo[2];
187   else
188     totVolFlo[1] = totMasFlo[1] / Density_Mixture;
189     totVolFlo[2] = 0;
190     totVolFlo[3] = totMasFlo[3] / (Vapour_Phase_Density * MW[3]);
191     Con_in[:] = inCompMolFlo[3, :] / totVolFlo[3];
192   end if;
193 //=============================================
194 //Isothermal Mode
195   if Mode == "Isothermal" then
196     outT = T;
197 //Outlet temperature defined
198   elseif Mode == "Outlet_temperature" then
199     outT = Tdef;
200   end if;
201 //=============================================
202
203
204 if Rate_Calculation_Mode == "General_Rate" then
205 k[1,:] = ReactionManager.General_Rate(Nr, outT, K0[1,:], K1[1,:], K2[1,:],
      K3[1,:], K4[1,:], K5[1,:], K6[1,:]);
206 k[2,:] = ReactionManager.General_Rate(Nr, outT, K0[2,:], K1[2,:], K2[2,:],
      K3[2,:], K4[2,:], K5[2,:], K6[2,:]);
207 elseif Rate_Calculation_Mode == "Arrhenius" then
208 k[1,:] = ReactionManager.Arrhenius(Nr, A[1,:], E[1,:], outT);
209 k[2,:] = ReactionManager.Arrhenius(Nr, A[2,:], E[2,:], outT);
210 end if;
211
212
213
214   for j in 1:Nr loop
215     for i in 1:NOC loop
216       if j == 1 then
217         Con_in[i] = cin[i, j];
218       elseif j > 1 then
219         cout[i, j - 1] = cin[i, j];
220       end if;
221     end for;
222   end for;
223   for i in 1:NOC loop
224     cout[i, Nr] = Con_out[i];
225     if Sc[i, Nr] < 0 then
226       X[i] = (Con_in[i] - Con_out[i]) / Con_in[i];
227     else
228       X[i] = 0;
229     end if;
230   end for;
231   for i in 1:Nr loop
232     r_base[i] = cin[Bc[i], i] * der(x[Bc[i], i]);
233   end for;
```

```
234     for j in 1:Nr loop
235       for i in 1:NOC loop
236         cout[i, j] = cin[i, j] - Sc[i, j] ./ Sc[Bc[j], j] * (cin[Bc[j], j] *
              x[Bc[j], j]);
237       end for;
238       for i in 1:NOC loop
239         if Sc[i, j] < 0 then
240           if i == Bc[j] then
241             x[i, j] = conversion[j];
242           else
243             x[i, j] = Sc[i, j] ./ Sc[Bc[j], j] * (x[Bc[j], j] * cin[Bc[j], j]
                  / cin[i, j]);
244           end if;
245         else
246           x[i, j] = 0;
247         end if;
248       end for;
249     end for;
250
251   //
        ================================================================================

252     for j in 1:Nr loop
253       for i in 1:NOC loop
254         if j == 1 then
255           Flow_in[i, j] = inCompMolFlo[1, i];
256         elseif j > 1 then
257           Flow_out[i, j - 1] = Flow_in[i, j];
258         end if;
259       end for;
260     end for;
261     for i in 1:NOC loop
262       Flow_out[i, Nr] = outCompMolFlo[1, i];
263     end for;
264     for j in 1:Nr loop
265       for i in 1:NOC loop
266         Flow_out[i, j] = Flow_in[i, j] - Sc[i, j] ./ Sc[Bc[j], j] * (Flow_in[
              Bc[j], j] * x[Bc[j], j]);
267       end for;
268     end for;
269
270
271     end Batch_Reactor;
```

7.6 Reaction Manager

```
1   package ReactionManager
2
3   model Reaction_Manager
4     //
        ================================================================================

5     import Simulator.Files.*;
6     import data = Simulator.Files.Chemsep_Database;
7     parameter Chemsep_Database.General_Properties comp[NOC];
8     parameter Integer NOC;
9     parameter Integer Nr;
10    //Number of Reactions involved in the process
11    parameter Integer Bc[Nr] "Base component of reactions";
12    parameter Integer Comp;
13    //Number of components involved in the reaction
14    parameter Real Sc[NOC, Nr];
```

```
15    //Stochiometry of reactions
16    parameter Real DO[NOC, Nr];
17    //Direct order of reactions
18    parameter Real RO[NOC, Nr];
19    //Reverse order of reactions
20    Real Stoic_Check[Nr];
21    //Returns whether the specified stoichiometry is correct
22    Real HOF_comp[NOC];
23    Real HOR[Nr];
24    parameter Real K0[2,Nr];
25    parameter Real K1[2,Nr];
26    parameter Real K2[2,Nr];
27    parameter Real K3[2,Nr];
28    parameter Real K4[2,Nr];
29    parameter Real K5[2,Nr];
30    parameter Real K6[2,Nr];
31    parameter Real A[2,Nr] "Arrhenius constants of forward reaction";
32    parameter Real E[2,Nr] "Activation Energy of the forward reaction";
33
34  equation
35  //Check of stoichiometric balance
36    //Stoic_Check = Simulator.Files.Models.ReactionManager.Stoichiometrycheck
          (Nr, NOC, comp[:].MW, Sc);
37    Stoic_Check = ReactionManager.Stoichiometrycheck(Nr, NOC, comp[:].MW, Sc)
          ;
38  //Calculation of Heat of Reaction
39    HOF_comp[:] = comp[:].IGHF .* 1E-3;
40  //
        ================================================================================

41    for i in 1:Nr loop
42      HOR[i] = sum(HOF_comp[:] .* Sc[:, i]) / Bc[i];
43    end for;
44    end Reaction_Manager;
45
46    function Stoichiometrycheck
47    //This functions checks the stoichiometry of the reaction we have given
          and returns "1" as output if the stoichiometry is okay and returns 0
          otherwise.
48    input Integer Nr "No. of Reactions";
49    input Integer NOC "Nomber of components in the required reactions";
50    input Real MW[NOC] "Molecular weight";
51    input Real Sc[NOC, Nr] "Reaction coefficients";
52    output Integer Check[Nr];
53  protected
54    Real D[Nr] = fill(0, Nr);
55  algorithm
56    for i in 1:Nr loop
57      for j in 1:NOC loop
58        D[i] := D[i] + MW[j] * Sc[j, i];
59      end for;
60      if D[i] <= 0.1 and D[i] >= (-0.1) then
61        Check[i] := 1;
62      else
63        Check[i] := 0;
64      end if;
65    end for;
66  end Stoichiometrycheck;
67
68  function Arrhenius
69    // Reaction rate constant k = A*exp(-E/RT)
70    input Integer Nr;
71    input Real A "To calulate reaction rate for forward reaction (Arrhenius
          constants of forward reaction)";
```

```
72    input Real E "To calculate reaction rate for forward reaction";
73    input Real T;
74    output Real k "reaction rate constants for forward reaction";
75  algorithm
76    k := A .* exp(-E / (8.314 * T));
77  end Arrhenius;
78
79
80  function General_Rate
81    // Reaction rate constant k = A*exp(-E/RT)
82    input Integer Nr;
83    input Real outT;
84    input Real K0 "reaction constant of the reaction";
85    input Real K1 "reaction constant of the reaction";
86    input Real K2 "reaction constant of the reaction";
87    input Real K3 "reaction constant of the reaction";
88    input Real K4 "reaction constant of the reaction";
89    input Real K5 "reaction constant of the reaction";
90    input Real K6 "reaction constant of the reaction";
91    output Real k "reaction rate constants for forward reaction";
92  algorithm
93      k := K0 * (K1 + K2 * outT + K3 * outT ^ 2 + K4 * log(outT) + K5 * exp(
          K6 / outT));
94
95  end General_Rate;
96
97  end ReactionManager;
```