LUND UNIVERSITY

Structured Modelling of Chemical Processes

An Object-Oriented Approach

Nilsson, Bernt

1989

Document Version:
Publisher's PDF, also known as Version of record

Link to publication

Citation for published version (APA):
Nilsson, B. (1989). *Structured Modelling of Chemical Processes: An Object-Oriented Approach*. [Licentiate Thesis, Department of Automatic Control]. Department of Automatic Control, Lund Institute of Technology (LTH).

Total number of authors:
1

General rights
Unless other specific re-use rights are stated the following general rights apply:
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.
• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal

Read more about Creative commons licenses: https://creativecommons.org/licenses/

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

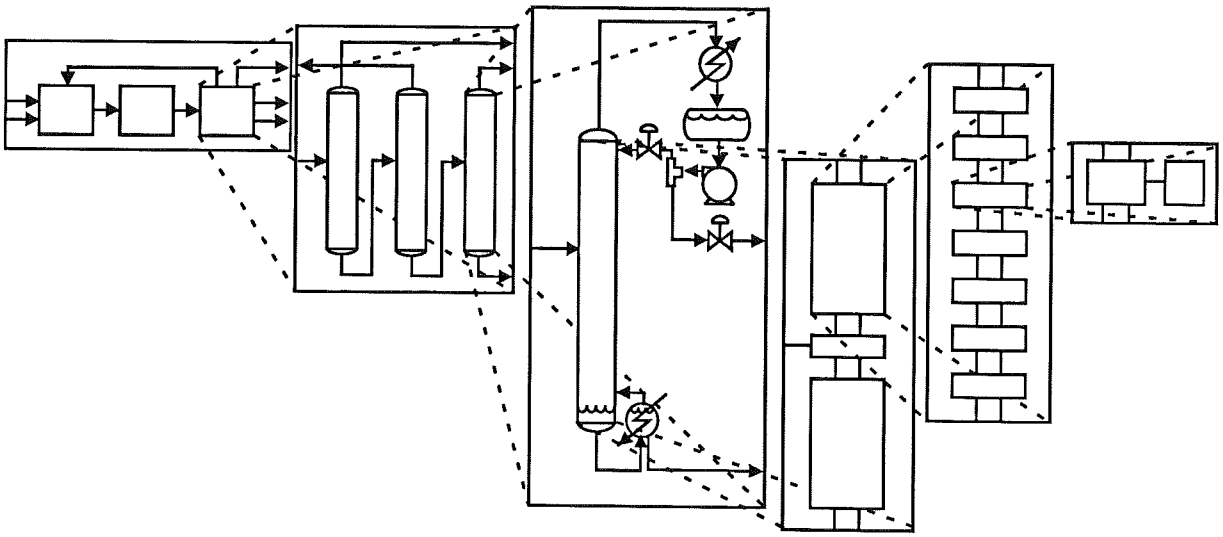
LUND UNIVERSITY

PO Box 117
221 00 Lund
+46 46-222 00 00

Structured Modelling of Chemical Processes

—

An Object-Oriented Approach



Bernt Nilsson

Department of Automatic Control
Lund Institute of Technology
September 1989

CODEN: LUTFD2/(TFRT-3203)/1–141/(1989)

Department of Automatic Control
Lund Institute of Technology
P.O. Box 118
S-221 00 Lund Sweden

Document name
Licentiate Thesis

Date of issue
September 1989

Document Number
CODEN: LUTFD2/(TFRT-3203)/1-141/(1989)

Author(s)
Bernt Nilsson

Supervisor
Sven Erik Mattsson and Karl Johan Åström

Sponsoring organisation
Swedish Board of Technical Development,
contract 87-02503

Title and subtitle
Structured Modelling of Chemical Processes – An Object-Oriented Approach

Abstract

Models and modelling are gaining more and more interest in process industry. Important tasks are model development, model reuse, model refinement and model maintenance. Systems for modelling and simulation of today do not support these tasks in a wider extent. A process modelling environment supporting these tasks will make it possible to distribute the modelling efforts to different model developers and end users. It will also increase the speed of model development. Modelling of a typical continuous chemical process based on an object-oriented approach is investigated. The study uses recently developed tools for model development and simulation. The chemical plant is used to illustrate ideas and benefits of the new object-oriented modelling methodology. In the case study the aim is to develop a process model of reusable and adaptable process objects. Modularization of large process models into encapsulated smaller submodels in a hierarchical submodel description is one of the major concepts that facilitates development and reuse. Inheritance is another concept, which also facilitates model development and reuse. Models and model components are represented as objects in single inheritance object class hierarchies. The power of inheritance in process modelling is shown and discussed. To further facilitate reuse and development of generic process submodels, a number of methods for process model decompositions and parameterizations are presented. Reusable and adaptable process models are stored in process model libraries, which can be used and modified by the user.

Key words
Modelling; process models; systems representations; hierarchical systems; computer simulation; simulation languages; computer-aided design; software tools; object-oriented modelling.

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title

ISBN

Language
English

Number of pages
141

Recipient's notes

Security classification

The report may be ordered from the Department of Automatic Control or borrowed through the University Library 2, Box 1010, S-221 03 Lund, Sweden, Telex: 33248 lubbis lund.

Structured Modelling of Chemical Processes – An Object-Oriented Approach

Bernt Nilsson

To Åsa

Department of Automatic Control
Lund Institute of Technology
P.O. Box 118
S–221 00 LUND
Sweden

© 1989 by Bernt Nilsson. All rights reserved

Published 1989
Printed in Sweden

Contents

Preface 3
Acknowledgements 3

1. Introduction 5

2. Structures in Process Models 7
2.1 A Typical Continuous Chemical Process 7
2.2 Model Decomposition 12
2.3 The Bubble Tank Reactor 15
2.4 Common Process Components 19
2.5 The Distillation Column 22
2.6 The Tubular Reactor 26
2.7 Summary 28

3. Object-Oriented Modelling 30
3.1 Object-Oriented Methodology 30
3.2 Introduction to Omola 32
3.3 Model Structures in Omola 38

4. Process Modelling in Omola 40
4.1 Top-Down Model Decomposition 40
4.2 Common Process Object Descriptions 43
4.3 The Tank Reactor 54
4.4 Distillation Columns 56
4.5 Modelling the Tubular Reactor 64
4.6 Summary 68

5. The Potential of Object-Oriented Modelling 70
5.1 A Comparison 70
5.2 Benefits of Object-Oriented Modelling 72
5.3 A Process Modelling Environment 75
5.4 Development of the Model Representation 78

6. Conclusions 84

7. References 86

A. Nomenclature 89

B. The Process Model 91
B1 The Process Model and its Subprocesses 92
B2 The Reaction Subprocess 95
B3 Separation Subprocess 104
B4 Common Process Submodel Library 118
B5 Process Terminal Library 130

Preface

This thesis presents an application project that uses new ideas and tools developed in the "Computer Aided Control Engineering" research program. Keywords in the project are model development and model reuse. The application is chemical process modelling and the work is therefore focused on how new tools support development of new models and how to support reuse of library process models. The result can be seen as a chemical process modelling methodology based on object-oriented approach and on model structuring concepts.

The reader is assumed to have basic knowledge in dynamic process modelling and in object-oriented methodology.

Acknowledgements

I would like to express my gratitude to Sven Erik Mattsson and Karl Johan Åström. Karl Johan Åström initialized the work and has been a consistent source of enthusiasm. Throughout this work Sven Erik Mattsson has supplied many useful ideas and taken time for our many and long discussions.

The project has been sponsored by the Swedish Board of Technical Development, contract 87-02503.

Furthermore, I am grateful to Mats Andersson, who is the inventor of the Omola language and a key person in the CACE project, for taking time for our discussions about modelling, Omola and everything.

Mats Andersson, Sven Erik Mattsson, Björn Wittenmark and Karl Erik Årzén have read several versions of the manuscript. Their suggestions and comments have greatly improved the quality of the thesis.

I also would like to thank Leif Andersson and Anders Blomdell for good TEX facilities and for convenient ways to include figures.

My colleagues at the Department of Automatic Control form a friendly and inspiring community and I am grateful for their encouragement and help.

I must not forget to thank my dog Julia, a true friend, for faithful companionship during long walks. Finally, I thank Åsa, my beloved wife, for her encouragement and for never losing her belief in me.

1. Introduction

An object-oriented approach to chemical process modelling is investigated in this thesis. It has been found to have a number of desirable properties. Model development, model reuse, model refinement and model maintenance are facilitated by an object-oriented approach to process modelling. Decomposition and parameterization are the basic methods to create these properties.

This thesis is based on an application study in the STU-supported research program "Computer Aided Control Engineering, CACE" . The study has been a part of the CACE-project "Tools for Model Development and Simulation" (Mattsson and Andersson, 1989b, or Nilsson et al, 1989). In the "Tool-project" a prototype is developed, which is called "System Engineering Environment, SEE". It is based on object-oriented model representation and includes a simulator for differential and algebraic equations, DAE. The aim of the "Tool-project" is to explore recent developments in computer science, like object-oriented representations, computer graphics, fast prototyping etc., in order to create a new environment for system engineering. This study has been focused on the application of these ideas in chemical process modelling.

Modelling and simulation of chemical processes have been done for many years. There are many well developed packages for steady-state simulations, for example ASPEN PLUS (ASPEN, 1985), and a few for dynamic simulations (Perkins, 1987). Almost every one is based on computer technology from the late sixties. None of the packages has an object-oriented approach, still the modelling of processes is based on objects. There are some new systems developed in the mid eighties, which are based on object-oriented methodology (Stephanopoulos, 1987, and G2, 1988).

The application considered in this study is a medium sized, typical chemical process. It is described in Chapter 2, where also model structuring concepts are discussed in order to identify general structures in process models. Appendix A contains a nomenclature list. In Chapter 3 the object-oriented model representation concepts, the SEE-prototype and the object-oriented modelling language Omola are presented. In Chapter 4 these tools are used successfully to model the plant described in

Chapter 2. The use of modularization, encapsulation, hierarchical decomposition, inheritance and various kinds of parameterization to facilitate development and reuse of process models are shown and discussed in detail. The Omola code of the continuous chemical process model is given in Appendix B.

Chapter 5 contains a more general discussion, including a comparison, a result discussion and a methodology description. Development, reuse, refinement and maintenance of process models are concluded to be facilitated through an object-oriented modelling methodology. A process modelling environment supporting this will make it possible to distribute the modelling efforts to different model developers and end users. Creation of model libraries with reusable models increases the speed in process model development. Support for model development and model reuse must be available together with tools for generating end user interface.

2. Structures in Process Models

Chemical processes are often complex plants that are composed of a large number of components. These process components are often standard process equipments that are used in different configurations in different processes. In this chapter a typical chemical process is described and modelled. The structure of the process model is of major importance. Based on the experiences of modelling the process, concepts for model structuring are postulated. The model structuring concepts are well known from Elmqvist (1978, 1986) and Mattsson (1988).

The organization of this chapter is as follows: First a typical continuous chemical process is described in Section 2.1. Then in Section 2.2 the decomposition of the process model is done. The modelling of the process components are then discussed. The bubble tank reactor is modelled in Section 2.3 and common process components, like pumps and valves, are described in Section 2.4. In the following section, Section 2.5, the distillation model is discussed and in Section 2.6 the tubular reactor is modelled. The purpose of the modelling of all these process components is to find common structures in chemical process models and the chapter concludes with a summary of the process model structuring concepts.

2.1 A Typical Continuous Chemical Process

A continuous chemical process is often described by a process flow sheet as in Figure 2.1. A process flow sheet is a two dimensional graphic description of the organization of a process. It shows what units a process is composed of and how they are connected to each other. Two chemical components, A and B, are processed in the process in order to produce the two major products, D and E. This is done in two chemical reaction steps. In the first step A and B are consumed to produce D and C.

$$A + B \rightarrow C + D$$

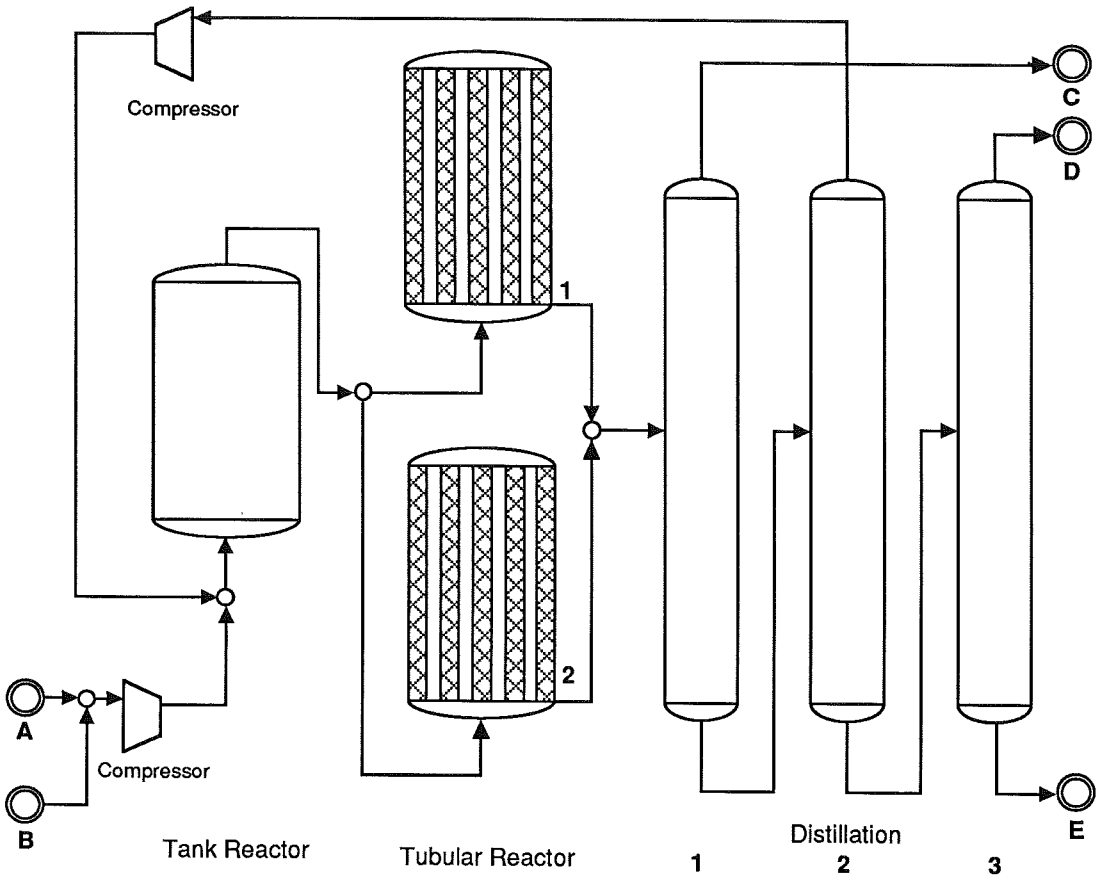


Figure 2.1 A continuous chemical process

The product C is an undesired by-product. Both the reactants and reaction products are in gas phase. The reaction is catalysted in liquid phase in a bubble tank reactor. In the second chemical reaction A is reacted with D and the reaction produce C and E. This reaction occur in gas phase and is catalysted by a solid material. The catalyst particles are packed in tubes in a so called fixed bed tubular reactor.

$$A + D \rightarrow C + E$$

The process must be flexible and allow production at different rates and qualities of the two products major products, D and E.

The chemical process can represent a quite common configuration. A minor modification of the process and it matches the process for production of ethyldiamine.

$$NH_3 + \text{ethylenoxide} \rightarrow \text{monoethanolamine}$$
$$NH_3 + \text{monoethanolamine} \rightarrow H_2O + \text{ethyldiamine}$$

Berol Kemi AB at Stenungsund in Sweden has a process that produces ethyldiamine in this way (see Törnquist, 1983).

The process can be divided into three subprocesses, namely preparation, reaction and separation. In the preparation subprocess the feed is prepared for the reaction subprocess. The chemical reactions occurs in different reactors in the reaction subprocess and in the separation subprocess are the products separated from each other through distillation.

Preparation Subprocess

In the preparation part of the process the raw materials, A and B, are mixed together with unreacted and recycled A and B. The following reaction subprocess operates at high pressure and high temperature. The reactants, that are in gas phase, are compressed in a compressor in order to create high pressure in the feed stream. The recycled stream is also compressed to the same pressure. The two streams are then mixed and heated before entering the reaction part.

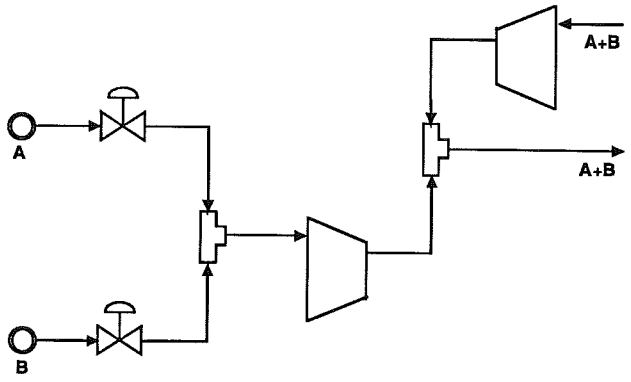


Figure 2.2 The preparation part with valves, mixers and compressors.

Reaction Subprocess

The feed stream, at high pressure, enters first a gas-liquid bubble tank reactor. During the rise of a gas bubble up through the reactor, the reactants diffuse into the liquid. The liquid contains catalyst material which makes the first reaction possible. The products diffuse back to the gas phase and leave the reactor in the gas outlet at the top of the tank.

The liquid is pumped through and heated in an external heat exchanger. See Figure 2.3. The reaction conditions are high pressure and high temperature. The reaction is weakly endothermic, which means that the reactor must be heated in order to make the reaction possible.

Liquid based catalyst makes it easy to regenerate the catalyst phase. It is possible to take out and regenerate a part of the liquid during operation. This is done in the external liquid loop.

The second reaction step occurs as a gas-solid reaction in a fixed bed tube reactor. The gas flows through process tubes filled with catalyst particles. The gas-solid reaction is exothermic. Heat released at the reaction is removed by the feed stream that is heated in the opposite direction to the reaction tube flow. There is also a large temperature change in the tube reactor. The warmest part of the reactor is called the *hot spot.*

A problem with fixed bed reactors is that they can not be regenerated during operation like the bubble reactor. It is therefore common and convenient to have two fixed bed reactors in parallel. It is then possible to change to half production during catalyst regeneration of one reactor. This reactor configuration is shown in Figure 2.3.

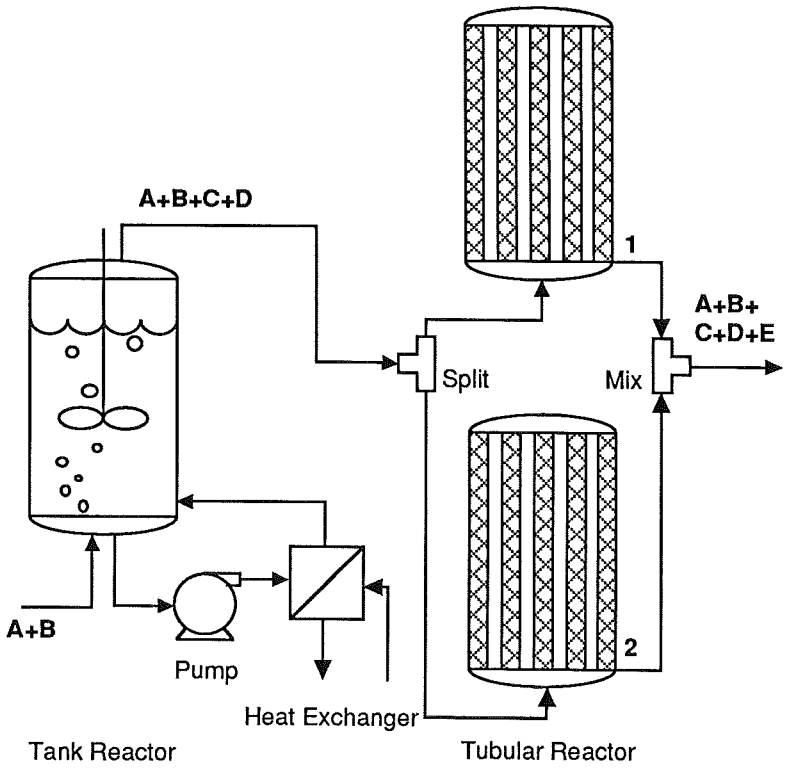


Figure 2.3 The reaction subprocess with a gas-liquid bubble tank reactor and two gas-solid catalyst fixed bed tubular reactors in parallel.

Separation Subprocess

In the next part of the process the products and unreacted reactants are separated from each other. The separation is done in three distillation columns in series, which is seen in Figure 2.4. Distillation is one of the most common unit operations for separation purposes. A unit operation is a part of a chemical process that have a special purpose and it is often composed of a number of physical components. The function of a distillation column is to split a flow into two new flow streams with different compositions by using energy.

In a distillation column heat is applied at the bottom of the column, which makes the liquid to boil. Vapour rises up through the column and is forced to interact with the liquid that is flowing down through the column. When vapour and liquid are in contact they tend to be at phase equilibrium. The phase equilibrium forces components that condense easily to be in the liquid phase and components that evaporate easily to be in the vapour phase. When the vapour comes to the top of the column it is condensed to liquid. Some part of this liquid is recycled to the column in order to create the important vapour-liquid interaction. The other part is the top product stream. The bottom product stream is a part of the liquid that are supposed to be reboiled.

In the first distillation column in the separation subprocess the lightest component C is separated from the main process stream. The product C is the only component in gas phase at the working conditions in this column. This means that the separation is done by evaporating dissolved C in the liquid. In Figure 2.4 this column is the one on the left. In the following separation unit unreacted A and B are separated and recycled to the preparation process part. This distillation column has a partial condenser, which allows recycling of the top product of A and B in gas phase. The bottom stream contains the major products, D and E. In the last distillation column, on the right in Figure 2.4, the product D is refined and separated from the product E. This column has a total condenser configuration. The condensed liquid is stored in a reflux drum, from which the reflux liquid and top product stream are removed. The other two columns have partial condensers, which means that the vapour from the column is only partially condensed to reflux liquid. In this case the top product stream is in vapour phase.

The process has two major products. The basic idea is to have a flexible production of the products, D and E, in order to cover the changing

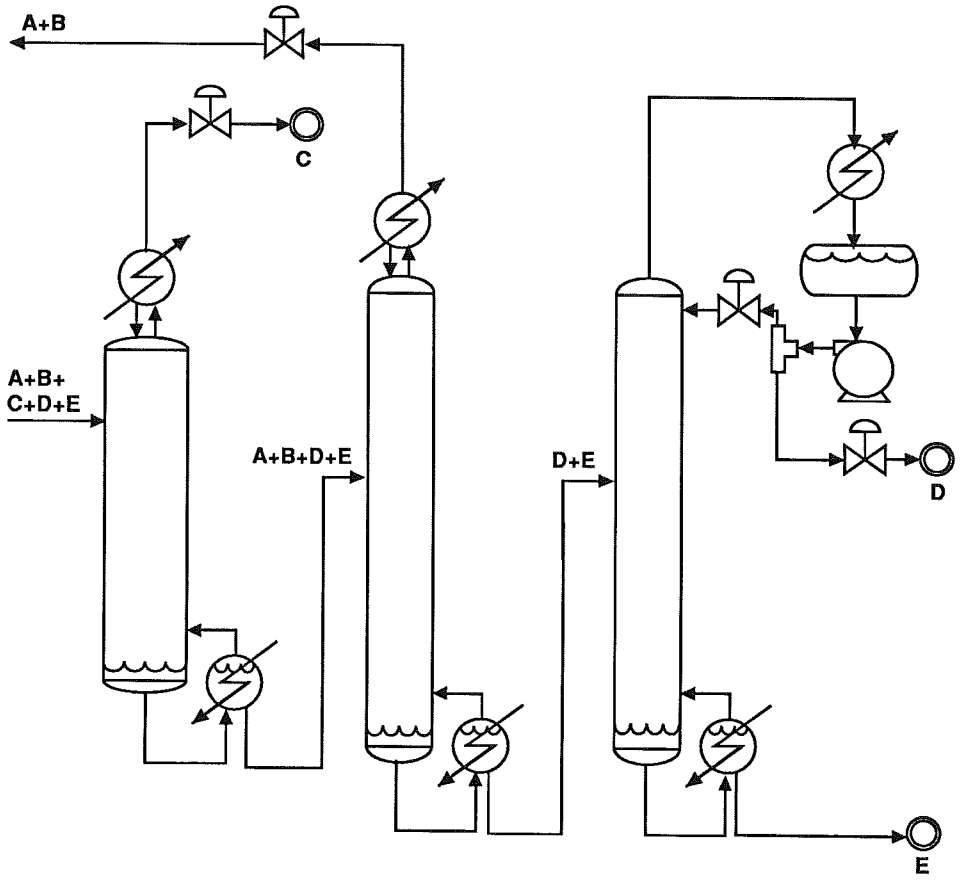


Figure 2.4 The separation part with three distillation columns in series.

demands on the markets for the products. The design of the process and control system must therefore be able to handle a wide range of production conditions.

2.2 Model Decomposition

A common way of describing chemical processes is by a flow sheet. A typical flow sheet is seen in Figure 2.1. The symbols on the map show what kind of equipment that are used in the process and how they are connected to each other. A unit or equipment symbol is often a representation of a physical object or a set of physical objects with a specific function. Connections in a flow sheet are representations for flow pipes but also other kinds of physical connections such as electric wires.

The flow sheet is a simplified map of the process. It is not a map of the process geometry or a description of the real physical plant. A great deal of knowledge is required to understand the behaviour of each individual unit based on the simplified map. A flow sheet often has hidden

or incomplete information. Often flow sheets do not contain information about control systems, safety equipment, start up and shut down equipment and energy supply systems. The reason for hidden and incomplete information in flow sheets, is to keep the amount of detailed information at a minimum. This is done in order to make the process map readable and easy to understand. An experienced user has on the other hand knowledge about this and knows about the details that are not given.

Hierarchical Modularization

In a process flow sheet description we have a natural and graphical description of the process. The process is modularized in process units and connections. A process unit can be a physical object like a pump, a heat exchanger or a chemical reactor. It can also be a set of subunits with a specific function. An example of a unit composed of a set of subunits is the distillation column, which is composed of the column, reboiler, condenser reflux drum, reflux pump and control valves. The connections represent flow pipes, which means that process medium is flowing from one unit to another unit.

Modularization of the Process Model: A large process model is very hard to overview and it is both convenient and natural to decompose the process model into submodels. We need a *modularization concept.* With this concept we can decompose the process model into submodels with more detailed information about this subprocesses. This means that a process model is composed of submodels and connections. The submodels contain descriptions of the submodel interactions and the submodel behaviours. The main drawback with a two level model description, like flow sheets, is the problem with abstraction. There is no possibility to choose model complexity. The process model description must be either at the connection description level or at the submodel level. It is hard to use a two level description in a efficient way.

Hierarchical Submodels: One way of develop the flow sheet description in a computerized modelling environment is to allow multiple level descriptions of the process flow sheet. This means that a unit symbol can have an internal structure that can be described as an internal flow sheet. In other words submodels can also be described with submodels and connections. A model can be composed of submodels, which can be composed of submodels etc. We can call this property *hierarchical submodel description.*

A process model can be described with two different kinds of submodels.

- *Composite models* are composed of submodels and connections between them. A natural description of a composite model is a graphical description like a flow sheet.
- *Primitive models* are submodels describing the behaviour of process components. Primitive models contains textual descriptions like equations.

If a process model is modularized and is described by a flow sheet, it is an example of a composite model.

Information Zooming: In the process flow sheet there are hidden or incomplete information. There is a trade off between information density and readability. In a multiple level description of a process model we have a possibility to go down into a submodel in order to find more detailed information. We have a property that is called *information zooming.* When zooming in on a submodel more detailed information shows up and we have solved the problem with hidden information. Hierarchical descriptions and information zooming are described in Elmqvist (1985) and in Elmqvist and Mattsson (1989).

Figure 2.5 shows how hierarchical submodel descriptions with information zooming can look like. Here we zoom in on a distillation column in the separation process part. In the third step we see the internal structure of subunits in the distillation unit. In the following step the structure of the actual column shows up and in the final step the serial connected tray descriptions are shown.

Decomposition of the Process Model

The most natural way of decomposing a large process model is into physical process object submodels with physical connections. One drawback with this approach is that a large process description becomes complex with a lot of components and therefore becomes unreadable.

Let us decompose the model in a hierarchical way, which can be called *process structure decomposition.* The process model is first decomposed into subprocess models, like preparation, reaction and separation processes. In the second layer the subprocess models are described by submodels for subsubprocesses, unit operations or physical objects.

In our process the preparation subprocess is decomposed into only process objects like pumps, valves etc. The reaction subprocess is decomposed

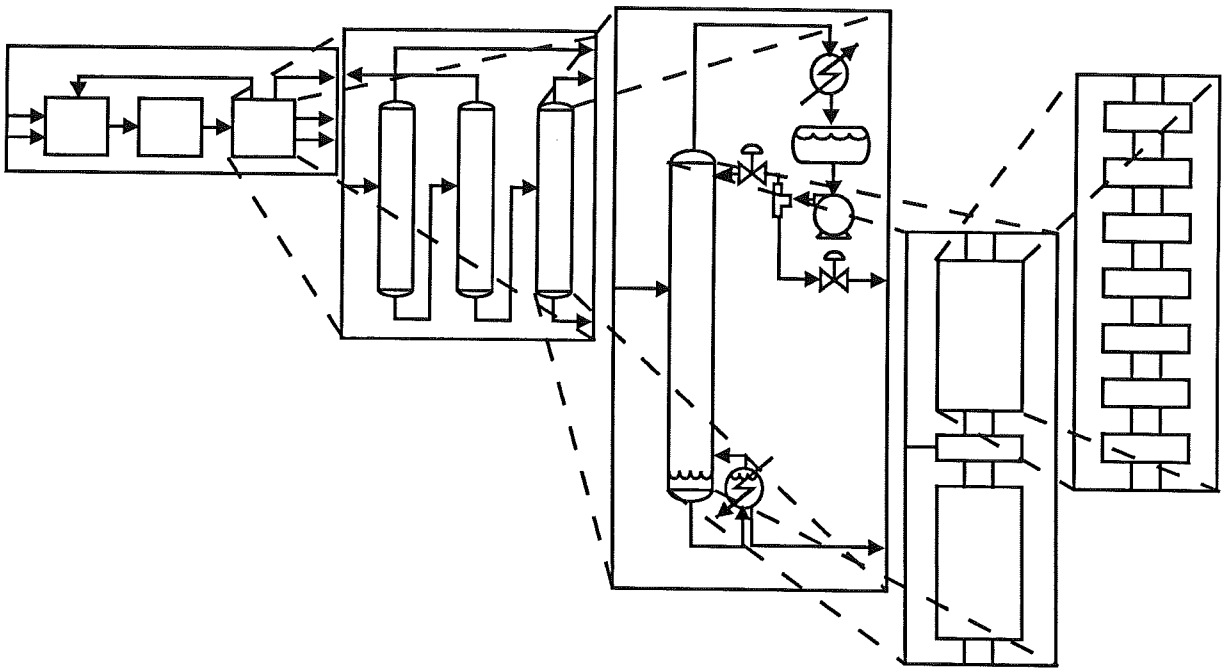


Figure 2.5 Hierarchical description and information zooming on a distillation column in the chemical process.

into a bubble tank reactor submodel and two tubular reactor submodels. They are discussed in more detail in the following sections. In the separation subprocess we find submodels for unit operations, namely three distillation columns. A distillation column is in its turn composed of physical objects with internal structures. Distillation column models are discussed in Section 2.5.

2.3 The Bubble Tank Reactor

The modelling of the bubble tank reactor will now be discussed. After that we postulate an internal structure of primitive models. The actual equations are not the important thing in this chapter. We are considering a complete plant in order to find model structures.

The gas feed of the raw materials, A and B, enter the tank reactor in a gas inlet at the reactor bottom. The liquid in the tank will absorb the gas. In the liquid a reaction between A and B is catalysted. The reaction products, C and D, leave the reactor in gas phase at the reactor top together with unreacted A and B. The reaction is also temperature dependent and the reaction is weakly endothermic. Heat is applied in an external heat exchanger in a liquid recycle loop in order to keep the

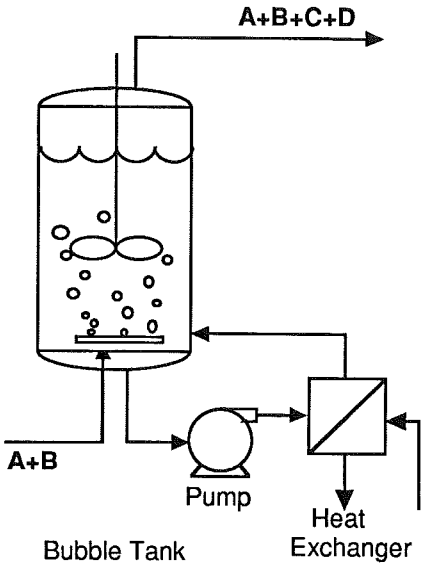


Figure 2.6 The bubble tank reactor system.

temperature at the operation point. In the long run the catalytic activity in the liquid decreases. This means that the catalyst must be regenerated.

Reactor Tank Model

The reactor tank is assumed to have constant liquid level. This means that the total reactor mass balance is at steady state. We also assume that the reaction is limited by the reaction velocity and not by mass transport phenomenon. This means that we assume homogeneous reaction in the reactor vessel and perfect mixing. In other words we assume an ideal reactor tank model. The reaction velocity is proportional to the concentration of the raw materials. The components evaporate at the liquid surface and leave the reactor at the gas outlet in the top. The evaporation is modelled as an ordinary mass transport.

Energy is consumed in the endothermic reaction. In a heat exchanger, heat is applied to the liquid. The liquid is pumped through the recycle heat exchanger loop. Assume that the gas flow does not effect the energy balance. The enthalpy content in the gas phase is assumed to be the same in the inlet as in the outlet.

The reactor tank is described by five differential equations, namely four for the contents of the four chemical components, A, B, C and D, and one for the energy content. The first equation below is a chemical component

mass balance and the second is an energy balance.

$$\begin{aligned}
\text{Accumulation} &= \text{In} && -\text{Out} && +\text{Production} \\
\frac{d(Vc_j)}{dt} &= q_v c_{j,v} && -N_j A && +Vr_j \\
\frac{d(\rho V C_p T)}{dt} &= \rho q_h C_p T_h && -\rho q_c C_p T + \Delta H_{reac} V r_D
\end{aligned}$$

The index j is used to indicate the four components A, B, C and D and the index v represents the vapour flow into the reactor tank. The index h and c represents flow from (hot) and flow to (cold) the heat exchanger respectively. The terms N_j and r_j are the mass transfer flux and the chemical reaction velocity. The mass transfer flux is proportional to the concentration gradient over the phase boundary as:

$$N_j = k_{Lj}(c_{j,equil} - c_j)$$

The k_{Lj} is the proportional factor. The reaction velocity is assumed to be independent of the product concentrations and temperature. It can be expressed as an first order reaction in the reactants, A and B.

$$r_A = r_B = -r_C = -r_D = -k_{reac} c_A c_B$$

A similar model has been developed in Luyben (1973).

Primitive Model Structure

The reactor tank model is an example of a primitive model description. The behaviour description of primitive models is a set of equations. These equations describe mathematically how the object is assumed to behave. The behaviour description of the reactor tank above is composed of five balance equations.

A submodel interacts with surrounding submodels. The tank model requires a number of variables describing the state of the feed, interaction with the heat exchanger and the product stream. The communication variables are associated with variables in other submodels through connections. In the tank example above we assumed that connections are the same thing as process flow pipes. This means that the tank has four connections to other submodels. One physical connection like a flow pipe contains more than one communication variable.

Inside our tank model description we now need a concept that can cluster communication variables together. We call this concept a *terminal.* The liquid out flow terminal in the tank model contains the communication variables for the flow, temperature and concentrations. This concept is important because it makes it possible to describe the process model in a flow sheet like description. The description of submodel structure and submodel behaviour can be decoupled. In a submodel we define the terminals and in a super-model we define the connections.

When users are working with process models they need a way to change the models in order to use them in a particular application. In the tank model above a user wants to be able to change reaction coefficients, tank volume etc. A variable that can be changed by the user, is called a *parameter.* A parameter is a time invariant variable during simulation but can be changed by the user between simulations. A proper parameterization of models makes the them flexible and facilitates the work for the user of library models.

A primitive model is therefore composed of three major parts: terminals, parameters and behaviour descriptions.

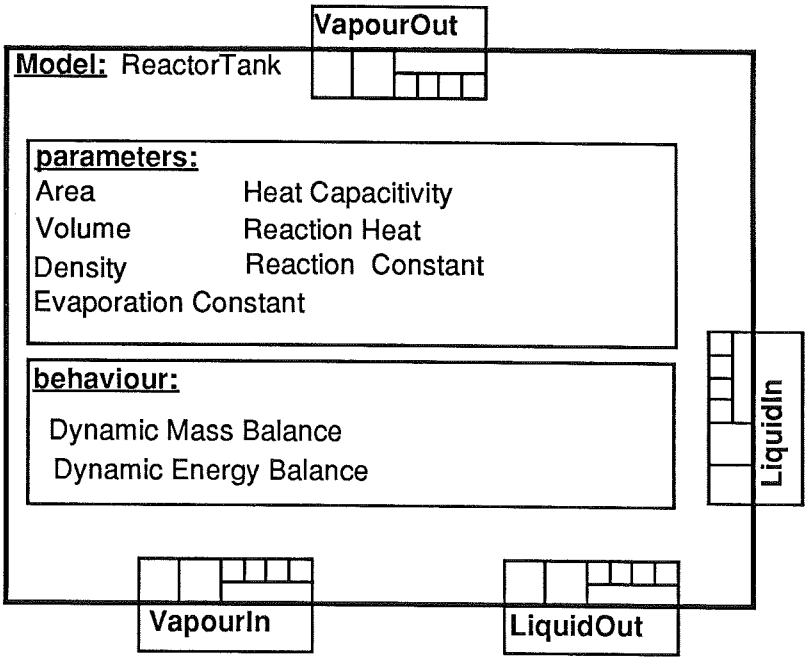


Figure 2.7 The structure of the reactor tank model with terminals, parameters and behaviour description.

2.4 Common Process Components

In a chemical process the same type of components appear over and over again. Examples of common process components are pumps, valves and heat exchangers. A *model type* concept that makes it possible to reuse models is important in process modelling. Below follows simple descriptions of a valve, a pump and a heat exchanger. They are used more then once in the process model.

Valve

The valve is assumed to have static mass balance and static energy balance. This means that the flow, chemical composition and temperature out from the valve are equal to the the ones into the valve. The valve is used to create a pressure drop, which decrease the flow through the valve.

$$
\begin{aligned}
q_{out} &= q_{in} \\
c_{j,out} &= c_{j,in} \\
T_{out} &= T_{in} \\
\Delta P &= P_{in} - P_{out} \\
\Delta P &= \frac{k_{lf}}{2A^2} q_{in} |q_{in}|
\end{aligned}
$$

We see that the pressure drop is proportional to the square of the flow and the proportional factor is the loss factor, k_{lf}. A valve is connected to the surrounding submodels through two terminals describing the two pipe connections. These terminals must express flow, composition, temperature and pressure. A control valve has also a terminal describing the position signal from a servo motor, which can change the loss factor, k_{lf}, of the control valve.

Pump

In a centrifugal pump we can assume, like in the valve model, static mass and energy balances. In a pump we apply energy to create a flow and

therefore we create a pressure rise.

$$
\begin{aligned}
q_{out} &= q_{in} \\
c_{j,out} &= c_{j,in} \\
T_{out} &= T_{in} \\
\Delta P &= P_{in} - P_{out} \\
q_{out} &= Q_{max} - (\frac{\Delta P}{k_{pump}})^2
\end{aligned}
$$

The descriptions of the valve and the pump are very similar. The major difference is the flow and the pressure relations. The pump has two terminals that represents the pipe connections. The terminals are equal to the pipe terminals in the valve model above.

To simulate a system with pumps and valves that have static mass and momentum balances requires numeric algorithms that handle algebraic equations. One example is a system, seen in Figure 2.8, with a pump and a valve connected in series. The inlet and outlet of the system is connected to known pressures, p_{ref}. The pump model needs to know the inlet and outlet pressure to calculate the flow through the system, but the outlet pressure of the pump, p, is unknown. The valve needs to know the flow to calculate the pressure drop or indirectly the inlet pressure of the valve, p, which is equal to the outlet pressure of the pump.

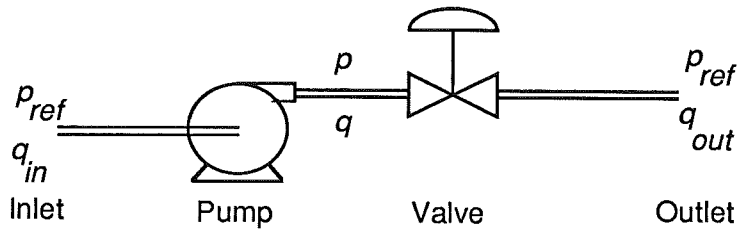


Figure 2.8 A system with a pump and a valve.

Heat Exchanger Model

The heat exchanger is modelled with two perfect mixed conducts on each side of a heat transfer boundary. The heat transport is described by a static heat transfer equation. The density and heat capacity are assumed to be constant in the model. The flow dynamics in the heat exchanger loop is much faster than the energy dynamics and it is therefore assumed

to be static. The index c and h represents the cool side and the hot side of the heat exchanger. This result in two energy balances expressing the temperature on each side of the boundary.

$$\begin{array}{lll} \text{Accumulation} = \text{In} & & -\text{Out} \\ \frac{d(\rho_c V_c C_{p_c} T_c)}{dt} = \rho_c q_c C_{p_c} T_{c_{in}} + Q_{trans} & & -\rho_c q_c C_{p_c} T_c \\ \frac{d(\rho_h V_h C_{p_h} T_h)}{dt} = \rho_h q_h C_{p_h} T_{h_{in}} & & -\rho_h q_h C_{p_h} T_h - Q_{trans} \end{array}$$

Static heat transfer equation:

$$Q_{trans} = kA(T_h - T_c)$$

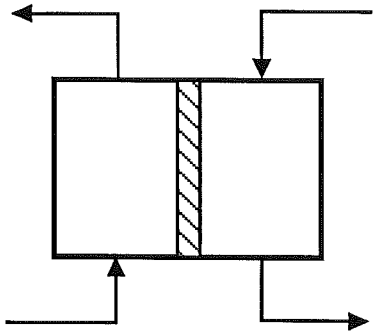


Figure 2.9 The heat exchanger described as two conducts with a heat transfer boundary.

A heat exchanger model similar to this can be found in Luyben (1973). The structure of the heat exchanger model is four terminals, eight parameters and one behaviour description with two differential equation and one algebraic equation. In the heat exchanger model above we assume that the composition and the pressure is the same in the flow that enters and in the flow that leaves. In order to be able to connect a heat exchanger with a pump or a valve compatible terminals are required The terminals of a heat exchanger must therefore express flow, composition, temperature and pressure.

Common Descriptions

In a process model there are a number of components that are similar. Typical examples are pumps, valves, tanks and heat exchangers. This requires a *model type* concepts, which makes it possible to reuse the model description many times with different parameter values.

We have also seen that model components like terminals and behaviour are used in different models. A natural way is to generalize the model type concept also to capture model components. A terminal type describing a pipe connection with flow, composition, temperature and pressure, which can be used in the different models will simplify the model development process.

2.5 The Distillation Column

In this section we discuss modelling of a distillation column. The major point is to make useful parameterizations of the column in order to facilitate the reusability of models. A distillation column is shown in Figure 2.10. The unit is composed of the actual column and its support equipments, like reboiler, condenser, reflux drum, reflux pump and control valves. The column is composed of a number of trays put on top of each other.

A Tray Model

A tray is a column component where the liquid and vapour interact. The liquid that is flowing downward in the column meets the vapour. Vapour rises up through the column and it is forced to interact with the liquid. The interaction is described by a phase equilibrium relation. The model chosen here is based on component mass balances and an energy balance.

$$\begin{aligned}
\text{Accumulation} &= \text{In} && -\text{Out} \\
\frac{d(mx)}{dt} &= w_{Lin}x_{Lin} + w_{Vin}y_{Vin} && -w_{Lout}x - w_{Vout}y \\
\frac{d(mC_pT)}{dt} &= w_{Lin}C_pT_{Lin} + w_{Vin}C_pT_{Vin} && -w_{Lout}C_pT - w_{Vout}C_pT
\end{aligned}$$

The variables x and y are mass fraction in the liquid and in the vapour. The mass of a chemical component is therefore described by mx. The

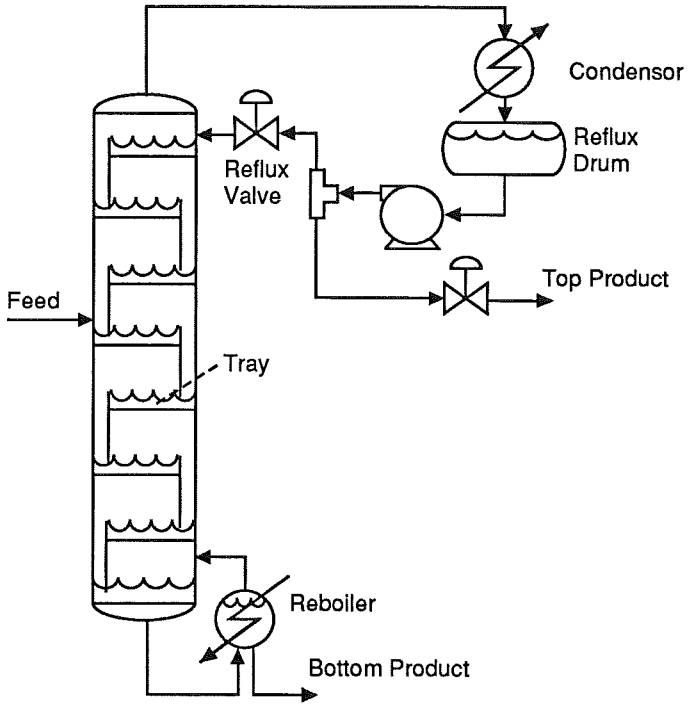


Figure 2.10 The structure of a tray based distillation column.

mass flow has the notation w. Note that the first differential equation uses matrix notation for expressing the chemical composition dynamics. It has the dimension of five in order to describe the composition of A, B, C, D and E. The composition variables x and y are vectors with the length of five.

The phase equilibrium can be described by following equations:

$$Py_j = p_j^o \gamma_j x_j$$

$$logp_j^o = A_j - \frac{B_j}{C_j + T}$$

The first equation is a so called state equation, which relates the total pressure to the partial pressures of each chemical component. The symbol γ is the activity factor and is equal to one if the medium is assumed to be ideal. The second equation is called Antoine's law and calculates the partial pressure for a chemical component. A, B and C in this equation are the Antoine's factors, which depend on the chemical component. This means that the vapour composition can be calculated from liquid composition, pressure and temperature.

Tray models are presented in Luyben (1973) and in Stephanopoulos (1984). The phase equilibrium model can be found in Zacchi et al (1984).

Implementations of a distillation column model using different modelling languages can be found in Nilsson (1987).

Regular Structure

The actual column is composed of a number of trays that are put on top of each other. Typical distillation columns can have twenty to hundreds of trays, which are connected to each other in a particular way. To create models with structures like this is boring and it is easy to make a mistake. A *regular structure mechanism* that can automatically generate this structure is therefore of major importance in process modelling. It should be possible to create a composite model with submodels that are connected to each other in a particular way. One suggestion is a vector of submodels with the parameters submodel name and vector length. The graphical interpretation of this is not clear but with some additional commands should the regular structure mechanism automatically generate a graphical representation, like the one in Figure 2.11.

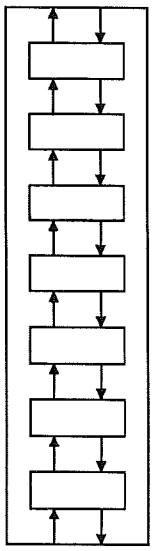


Figure 2.11 The regular structure of trays in a column.

Parameterization of the Column

We mentioned above that one of the major parts of a model is the parameters. Parameters can be used in order to adapt the model to a new application. To create flexible models that can be reused in different applications we need to have a number of ways to parameterize models. We are now going to look at the parameterization problems of the column.

Super-model Parameter Assignment: We have seen that a column is composed of a number of trays. Often these trays are designed in the same way. They are often equivalent. Assume that we want to change a design parameter of the tray model in the column. This can be done in three ways.

- Change the parameter in every tray submodel in the column.
- Change the parameter in the tray model type and recompile the column model.
- Change a super model parameter that in its turn automatically changes the tray submodel parameters.

The third way is the most convenient but it requires that the modeller has thought about the parameterization problem. In other words we want to have a *super-model parameter* that can be bounded to parameters in the submodels.

Structure Parameterization: Another need in modelling of a distillation column is to allow parameterization of the number of trays in the regular structure of column. Assume that we have a regular structure mechanism discussed above. One way to parameterize this is by making the variable describing the number of trays in the structure as a parameter. This is not an ordinary parameter, because it influence the structure of the model and it can not be changed in the same way as ordinary parameters.

Submodel Parameterization: If we continue to assume a regular structure mechanism then it is also interesting to parameterize the submodel type in the regular structure. Studies of different tray models in a column with a number of trays are easily done if it is possible to parameterize the submodel types. An interesting way of handling this problem is to introduce plug-in submodels. This can be done by allowing the submodel type to be a parameter. This parameter can be set by the user in a straightforward parameter assignment fashion.

Medium and Machine Parameterization: An advanced way of doing a parameterization is to separate the descriptions of the medium and the machine. This means that it must be possible to change the medium description independently of the tray machine description. As discussed in the tray model subsection the description of the medium and of the tray is separated in different expressions.

2.6 The Tubular Reactor

The model of a tubular reactor is now discussed. It shows us how to handle an infinite dimensional system. In the catalytic tubular reactor the process medium is forced to go through a tube filled with catalytic particles. In the tubes the gas-solid reaction occurs as discussed in Section 2.1. This second reaction in exothermic. The released reaction heat is removed by heating the feed stream. The feed flows outside the tubes in the opposite direction to the tube flow, see Figure 2.12.

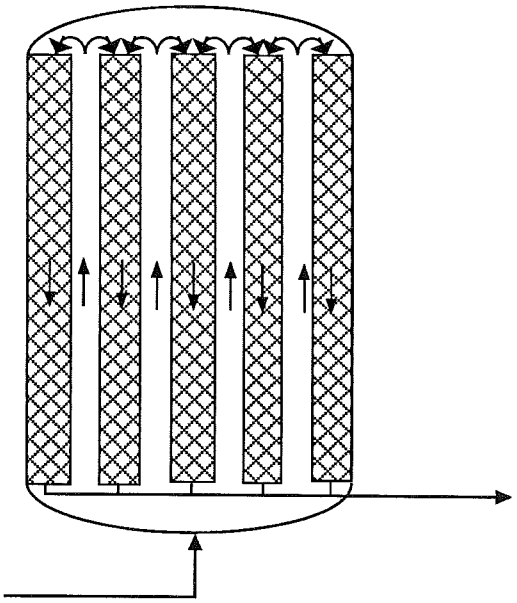


Figure 2.12 A description of the fixed bed catalytic tubular reactor.

If we cut out a thin slice of the tube and put up the balance equations we get the same description for the slice as for an ideal tank reactor. Let us assume a constant section area of the tube.

$$
\begin{array}{rlll}
\text{Accumulation} = & \text{In} & -\text{Out} & +\text{Production} \\
\dfrac{d(A\delta x c_j)}{dt} = & q_{in}c_{j_{in}} & -q_{out}c_j & +SN_j \\
\dfrac{d(\rho A\delta x C_p T)}{dt} = & \rho q_{in}C_pT_{in} & -\rho q_{out}C_pT - Q + \Delta H_{reac}SR_v & \\
\dfrac{d(\rho A_f\delta x C_p T_f)}{dt} = & \rho q_{in,f}C_pT_{in,f} + Q & -\rho q_{out,f}C_pT_f &
\end{array}
$$

The tube cross area is called A and the gas-solid interface is called S. The index f refers to the feed stream. Here we assume that the diffusion

into the catalyst is a static mass transfer model and that the chemical reaction occur on the surface of the catalyst. We can therefore assume that the mass transfer rate is equal to the reaction rate. No accumulation at the boundary or on the catalyst surface are assumed.

$$
\begin{aligned}
N_j &= k_{Vj}(c_{j,surface} - c_j) \\
R_v &= k_y c_{A,surface} c_{D,surface} \\
N_j &= [R_v; 0; -R_v; R_v; -R_v]
\end{aligned}
$$

The reaction is first order in the reactants, A and D. Heat, Q, is transfered from the hot side to the cold side. The heat transport can be described by a static transport equation as the one in the heat exchanger model in Section 2.4. If we now let the thickness of the slice become infinite small we get following balance equations:

$$
\begin{aligned}
\frac{\partial c_j}{\partial t} &= -\frac{\partial(vc_j)}{\partial x} + \frac{1}{A}\frac{\partial(SN_j)}{\partial x} \\
\frac{\partial T}{\partial t} &= -\frac{\partial(vT)}{\partial x} - \frac{1}{\rho A C_p}\frac{\partial Q_{trans}}{\partial x} + \frac{\Delta H_{reac}}{\rho A C_p}\frac{\partial(SR_v)}{\partial x} \\
\frac{\partial T_f}{\partial t} &= \frac{\partial(vT_f)}{\partial x} + \frac{1}{\rho A_f C_p}\frac{\partial Q_{trans}}{\partial x}
\end{aligned}
$$

This is a set of partial differential equations (PDE's). Models for tubular reactors that are similar to the one described here are developed in Luyben (1973).

Finite Dimensional Description

Behaviour descriptions like the ones above are quite common in process engineering systems. The problem with partial differential equations or distributed parameter systems is that they are hard to solve and simulate. We are therefore interested in simplifications of this model behaviour description.

One way to simplify the PDE system is to go back one step in the development and discretize the tube with a number of small slices. A discretization of the model like this can be called *model approximation decomposition*. This is done in order to approximate the model. Each slice have four terminals. Two describing the feed stream and two for the tube stream. Each side of the heat transfer boundary have a behaviour

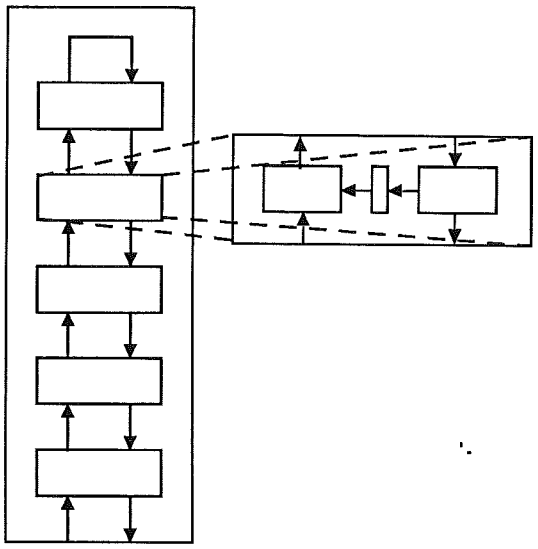


Figure 2.13 The regular structure in a discretized tubular reactor.

like a tank with perfect mixing. This means that we once again need a regular structure mechanism. A decomposition like this is seen in Figure 2.13.

2.7 Summary

In this chapter we have tried to find *model structuring concepts.* The knowledge of model structures can be used in order to facilitate model development and reuse.

We have seen that models can be of two different types.

- *Primitive models* define their behaviour by symbolic descriptions like equations.
- *Composite models* is described with a structure of submodels and connections between submodels. Submodels can be either primitive or composite models.

Composite models can be used to create a *hierarchical submodel description.*

Primitive and composite models are assumed to have similar internal model structure. The internal structure of a model is composed of three major component types, namely terminal, parameter and behaviour components.

- *Terminals* are model components that can be used to describe the interaction with surrounding submodels.
- *Parameters* are the only variables that can be changed by the user in order to adapt the model to new applications.
- *Behaviour components* are the actual model description. It can be a structure of submodels (composite) or a set of equations (primitive).

Model type description is another important concept in process modelling. Models are often used more then once in process models. Common process components, like pumps and valves, are used over and over again the same model and it is therefore interesting to have a way to make a common model description. This concept should also include model components.

Mechanisms that create *regular structures* is also important in process modelling.

We have also defined *process structure decomposition* as a way to create a model hierarchy based on the structure of physical process objects. *Model approximation decomposition* can be used to approximate models that are to complex to simulate.

3. Object-Oriented Modelling

Structured chemical process modelling was discussed in the previous chapter and in this chapter the object-oriented approach is presented. Object-oriented methodology is a way of structuring data into objects. Inheritance is an important concept. One object can inherit properties from another predefined object. A system that uses an object-oriented approach to modelling is the "System Engineering Environment, SEE", developed at the Department of Automatic Control (Andersson, 1989a, or Mattsson and Andersson, 1989ab) and a first prototype is implemented in KEE (1988). A prototype has been developed for simulating dynamic behaviour. The basic architecture of the prototype is composed of a model database, and tools that operate on models. Tools that can operate on the model database can be for instance a simulator, numerical algorithms, symbolic manipulators and interfaces of different kinds. A formal language that describes the object-oriented model representation in SEE is developed by Andersson (1989b) and it is called Omola.

In the SEE-prototype there is a simulator tool for DAE-systems (differential/algebraic equations, see Mattsson, 1988) and a model/user interface. The user interacts with the model representation through the SEE-interface. The interaction is based on a interactive dialog using commands, menus and graphics. The models can then be simulated with the SEE-simulator which can extract the equations from the model representation. This architecture allows an object-oriented approach to model development and an equation-oriented approach to the problem solving.

A general discussion about object-oriented methodology and inheritance is found in Section 3.1. In the following section, Section 3.2, an introduction to Omola is made and in Section 3.3 a short summary of the Omola model structuring concept is found.

3.1 Object-Oriented Methodology

Object-oriented programming has been an increasingly popular methodology for software development. Increased programmer productivity, in-

creased software quality and easier program maintenance are the objectives for this new methodology. Object-oriented programming supports these objectives by facilitating modularization and reuse of code. Ideas from object-oriented programming that are useful also for model representation are encapsulation and modularization into objects, instantiation of classes into objects and the inheritance concept. A good and brief introduction to object-oriented programming is given by Stefik and Bobrow (1986).

Modularization and Instantiation

An object has an unique identity within the system and it contains a collection of attributes. This is a way to encapsulate, modularize and structure data. A set of similar objects is described by a class. The class can be instantiated into any number of objects. In many systems classes are themselves represented and manipulated as objects.

Inheritance

A class can be a subclass of another class, a super-class. This is a "IS A"-relation and it dictates the rules of inheritance. The subclass inherits attributes from its super-class. In this context we are only dealing with single inheritance, which means that a subclass only has one super-class. A subclass can be assigned local attributes. If a local attributes and an inherited attribute have the same name the local attribute overwrites the inherited one.

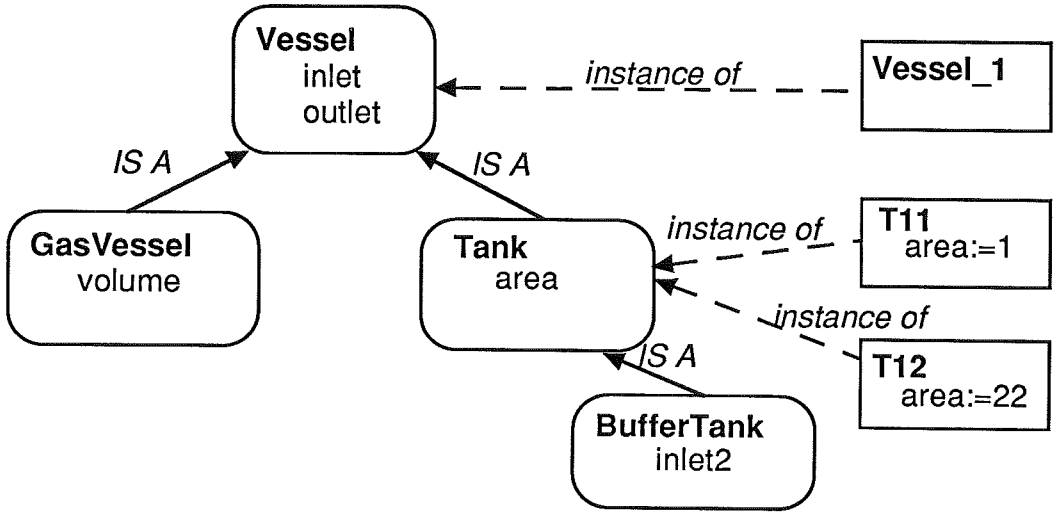


Figure 3.1 An example of an object class hierarchy showing inheritance and instantiation.

A small example that show the inheritance and instantiation concepts is found in Figure 3.1. In the figure we find four class objects and three instantiated objects. The super-class **Vessel** contain two attributes, namely inlet and outlet. The subclasses will inherit these attributes. The subclass **Tank** has also a local attribute is defined with the name area. This means that the class **Tank** contains three attributes: inlet, outlet and area. **BufferTank**, which is a subclass to **Tank**,has four attributes. The same as tank plus the local attribute inlet2. The vessel class is instantiated into **Vessel_1**, which means that this object has two attributes. The tank class is instantiated twice into **T11** and **T12**. In the **T11** object the attribute area is given a value, which means that the old inherited value is overwritten.

3.2 Introduction to Omola

Omola is a formal language describing the object-oriented model representation in SEE. Models are represented as objects in Omola, which are subclasses of previously defined super-classes. A subclass will inherit properties from its super-class. Omola has single inheritance, which means that a subclass only has one super-class.

In Omola there are a number of predefined super-classes that can be used to describe models. There is a predefined super-class for models, called `Model`. Model components are also represented as objects. For model components there are classes for terminals, parameters and for behaviours, which are called realizations. The attributes of a model object are definitions of model component objects. Examples of model attributes are definitions of terminals, parameters and realizations.

Omola can describe both primitive and composite models, which means that Omola supports hierarchical submodel description. For more detailed information about Omola see Andersson (1989b).

Primitive Models

In the example below we see the different model components, terminals, parameters and realizations. All model components are objects predefined or defined in the model. The `Tank` model requires four predefined objects, namely `Model`, `Terminal`, `Parameter` and `Primitive`.

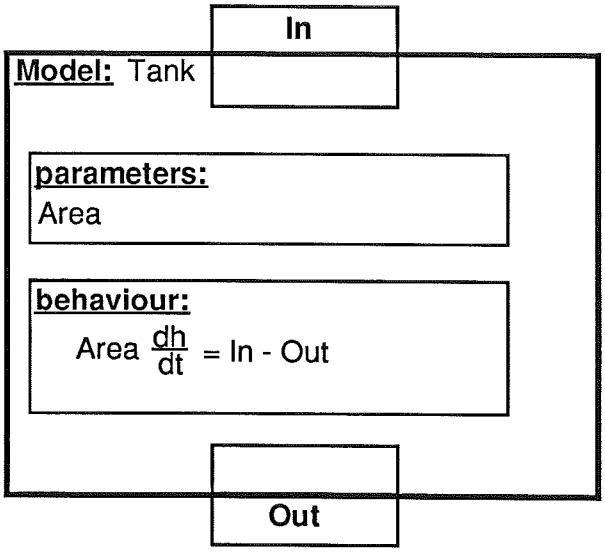


Figure 3.2 An example of a primitive model of a tank.

```
Tank IS A Model WITH
  terminals:
    Inlet IS A Terminal;
    Outlet IS A Terminal;
  parameter:
    Area IS A Parameter WITH value := 1; END;
  realization:
    Behaviour IS A Primitive WITH
      variable:
        Height TYPE Real;
      equation:
        Area*DOT(Height) = Inlet - Outlet;
    END;
END;
```

The `Tank` object is a subclass of `Model`, which is a predefined class in Omola. The model components are structured into categories indicated by tags. Each model component is a subclass of some globally defined super-class. The terminals `Inlet` and `Outlet` are subclasses of `Terminal`. The `Area` attribute is a specialization of the `Parameter` class. It is specialized in order to get a value of 1. The last model component `Behaviour` is a specialized subclass of `Primitive`. The realization model component has an internal structure of attributes, namely one variable object and one equation object. The variable object is of the type real (classless) and the equation object is a differential equation. `DOT(x)`

means the time derivative of the variable x, $\dot{x}$ or $\frac{dx}{dt}$. A graphical description of the primitive model Tank is found in Figure 3.2.

Composite Models

The Tank example shows how Omola handles primitive models. A composite model has a similar structure. The only difference is another realization super-class called Structure. In a structure object we can define submodels and connections as attributes. Below follows an example of a simple process with a valve, a tank and a pump described in Omola. In Figure 3.3 can the same composite model be seen in a block diagram description.

```
TankSystem IS A Model WITH
  terminals:
    Inlet IS A Terminal;
    Outlet IS A Terminal;
  realization:
    Flowsheet IS A Structure WITH
      submodels:
        Valve1 TYPE Valve;
        Tank1 TYPE Tank;
        Pump1 TYPE Pump;
      connections:
        Inlet          AT Valve1.Inlet;
        Valve1.Outlet  AT Tank1.Inlet;
        Tank1.Outlet   AT Pump1.Inlet;
        Outlet         AT Pump1.Outlet;
    END;
END;
```

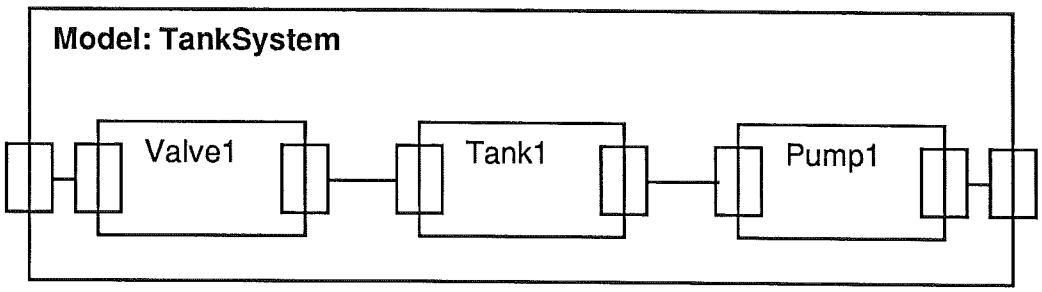


Figure 3.3 The tank system example describe as a composite model.

We see that the model component `Flowsheet` is a subclass of the `Structure` super-class. It allow model components of the submodel and connection categories. Under the submodel tag subclasses of the models `Valve`, `Tank` and `Pump` are defined. These super-classes are predefined. Connections are defined between terminals of the model and terminals of the submodels.

Terminals

An important concept in structured modelling is the modularization and encapsulation of a model into submodels. The submodels only interact with other submodels through terminals. The interaction between submodels are discussed in more detail by Mattsson (1988, 1989).

In Omola it is possible to structure terminals. The basic elements of hierarchical structured terminals are called simple terminals. The simple terminal class is predefined and have a number of attributes, which represent value, default value, quantity, unit of measure and direction. This can be seen in the example below. The meaning of a connection between two simple terminals can be interpretated in two ways. *Through terminal* assumes a direction. They follow the "zero-sum" law. A through terminal is either **in** or **out**. A connection between two *across terminals* means that they should be equal. In Mattsson (1989) and in Andersson (1989b) are a more detailed description of the terminal concept given.

```
SimpleTerminal IS A Terminal WITH
  attributes:
    value TYPE Real;
    default_value TYPE Real;
    quantity TYPE Quantity;
    unit TYPE string;
    direction TYPE (In,Out,Across);
END;
```

A structured terminal (record terminal) describing a process pipe is found in `Pipe_In_Terminal` example below and is also seen on the Figure 3.4. It is composed of four components, describing the flow, chemical composition, temperature and pressure. The chemical composition is described with a vector terminal, `Composition_Terminal` with the length of two and with a component terminal describing the concentration. The basic

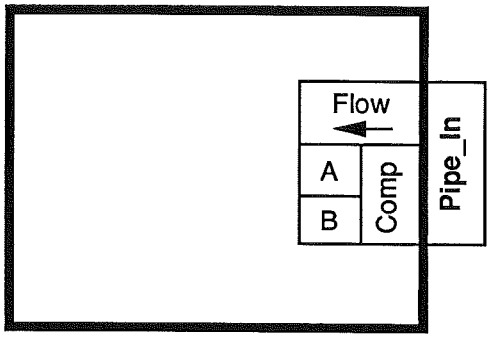


Figure 3.4 A graphical description of the structured terminal `Pipe_In`.

simple terminals is seen in Figure 3.5, which shows the terminal class hierarchy.

```
Pipe_In IS A RecordTerminal WITH
  components:
    Flow IS A Flow_In_Terminal;
    Comp IS A Composition_Terminal;
END;

Composition_Terminal IS A VectorTerminal WITH
  Length := 2;
  CompType IS A Concentration_Terminal;
END;
```

Realizations

A realization class contains a collections of attributes describing the realization components. A primitive model realization contains definitions of variables and equations. In a composite model realization there is a set of submodel definitions and connection definitions. Realization objects can not exist outside a model and must therefore be inherited through the model inheritance.

Below follows an example of how realization objects can be inherited. The behaviour of `Dynamic_Object_1`, the `Dynamic_Mass_Balance` realization object, is inherited and specialized in the subclass `Dynamic_Object_2`.

```
Dynamic_Object_1 IS A Model WITH
  terminals:
    Tin IS A Pipe_In_Terminal;
```

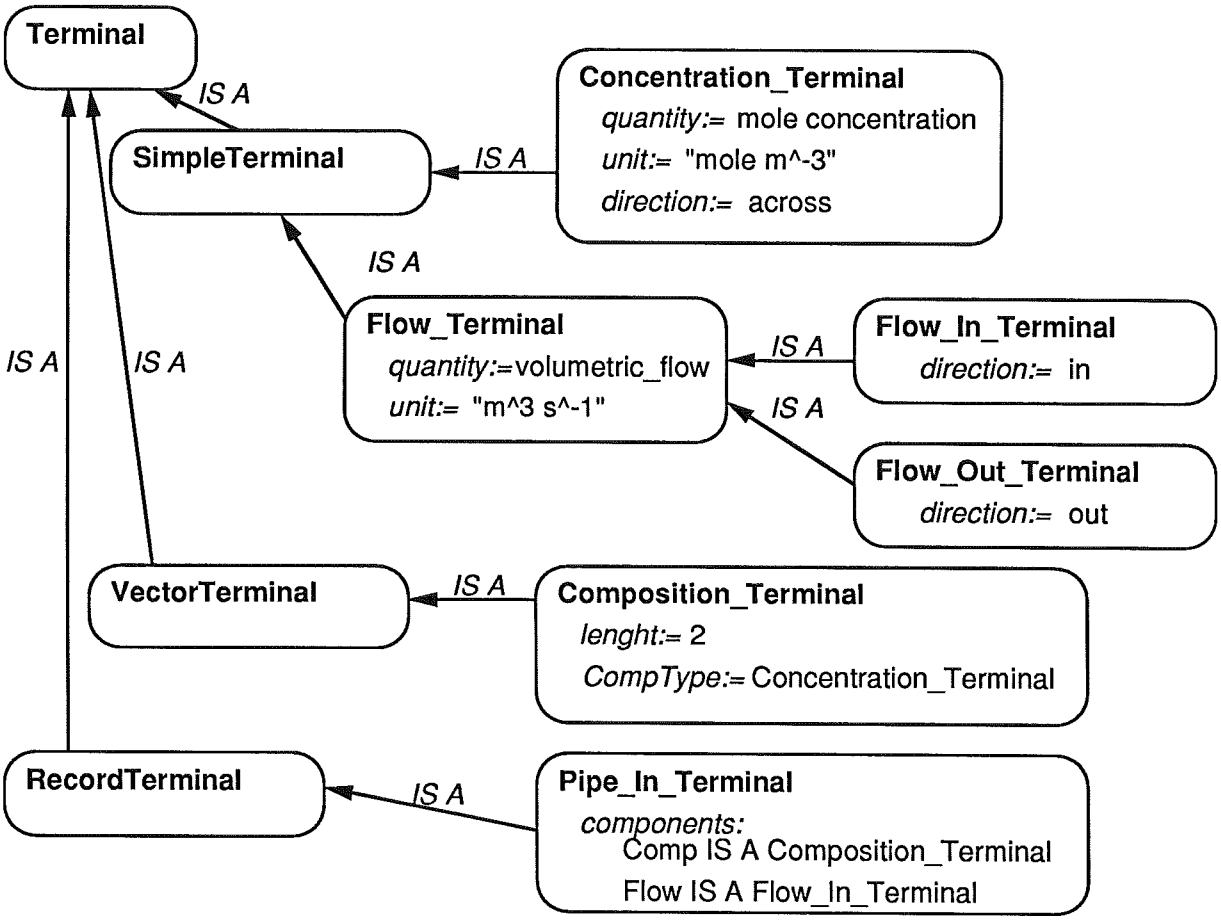


Figure 3.5 The terminal class hierarchy of the `Pipe_In` example.

```
    Tout IS A Pipe_Out_Terminal;
  parameter:
    density IS A Parameter;
  realizations:
    Dynamic_Mass_Balance IS A Primitive WITH
      variables:
        mass, mole TYPE Real;
      equations:
        DOT(mass) = density*(Tin.Flow - Tout.Flow);
        DOT(mole) = Tin.Flow*Tin.Composition -
          Tout.Flow*Tout.Composition;
        Tout.Composition = mole*density/mass;
    END;
END;
```

If we want to create a model that also have a dynamic energy balance then we can use inheritance. We define a subclass of `Dynamic_Object_1` that

inherit the realization object Dynamic_Mass_Balance. The realization can then be specialized in the desired way. This is done in the example below.

```
Dynamic_Object_2 IS A Dynamic_Object_1 WITH
  parameter:
    Cp IS A Parameter;
  realizations:
    Dynamic_Heat&Mass_Balance IS A Dynamic_Mass_Balance WITH
      variable:
        energy TYPE Real;
      equations:
        DOT(energy) = density*Cp*(Tin.Flow*Tin.Temp
          - Tout.Flow*Tout.Temp);
        Tout.Temp = energy/(mass*Cp);
    END;
END;
```

3.3 Model Structures in Omola

Omola captures the model structuring concepts discussed in Chapter 2. The structure realization, which is a way to implement the composite model concept, allows a hierarchical model description. We have also seen one way to use object-oriented methodology in modelling. Inheritance makes it possible to reuse previously defined models and change it in a desired way. This means that we have two different hierarchies in object-oriented modelling. The first is the hierarchical model description and the second is the model class hierarchy.

- *Model* is the main structural entity. A model object is composed of a number of attributes defining model component objects.
- *Terminal* is a model component class for describing the interaction with a connected submodel.
- *Parameter* dictates the interaction between the user and the model.
- *Realization* is a component object defining the behaviour of the model. The behaviour can be described as a primitive model (equations), or a structure of submodels (composite).

A realization is composed of a number of realization component definitions.

- *Variable* is a realization component defining the variable used in the equations.
- *Equation* is an component object in a primitive realization and describes relations between variables.
- *Submodel* is a model object used as a realization component object in a structure realization.
- *Connection* is a component of a structure realization that describe a relation between terminals.

Model structures are also discussed in Åström and Kreutzer (1986) and in Åström and Mattsson (1987).

4. Process Modelling in Omola

In this chapter we use Omola for modelling of the chemical process in Chapter 2. Model structuring concepts and object-oriented modelling using Omola are discussed. Some of the concepts are not directly included in Omola. Suggestions of what the missing concepts might look like are presented. The presentation of the process model in this chapter is not complete because it is used to exemplify concepts and ideas. The complete process model can be found in Appendix B. The process model is composed of about 100 class definitions. A small example of how to use Omola for process modelling is found in Nilsson (1989).

We are first going to do a top-down model decomposition of the process model, which is done in Section 4.1, and in Section 4.2 we are discussing common process objects and common model components. In the third section, Section 4.3, the tank reactor model is discussed. Distillation column models are discussed in the following section, Section 4.4. Distributed parameter systems such as the tubular reactor model are discussed in Section 4.5.

4.1 Top-Down Model Decomposition

This section focuses on modularization and hierarchical decomposition of the chemical process model.

Process Structure Decomposition

In the description of the process in Chapter 2 we decomposed the process into subprocesses. Each subprocess had a special meaning or function in the process. Then each subprocess was decomposed further into process components or unit operations. Unit operations are objects with a subfunction. They represents a process component or a set of process components. The lowest level in the hierarchy is the physical process objects. This structure of process objects is given by the topology of the

process. The hierarchy is based on the process structure. In Chapter 2 we called this way of structuring *process structure decomposition.*

Hierarchical structuring is very well taken care of in Omola. The model of the chemical process can be seen below. It is modularized into sub-processes which interact with each other through connections.

```
Process_Model IS A Model WITH
  terminals:
    Feed_A, Feed_B       IS A Vapour_In_Pipe;
    Product_C            IS A Vapour_Out_Pipe
    Product_D, Product_E IS A Liquid_Out_Pipe;
  realizations:
    Process_Flowsheet IS A Structure WITH
      submodels:
        Preparation IS A Preparation_Part_Model;
        Reaction    IS A Reaction_Part_Model;
        Separation  IS A Separation_Part_Model;
      connections:
        Feed_A                  AT Preparation.Feed_A;
        Feed_B                  AT Preparation.Feed_B;
        Preparation.Mixed_Feed AT Reaction.Feed;
        Reaction.Products       AT Separation.Feed;
        Separation.Recycle      AT Preparation.Recycle;
        Separation.To_Recovery AT Product_C;
        Separation.Product_D   AT Product_D;
        Separation.Product_E   AT Product_E;
    END;
END;
```

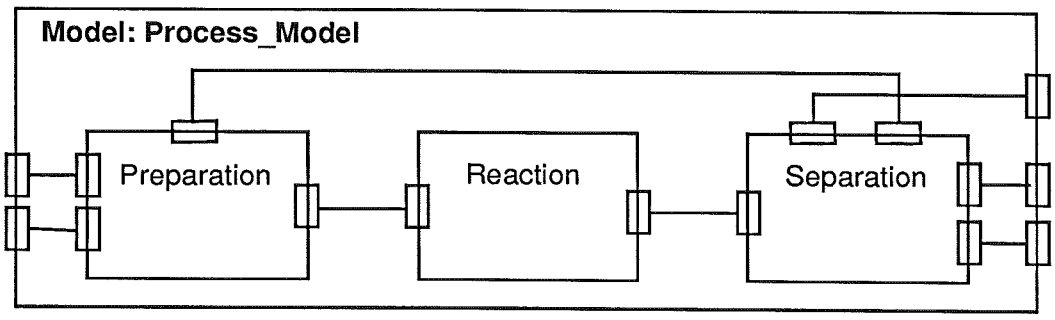


Figure 4.1 A block diagram showing the internal structure of the composite model `Process_Model`.

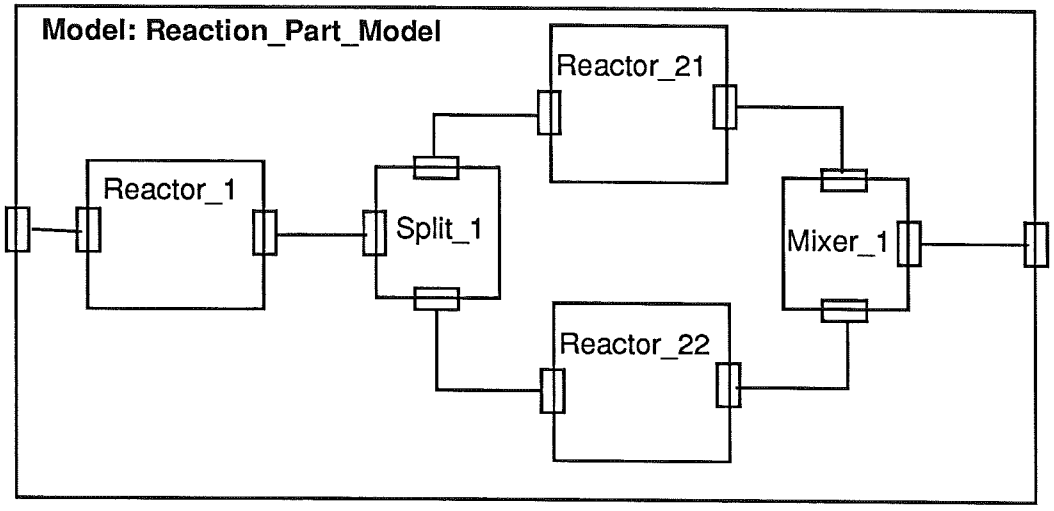


Figure 4.2 The internal structure of `Reaction_Part_Model`.

Composite models, like the process model, are natural to describe with graphics. Block diagram is one way to show the internal structure of a composite model. Composite models are from now on described with block diagrams. In Figure 4.1 the structure of `Process_Model` is shown. In the next level of the process structure hierarchy we find the submodel, like `Reaction`, which is seen in Figure 4.2. This submodel is a subclass of the super-class `Reaction_Part_Model`. It is a composite model with terminals and submodels with connections. It is composed of one tank reactor and two parallel tubular reactors. The submodel, `Reactor_1`, is a subclass of the `Tank_Reactor` class, which is also a composite model. In Figure 4.3 the class describing the tank reactor is seen. The submodels `Reactor_21` and `Reactor_22` are models of tubular reactors and are subclasses of the same super-class `Tubular_Reactor`.

We are now down on the process object level. The hierarchy can of course have different number of levels in different areas of the process model. The model class `Tank_Reactor` have an internal structure composed of a vessel model, a recycle pump model and a heat exchanger model.

Summary

We have seen how Omola supports hierarchical decomposition. We have defined a decomposition method called process structure decomposition. It is a hierarchy describing virtual and physical process objects based on the process structure. Composite models can be described by graphics showing the connections between submodels and terminals.

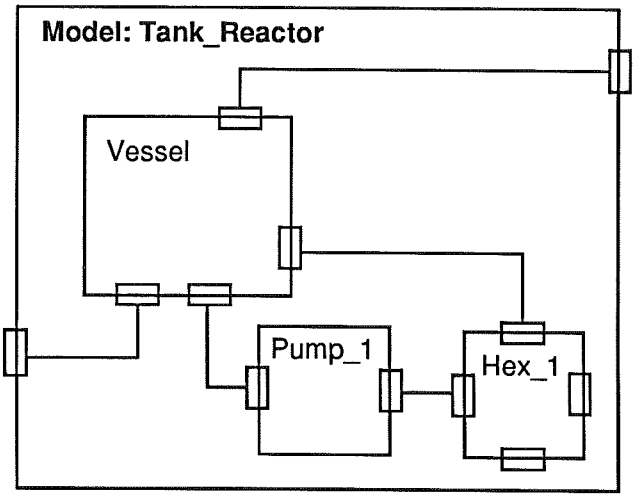


Figure 4.3 The internal structure of `Tank_Reactor`

4.2 Common Process Object Descriptions

In chemical processes some of the equipments are often of identical type. Examples are pumps, valves and heat exchangers. Model objects like these are composed of a number of model components. Before modelling some common process objects we are going to describe common terminal definitions.

Process Terminals

A connection often represents a physical connection of pipes. Structured terminals in process models often are composed of subterminals that describe physical properties of the medium that flow in the pipe. Typical properties are pressure, flow, temperature, chemical composition etc. Sometimes it is not necessary to describe all of these. Other times the flows are even more complex like multi phase flows.

A graphical description of a process terminal is shown in Figure 4.4. This structured terminal is composed of four subterminals. Two of these are simple terminals describing across variables, namely pressure and temperature. Flow is also a simple terminal but it is a through variable with a direction. The last one is a vector terminal describing the chemical composition. The vector elements are simple terminals describing the concentration of a chemical component. The length of the vector is equal to the number of chemical components, which is four in this example. The Omola description of the process terminal is seen below. The process terminal `In_Pipe` is a so called record terminal. The composition

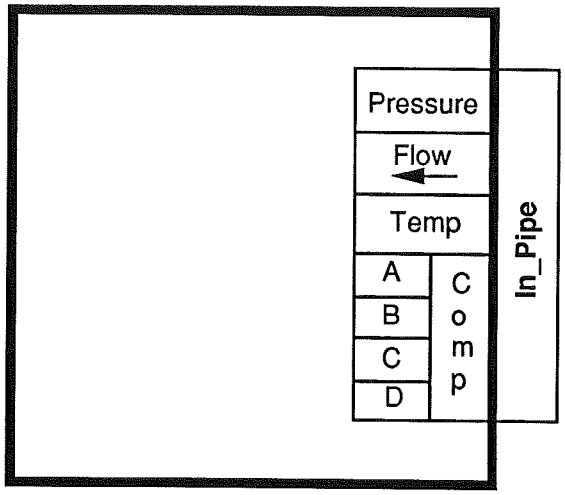


Figure 4.4 An example of a structured terminal used in process models.

subterminal is also defined below.

```
In_Pipe IS A RecordTerminal WITH
  components:
    Pressure IS A Pressure_Terminal
    Flow     IS A Flow_In_Terminal;
    Temp     IS A Temperature_Terminal;
    Comp     IS A Composition_Terminal;
END;

Composition_Terminal IS A VectorTerminal WITH
  Length := 4;
  CompType IS A Concentration_Terminal;
END;
```

The basic elements, the simple terminals, are described as across or through variables as discussed in Chapter 3. An `Out_Pipe` terminal is easy to create. The direction attribute in the flow subterminal is changed to **out**. All these attributes of the simple terminals make it possible to have advanced consistency checks on connections. The SEE system is supposed to check the consistency. The simple terminals that are connected to each other must have the same quantity in order to create a consistent connection. After this automatic scaling is done using the unit attributes. A structured terminal can also contain descriptions of the physical connection, like pipe diameter and pipe construction material etc.

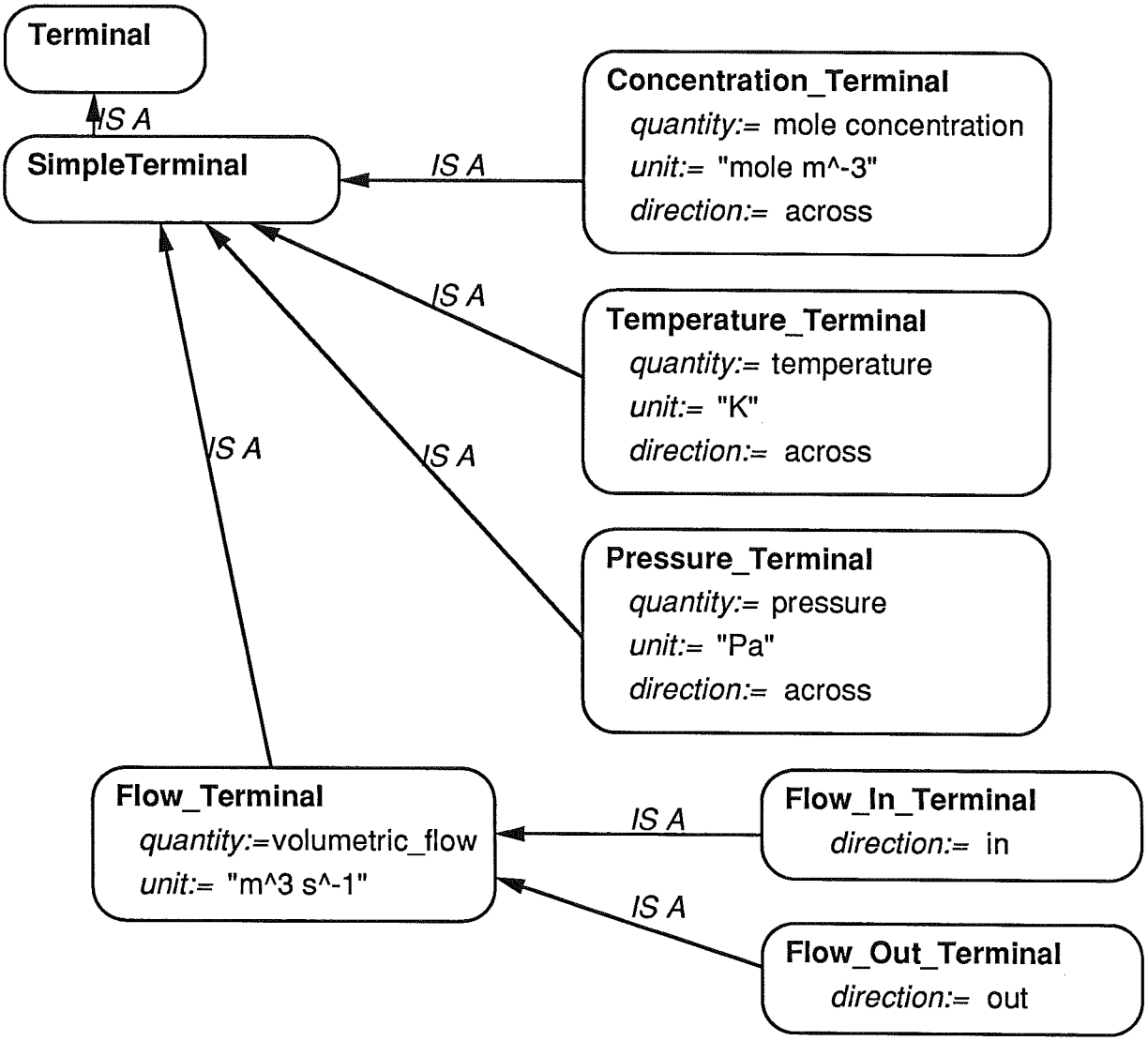


Figure 4.5 The simple terminal class hierarchy in the process model.

A connection between two structured terminals is assumed to be consistent if the subterminals can be connected to each other. If we have two structured terminals that have the same internal structure but should not be connected to each other, then there is not anything that prevent it. For instance it should not be possible to connect a vapour pipe to a liquid pipe. The only solution is to change the internal structure of the terminals. One way to do this is to create a terminal tag that can be used as a terminal type indicator. The model developer have to create this indicator because it is something that is dependent of the model context. A terminal tag is defined below.

```
Terminal_Tag IS A SimpleTerminal WITH
```

```
  value TYPE String;
  variability := constant;
END;
```

The variability attribute of the simple terminal is set to `constant`. If this terminal is connected to another similar terminal their values have to be equal in order to create a consistent connection. This means that we use the value attribute of the terminal tag as an terminal type indicator. The variability attribute is assumed to has the value `time_varying` as default. Below we see two structured terminals describing one liquid and one vapour flow. The terminal tag is used as an indicator of the phase properties of the flow. The terminals are identical accept for the value of this tag. The `In_Pipe` class is specialized to create the `Liquid_In_Pipe`, which has the additional attribute `Phase`.

```
Liquid_In_Pipe IS A In_Pipe WITH
  component:
    Phase IS A Terminal_Tag WITH
      value := "Liquid";
    END;
END;

Vapour_In_Pipe IS A Liquid_In_Pipe WITH
  components:
    Phase := "Vapour";
END;
```

The liquid and vapour terminals can not be connected to each other. They can be used to create a more complex terminal, like a two phase flow pipe.

```
Two_Phase_In_Pipe IS A RecordTerminal WITH
  components:
    Vapour_Phase IS A Vapour_In_Pipe;
    Liquid_Phase IS A Liquid_In_Pipe;
END;
```

Flow Equipments

Flow equipments are objects that are used in the process for transporting the chemical medium through the processing units. In many cases we can assume static behaviour in flow equipment objects. An example is the pressure dynamics which is very fast.

Our models of valves, pumps and pipes assume static behaviour. These objects can be subclasses of a super-class that is described with a static behaviour description. The Omola version can be seen below.

```
Static_Flow_Object IS A Model WITH
  terminals:
    Inlet  IS A Liquid_In_Pipe;
    Outlet IS A Liquid_Out_Pipe;
  realization:
    Statics IS A Primitive WITH
      variable:
        PressureDrop IS A Real;
      equations:
        Outlet.Flow  = Inlet.Flow;
        Outlet.Temp  = Inlet.Temp;
        Outlet.Comp  = Inlet.Comp;
        PressureDrop = Inlet.Pressure - Outlet.Pressure;
    END;
END;
```

A valve model class and a pump model class can now be defined as subclasses of the super-class `Static_Flow_Object`. The only thing that have to be locally defined are expressions for the pressure drops.

```
Static_Valve IS A Static_Flow_Object WITH
  parameters:
    loss_factor, Cross_Area IS A Parameter;
  realization:
    ValveStatics IS A Statics WITH
      equation:
        PressureDrop =
          loss_factor/(2*Cross_Area^2)*
          Outlet.Flow * ABS(Outlet.Flow);
```

```
    END;
END;
```

The pump model class can be defined in a similar way. In the valve the pressure drop is a function of the flow. The flow through the pump is a function of the pressure drop over the pump. In a model with static flow objects, like the pump and the valve, the simulator have to solve an algebraic loop. The SEE-prototype handle equations like this. The Omola model of the centrifugal pump is shown below.

```
Centrifugal_Pump IS A Static_Flow_Object WITH
  parameters:
    pump_factor, Max_Flow IS A Parameter;
  realization:
    PumpStatics IS A Statics WITH
      equation:
        Outlet.Flow = Max_Flow -
        (PressureDrop/pump_factor)^2;
    END;
END;
```

The connections in our composite models are not models of process pipes. They are just representations of relations. If we want to model pipes then we have to make primitive models describing the behaviour of pipes and use them in composite models.

```
Static_Pipe IS A Static_Flow_Object WITH
  parameters:
    Diameter, Length, loss_factor IS A Parameter;
  realization:
    Statics IS A Primitive WITH
      variable:
        PressureDrop, CrossArea, DeadTime TYPE Real;
      equations:
        Outlet.Flow = Inlet.Flow;
        Outlet.Temp = DELAY(Inlet.Temp,DeadTime);
        Outlet.Comp = DELAY(Inlet.Comp,DeadTime);
        PressureDrop = Inlet.Pressure - Outlet.Pressure;
        PressureDrop =
```

```
            loss_factor*Length/(2*Diameter*CrossArea^2)*
            Outlet.Flow*ABS(Outlet.Flow);
        CrossArea = (Diameter/2)^2*PHI;
        DeadTime = Length/(Outlet.Flow/CrossArea);
    END;
END;
```

The pipe behaviour description is static but temperature and composition will be delayed depending on the flow velocity. Notice that the pipe model has a direction depending on the DELAY function, which means that the pipe model must be connected a particular way depending on the application. If the flow direction change then we are in trouble. One solution is to have a DELAY function that change the input signal to an output signal if the dead time is negative. This model of a pipe can not use inheritance of the realization object because it is hard to overwrite existing equations with new ones.

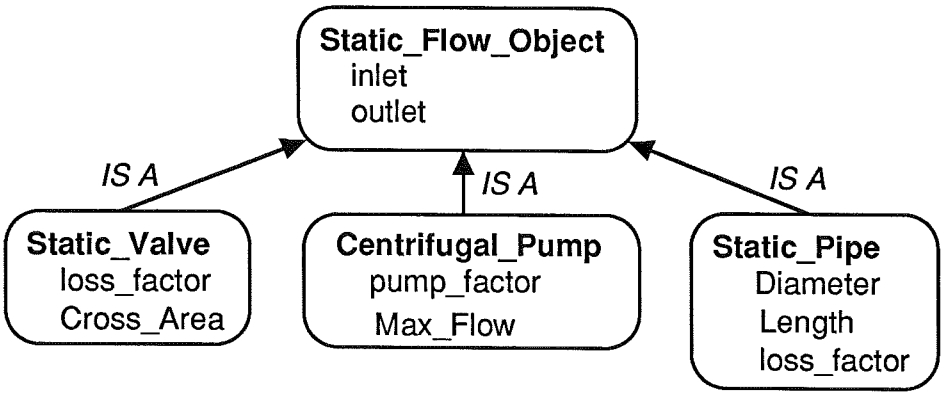


Figure 4.6 The class hierarchy tree of the flow equipment example.

We have seen that inheritance can be used to facilitate model development and it makes it possible to reuse models in a new way. In our last example it had some limitations.

Vessels

Many different kinds of vessels are used in chemical processes. They are used for storage, buffers and processing units. The main characteristics is the dynamic mass balance. The mass in a vessel can increase and decrease due to the operation conditions.

As in the previous section we can create generic super-classes that can be specialized in order to capture different kinds of vessels. In order to create

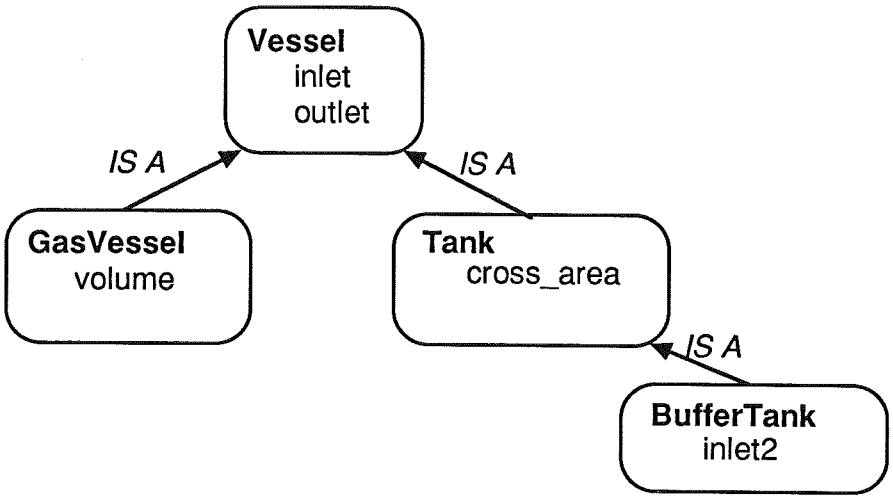


Figure 4.7 The vessel class hierarchy tree.

generic vessels like this we must be able to change the number of chemical components in the objects. The primitive behaviour is expressed in a matrix form. The matrix dimensions can be described by a "number-of-component" parameter. The terminal dimensions must also be changed by this parameter. The composition subterminal is a vector terminal, which can be parameterized in a desired way. We have the possibility to create *generic* vessels that can handle different number of components.

```
Vessel IS A Model WITH
  terminals:
    Inlet  IS A Process_In_Pipe;
    Outlet IS A Process_Out_Pipe;
  parameters:
    No_Comp IS A Parameter WITH value:=2; END;
    Phase   IS A Parameter WITH value TYPE String; END;
  constraints:
    Inlet.Composition.Length :-
      Outlet.Composition.Length :- No_Comp;
    Inlet.Phase :- Outlet.Phase :- Phase;
END;
```

The super-class model `Vessel` contains the description of two terminals and a parameter describing the number of components in the object. The `No_Comp` is a parameter describing the chemical dimension. A constraint is an important model component category for model parameterization. A parameter is bounded to another parameter with a constraint. The

two parameters are not independent any more. Under the constraint tag the vector terminal length are getting their right value, which are equal to No_Comp. Note also that the terminals contains a phase component as type indicator. These indicators are bounded to a string parameter, which can be assigned a value describing the phase of the vessel content. In the gas vessel below the phase parameter is assigned the value "Vapour".

```
GasVessel IS A Vessel WITH
  parameters:
    Volume, GasConst, HeatCap IS A Parameter;
    MoleWeight IS A Row[No_Comp];
    Phase := "Vapour";
  realizations:
    Behaviour IS A Primitive WITH
      variables:
        composition IS A Column[No_Comp];
        pressure, temperature, density, energy IS A Real;
      equations:
        pressure*Volume =
          GasConst*SUM(composition)*temperature;
        Volume*DOT(composition) =
          Inlet.Flow*Inlet.Composition -
          Outlet.Flow*Outlet.Composition;
        density = MoleWeight*composition;
        DOT(energy) =
          Inlet.Flow*density*HeatCap*Inlet.Temperature -
          Outlet.Flow*density*HeatCap*Outlet.Temperature;
        energy = density*Volume*HeatCap*temperature;
        Inlet.Pressure = pressure;
        Outlet.Pressure = pressure;
        Outlet.Temperature = temperature;
        Outlet.Composition = composition;
    END;
END;
```

The GasVessel object is a subclass of the Vessel and is specialized with a realization description. This realization contains the general gas law, a dynamic component mass balance and a dynamic energy balance. Notice

that some of the equations are on vector form. The variable `composition` is a column vector with the length of `No_Comp`. This model can now be used in applications with different number of components and it can be changed by the inherited parameter `No_Comp`.

The Heat Exchanger Model

A heat exchanger (hex) model is described in Section 2.4. The heat exchanger is described by two energy balances and one heat transfer equation. Another way to decompose the heat exchanger is into two separate parts with a heat transfer object in between. This can be called a *transport phenomenon decomposition.* This is discussed in more detail in Section 4.5. In Figure 4.8 the composite heat exchanger model is shown.

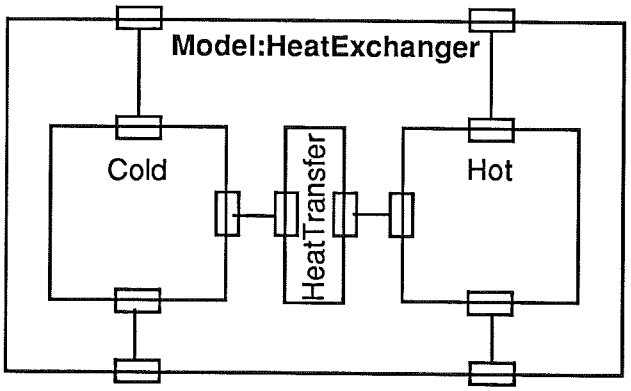


Figure 4.8 The composite model class describing the heat exchanger.

The submodels Cold and Hot are subclasses of the class object `Perfect-Mixing`. It is an ordinary unit with ideal mixing and dynamic energy balance and it is found in Appendix B.4. It is a vessel with constant volume and it has a terminal describing the heat input to the submodel named `Heat_In`, which is a subclass of the `Heat_Transfer_In` terminal class.

```
Heat_Transfer_In IS A RecordTerminal WITH
  components:
    Flux        IS A Heat_Flux_In;
    Temperature IS A Temperature_Terminal;
END;
```

The heat transfer model contains the heat transfer description. In this case it is a static description of the transport phenomenon.

```
Heat_Transfer_Model IS A Model WITH
  terminals:
    Heat_In  IS A Heat_Transfer_In;
    Heat_Out IS A Heat_Transfer_Out;
  parameters:
    HeatConst, HeatArea IS A Parameter;
  realizations:
    Static_Heat_Transfer IS A Primitive WITH
      equations:
        Heat_In.Flux = HeatArea*HeatConst*
          (Heat_Out.Temperature - Heat_In.Temperature);
        Heat_Out.Flux = Heat_In.Flux;
    END;
END;
```

By the transport phenomenon decomposition we have decomposed the primitive model into a composite model. The heat exchanger is composed of submodels that are easily reused. A heat transfer model can be used in many different applications.

Summary

Omola includes a powerful connection and terminal concept. By this we can decouple the connections in the super-model from the terminal description in the submodel. We have also seen the power of object-oriented methodology in describing structured terminals. By using the terminal attributes it is easy to create safe connections with a lot of consistency checks. Connections and consistency checks are discussed in more detail in Mattsson (1989). This makes it easy and safe to reuse submodels in new applications. In this section we have studied primitive models using matrix formalism. The constraint concept is important in order to create advanced parameterization of models. Together with matrix equations it is possible to create generic process objects. This gives a compact description of the behaviour. We made a transport phenomenon decomposition of the heat exchanger. This means that we described the heat transfer phenomenon in a separate submodel which can be reused.

4.3 The Tank Reactor

The Tank_Reactor is a composite model which is composed of a reactor vessel and submodels for energy supply, such as a pump and a heat exchanger. The structure of this model i shown in Figure 4.3. The submodels Pump_1 and Hex_1 are subclasses of Centrifugal_Pump and HeatExchanger, which have been developed in the previous section.

The Reactor Vessel Model

The behaviour of the vessel is described by six equations. By using matrix formalism the description becomes very compact. Note that the subterminal Composition is a vector terminal of length four of simple terminals, which is assigned under the constraint tag.

```
Reactor_Vessel IS A Model WITH
  terminals:
    V_In  IS A Vapour_In;
    V_Out IS A Vapour_Out;
    L_In  IS A Liquid_In;
    L_Out IS A Liquid_Out;
  parameters:
    No_Comp IS A Parameter WITH value:=4; END;
    Volume, Area, Density, K_reac, HeatCap, GasConst,
      Reac_Heat, Pressure IS A Parameter;
    KL          TYPE Matrix[No_Comp,No_Comp];
    comp_equil TYPE Column[No_Comp];
  constraints:
    V_In.Composition.Length :- V_Out.Composition.Length :-
      L_In.Composition.Length :-
      L_Out.Composition.Length :- No_Comp
  realizations:
    Ideal_Tank_Real IS A Primitive WITH
      variables:
        composition, evap_flux TYPE Column[No_Comp];
        temperature, r_velo    TYPE Real;
      equations:
        % Component Mass Balance with Reaction
        Volume*DOT(composition) =
          V_In.Flow*V_In.Composition +
```

```
        evap_flux + Volume*Reac_Velocity;
      % Energy Balance with Reaction Heat
      Density*Volume*HeatCap*DOT(temperature) =
        Density*V_In.Flow*HeatCap*V_In.Temperature -
        Density*V_Out.Flow*HeatCap*V_Out.Temperature +
        Reac_Heat*Volume*r_velo;
      % Evaporation
      evap_flux = Area*KL*(comp_equil - composition);
      % Reaction
      r_velo = K_reac*composition[1]*composition[2];
      Reac_Velocity = [-R_V; -R_V; R_V; R_V; 0];
      % Static Momentum Balance
      V_In.Pressure  = Pressure;
      V_Out.Pressure = Pressure;
      L_In.Pressure  = Pressure;
      L_Out.Pressure = Pressure;
      % Liquid Out Flow
      L_Out.Temperature = temperature;
      L_Out.Composition = composition;
      % Vapour Out Flow
      V_Out.Temperature = temperature;
      V_Out.Composition = evap_flux/V_Out.Flow;
  END;
END;
```

The variable `composition` is a column vector with the length of four. The equation describing the evaporation flux, `evap_flux`, is an example of the powerful matrix notation. The parameter `Area` is a scalar. `KL` is a matrix describing the evaporation coefficients. The driving force for the flux is the difference between equilibrium concentrations and the reactor vessel concentrations.

We see that it is hard to read the equations without knowing what they describe. In the reactor vessel model the equations are explained with comments. There should be a better way of structuring a set of equations like these.

4.4 Distillation Columns

In this section we will discuss the modelling concepts needed in modelling of a distillation column. We concentrate on the parameterization demands on a distillation column model. In the first subsection the medium and machine decomposition problem is approached. In the following subsection a proposal to extend Omola to handle regular structures is presented. After this we discuss the parameterization problem of the column model.

A Tray Model

As mentioned in Section 2.5 a tray model should be medium and machine parameterized. The machine model is a description of the physical properties of the tray. This means that the conditions for mass, energy and momentum phenomena are described by mathematical models. In this case we assume homogeneous mixing which results in differential equations describing the mass and energy dynamics. The medium model contains the description of the medium behaviour that is independent of the machine model and in the model from Chapter 2 it is composed of two vector equations.

The medium and machine decomposition of the tray model opens up a new perspective on dynamic models. We can now develop generic machine models and create media data model libraries. This is perhaps the most interesting application of the ideas in this thesis.

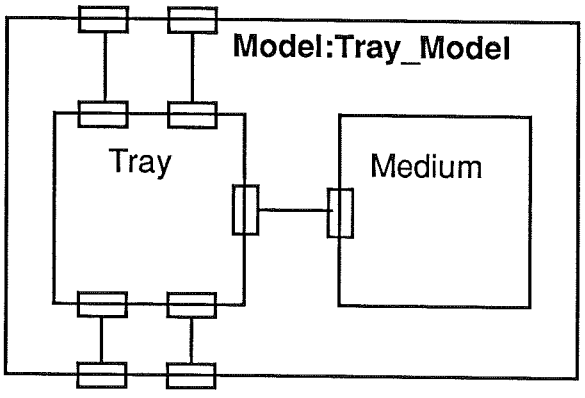


Figure 4.9 The structure of the tray model.

Machine Model: A general distillation element class is created first and it contains only the description of the liquid and vapour terminals.

This is done in order to make it simple to develop the components of the column. It is the super-class for a number of the model classes in this section.

```
Distillation_Generic_Element IS A Model WITH
  terminals:
    LIn  IS A Liquid_In;
    VOut IS A Vapour_Out;
    LOut IS A Liquid_Out;
    VIn  IS A Vapour_In;
  parameter:
    No_Comp IS A Parameter WITH value:=2; END;
  constraints:
    LIn.Comp.Length :- VIn.Comp.Length :-
      LOut.Comp.Length :- VOut.Comp.Length :- No_Comp;
END;
```

This generic distillation element is also the super-class of the machine model for the tray. Once again constraints are used to make nice parameterization of a generic model. The generic tray model (machine model) is described by matrix equations. The equations express the generic behaviour of the tray. The dimensions of matrices and vectors can be set by the parameter No_Comp.

```
Tray_Generic_Model IS A Distillation_Generic_Element WITH
  terminals:
    MData IS A Media_Data_Terminal;
  parameters:
    lmin, Mmin, beta, pressure IS A Parameter;
  constraint:
    No_Comp :- MData.X.Length :- MData.Y.Length :-
      MData.NoComp;
  realizations:
    Tray_Realization IS A Primitive WITH
      variables:
        Mcomp, comp    TYPE Column[No_Comp];
        Mtemp, temp, M TYPE Real;
      equations:
        DOT(Mcomp) =
```

```
        LIn.Flow*LIn.Comp + VIn.Flow*VIn.Comp -
        LOut.Flow*LOut.Comp - VOut.Flow*VOut.Comp;
      DOT(Mtemp) =
        LIn.Flow*LIn.Temp + VIn.Flow*VIn.Temp -
        LOut.Flow*LOut.Temp - VOut.Flow*VOut.Temp;
      M = SUM(Mcomp);
      comp = Mcomp/M;
      temp = Mtemp/M;
      MData.X         = comp;
      MData.Pressure = pressure;
      MData.Temp      = temp;
      LOut.Flow         = lmin + (M - Mmin)/beta;
      LOut.Pressure     = pressure;
      LOut.Composition = comp;
      LOut.Temperature = temp;
      VOut.Flow         = VIn.Flow;
      VOut.Pressure     = pressure;
      VOut.Composition = MData.Y;
      VOut.Temperature = temp;
  END;
END;
```

In the second equation we assume constant density and heat capacitivity, which means that they can be canceled. We see also that the "number-of-component" parameter is assigned through the MData terminal, which means that the chemical dimension is set from the outside.

Medium Data Communication: An important thing to notice is the communication with the medium model. The medium model is discussed below. The operating condition is described in the generic tray model. The medium model describes the vapour composition based on liquid composition, temperature and pressure. The interaction between the machine model and medium model is described by a structured terminal. This has a nice property. The communication has a given structure and there is an unique way to access the medium model from the machine model. The terminal below contains subterminals for number of chemical components, heat data parameters, temperature, pressure, liquid and vapour compositions.

```
Media_Data_Terminal IS A RecordTerminal WITH
```

```
  components:
    No_Comp   IS A Simple_Terminal;
    Cond_Heat IS A Heat_Data_Terminal;
    Evap_Heat IS A Heat_Data_Terminal;
    Temp      IS A Temperature_Terminal;
    Pressure  IS A Pressure_Terminal;
    X, Y      IS A Mass_Composition_Terminal;
END;
```

X and Y are vector terminals describing the mass fraction in the liquid and vapour respectively.

Medium Model: The medium model contains relations that describe the liquid-vapour equilibrium. In this model we use a state equation and Antoine's law as a medium model. Some other medium property parameters are also assigned here. Observe that the parameters are not given any values, except No_Comp. Notice also that No_Comp is assigned the value 5 and through the constraints the Op_Cond terminal gets the right dimension. The value of No_Comp is then propagating through the connection to the machine model, which then gets the same dimension. This makes it possible to create the medium model independently of the machine model and vice versa.

```
Distillation_Media_Model IS A Model WITH
  terminals:
    Op_Cond IS A Media_Data_Terminals;
  parameters:
    No_Comp IS A Parameter WITH value:=5; END;
    Cond_Heat, Evap_Heat IS A Parameter;
    A, B, C              TYPE Column[No_Comp];
  constraint:
    Op_Cond.No_Comp :- Op_Cond.X.Length :-
      Op_Cond.Y.Length :- No_Comp;
  realizations:
    Ideal_Four_Comp IS A Primitive WITH
      variable:
        LogPo TYPE Column[No_Comp];
      equations:
        LogPo = A - B ./ (C + Op_Cond.Temp)';
        Op_Cond.Y =
```

```
            EXP(LogPo)/Op_Cond.Pressure .* Op_Cond.X;
          Op_Cond.Cond_Heat = Cond_Heat;
          Op_Cond.Evap_Heat = Evap_Heat;
    END;
END;
```

Regular Structures

In the original version of Omola there is no mechanism for handling regular structures. Below follows a proposal of how a regular structure mechanism might look like in the Omola language. An advanced use of the matrix formalism is proposed for solving the regular structure problem. By the use of matrices we can define and connect submodels in regular structures. Subclasses of one model super-class can be arranged in a vector or a matrix. Inspired of the powerful matrix notation of MATLAB (Moler et al, 1987) a vector of submodels is defined like this:

```
Tray[1..5] IS A Tray_Model
```

A particular tray model can be reached by the vector notation `Tray[1]`. Connections can of cause be done in a similar way. The matrix notation is used to connect the submodels to each other. In the example below we are connecting the `LIn` of every tray in the vector of tray models to the `LOut` of every tray with a higher vector index.

```
Tray[1..4].LIn AT Tray[2..5].LOut
```

In the column part model below this regular structure mechanism is used to create the structure of a part of a distillation column. This column part is then parameterized by two parameters that describe the index number of the first and the last tray in the vector.

```
Column_Part_Model IS A Distillation_Generic_Element WITH
  parameters:
    bottom := 1;
    top := 5;
  realizations:
    Column_Regular_Structure IS A Structure WITH
      submodels:
        Tray[bottom..top] IS A Tray_Model;
      connections:
        LIn                          AT Tray[top].LIn;
```

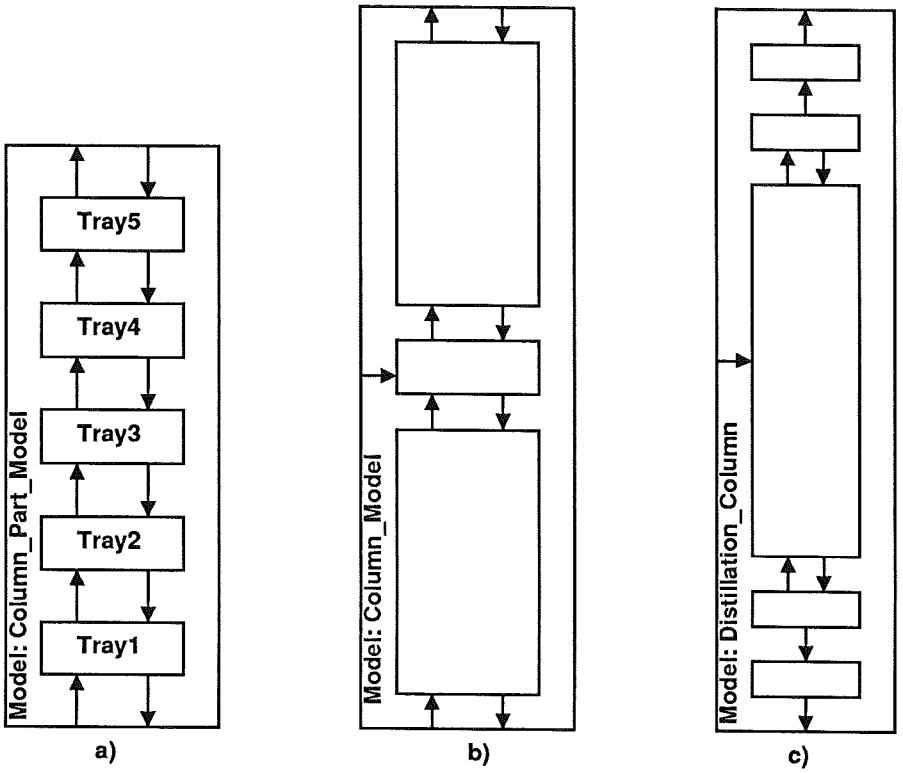


Figure 4.10 **a)** The regular structure model `Column_Part_Model`. **b)** The column model. **c)** A distillation unit with partial condenser.

```
        Tray[bottom..top-1].LIn AT Tray[bottom+1..top].LOut;
        Tray[bottom].LOut       AT LOut;
        VIn                     AT Tray[bottom].VIn;
        Tray[bottom+1..top].VIn AT Tray[bottom..top-1].VOut;
        Tray[top].VOut          AT VOut;
    END;
END;
```

The regular structure mechanism is very useful and makes the creation of structures easy. In a graphical based front-end there is some trouble. Composite models is described by graphics. But a regular structure have to create the picture of the structure by itself. The user must help the system with some additional commands for describing the picture. For instance if the structure is horizontal or vertical and in which end the structure index begins. An example of a vertical regular structure of the type `Column_Part_Model` is seen in Figure 4.10a.

A Column Model

We have developed a tray model based on medium and machine decomposition. We have also discussed a mechanism for regular structures. The actual column model can now be defined as being composed of two column regular structures, one rectifier and one stripper. Between these two we also need a feed tray model, which is a specialization of the tray model. A column model is seen in Figure 4.10b.

When the column model is defined we can now define different kinds of distillation units. In the process we have two kinds of units, one column with a total condenser and two columns with partial condensers. All three have the same reboiler configuration. The two kinds of units are specializations of a basic structure super-class. In Figure 4.10c a distillation column model with partial condenser is seen on the right.

Parameterization of the Column

A distillation column can be parameterized in a number of ways. We need parameters for changing structures and submodels in structures. We also discussed the need of assigning parameters through the super model. Another way of parameterizing is to propagate parameters in the connections of submodels.

Structure Parameterization: This can be handled quite easily by using the object-oriented modelling concepts in a proper way. First create a model class that have parameters that describe the structure and that have submodels definitions. A change of these parameters can be done by making a specialization of this model class into a subclass with overwriting of the old parameter values, submodel names or name on submodel classes. In other words, we can use inheritance to make this kind of parameterization. Below follows an example of structure parameterization. The super-class is a description of a distillation column with a partial condenser configuration. It is a subclass of a distillation column with a description of the reboiler configuration.

```
Distillation_Column_With_PC IS A Distillation_Column_BS WITH
  terminals:
    Top IS A Vapour_Out;
  parameters:
    No_Comp       := 2;
```

```
    No_trays     := 10;
    Feed_tray_at := 6;
  constraint:
    Top.Comp.Length :- No_Comp;
  realizations:
    Column_Structure_PC IS A Column_Structure WITH
      submodels:
        Condenser IS A Partial_Condenser;
        Top_Valve IS A Control_Valve_D;
      constraints:
        Condenser.No_Comp :- Top_Valve.No_Comp :- No_Comp;
      connections:
        Column.VOut     AT Condenser.VIn;
        Condenser.VOut  AT Top_Valve.Inlet;
        Condenser.LOut  AT Column.LIn;
        Top             AT Top_Valve.Outlet;
    END;
END;
```

A distillation unit can now be described as a subclass `Column_12`. This unit uses inheritance in order to change structure parameters. like number of components and number of trays. We have also changed the model class of the condenser.

```
Column_12 IS A Distillation_Column_With_PC WITH
  parameters:
    No_Comp      := 5;
    No_trays     := 5;
    Feed_tray_at := 4;
  realizations:
    Column_Struc IS A Column_Structure_PC WITH
      submodel:
        Condenser IS A Partial_Condenser_2;
    END;
END;
```

Super-Model Parameter Assignment: In Omola This can be done directly through constraints. We see in the example above that the "number-of-component" parameter `No_Comp` is a super-model parame-

ter. It is used to set a number of other parameters in the submodels. This is done under the `constraint` tag in the realization description. It can also set parameters in model components like terminals.

Parameter Propagation: Propagation is used to set parameters in the tray machine model by parameters in the distillation medium model. In the medium model we define the number of chemical components and their physical behaviour. This parameter describes the dimension of the vector equations in the medium and machine models. The `No_Comp` parameter is propagated through the medium data terminal from the medium model to the machine model. In the machine the dimension of the vector equations and the composition vector terminals is assigned by the propagated parameter.

Summary

Model objects that are complex and often used must be parameterized in a way that make them reusable. In this section we have discussed decomposition and parameterization of a distillation column model. Distillation column can have different structures, which can be parameterized in a convenient way. Columns can be used for different media by only changing the medium description. The tray unit can also be changed independently of the medium and column models. Decomposition, parameterization and object-oriented modelling make it possible to specialize the distillation column model in different directions independently. This is an important result.

4.5 Modelling the Tubular Reactor

In the first subsection we look at decomposition of the tubular reactor and in the next at regular structures and parameterization. Model reuse and model refinement are also discussed in one subsection each.

Model Decomposition

The tubular reactor is modeled by partial differential equations. In the approximated model in Section 2.5 we discretized the model in space (one dimensional discretization). This gives us a new model with a number of submodels in series. This means that the reactor is sliced horizontally. Each slice is described by a number of differential equations. How

many slices to chose in the discretized model is just a matter of model approximation. This method of decomposition we therefore call *model approximation decomposition.* Submodels are connected in a structure in order to approximate a model behaviour. Transport phenomenon decomposition is also used in modelling the horizontal slice of the tubular reactor. In this example we decompose the heat interaction between the tube and the surrounding feed stream. The tube can be medium and machine decomposed into one perfect mixing machine model and one chemical reaction model. The different decompositions are clearly seen in Figure 4.11.

Regular Structures and Parameterization

The tubular reactor model is composed of a regular structure model. This model is a generic regular structure model, which could be used as a super-class in the distillation column model example. The reactor is discretized in a number of slices which can be changed by a parameter. Observe that if the number of slices changes, the constraint relations also change the relevant parameters in the submodels. These are tube length and different geometric data that are assigned on the tubular reactor level. The submodel parameters are then assigned the right value through super-model parameter assignment, where some of the submodel parameters must be calculated.

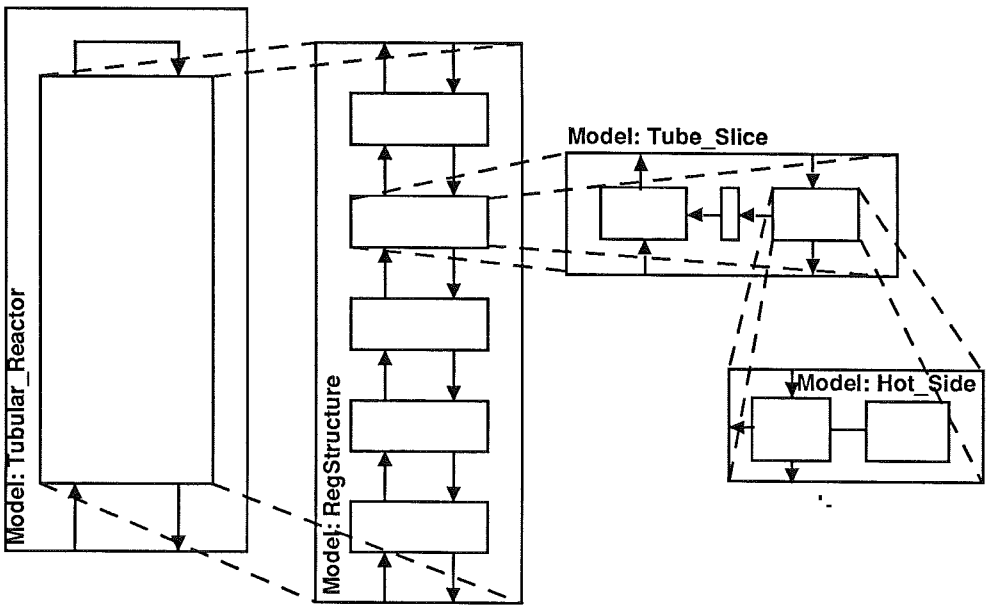


Figure 4.11 The decomposition of the tubular reactor.

```
Tubular_Reactor IS A Model WITH
  terminals:
    Inlet  IS A Vapour_In_Pipe;
    Outlet IS A Vapour_Out_Pipe;
  parameters:
    Tube_Length, Total_Heat_Area, Total_Reac_Surface,
      Cold_Cross_Area, Hot_Cross_Area IS A Parameter;
  realizations:
    ReactorRealization IS A Structure WITH
      parameters:
        No_slices IS A Parameter WITH value:= 10; END;
      submodels:
        Reactor_Structure IS A RegStructure1N WITH
          Element_Model IS A Tube_Slice;
        END;
      constraints:
        Reactor_Structure.n :- No_Slices;
        Reactor_Structure.Element[1..n].Length :-
          Tube_Length/No_Slices;
        Reactor_Structure.Element[1..n].Heat_Surface :-
          Total_Heat_Area/No_Slices;
        Reactor_Structure.Element[1..n].Reac_Surface :-
          Total_Reac_Surface/No_Slices;
        Reactor_Structure.Element[1..n].Area_Cold :-
          Cold_Cross_Area;
        Reactor_Structure.Element[1..n].Area_Hot :-
          Hot_Cross_Area;
      connections:
        Inlet                          AT
          Reactor_Structure.InBottom;
        Reactor_Structure.OutTop       AT
          Reactor_Structure.InTop;
        Reactor_Structure.OutBottom AT Outlet;
    END;
END;
```

The tube is only described with tube elements in series. The tube can of course be described as a matrix of tube elements. If we discretize the

tube along the vertical axis and on the same time discretize it along the tube radius we get a matrix of elements.

Model Reuse

We see that the structure from the column model appears again. There is no difference between the two and we can reuse the regular structure model. We can also reuse some parts of the heat exchanger model because it was transport phenomena decomposed. The tube slice model is described as a composite model with heat interaction between the two sides of the tube wall and is therefore a subclass of the heat exchanger model. All the different levels of decompositions can be reused.

The cold side is a gas vessel description with perfect mixing and heat interaction. The cold side model is not a subclass of the gas vessel model class defined in Section 4.2. The gas vessel model does not contain any heat interaction with the surroundings. It is difficult to specialize realization objects, as discussed in Section 4.2, and this is an example where we can not use inheritance. This is one of the major limitations of the inheritance concept.

The hot side or tube side is similar to the cold side but must also describe the chemical reaction. The tube side is therefore decomposed into two submodels. One submodel is a gas vessel description with perfect mixing, constant volume and heat interaction. The other one is a description of the gas–solid chemical reaction. This decomposition is similar to the medium and machine decomposition of the distillation tray model.

Model Refinement

To adapt a model to new applications or to increase the model accuracy we refine the model. The model is changed in order to fit the new demands. On the same time we do not want to throw the old model away. This means that we need multiple behaviour descriptions of a model. Omola handles this.
The tubular reactor model is decomposed into a regular structure of slices. The choice of the number of slices is a tuning parameter. This means that it is easy to have a number of realizations with different numbers of slices. Examples of realizations are one simple behaviour descriptions for elementary dynamic studies, one more complex realization for control system design. This is seen in the example below. Multi-

ple realizations and inheritance are the fundamentals in model reuse and model refinement.

```
Tubular_Reactor_2 IS A Tubular_Reactor WITH
  primary_realization := SimpleRealization3;
  realizations:
    SimpleRealization3 IS A ReactorRealization WITH
      parameter:
        No_Slices := 3;
    END;
    ComplexRealization35 IS A ReactorRealization WITH
      parameter:
        No_Slices := 35;
    END;
END;
```

To summarize the modelling of the tubular chemical reactor we needed to decompose the model with model approximation decomposition, transport phenomenon decomposition and medium and machine decomposition. We also saw that a lot of models can be reused from models discussed earlier We also parameterized the model in a convenient way for the user to select a proper model discretization. Multiple realizations allow the user to add new realizations to the model. This makes it flexible an the user can refine the models as the demands on the model changes.

4.6 Summary

In this chapter we have used Omola to describe the chemical process model from Chapter 2. The full implementation is found in Appendix B. Extensions of Omola is needed to capture regular structures. A suggestion of a mechanism is done based on matrix formalism.

The object-oriented approach with the inheritance concept facilitate development and reuse of models and model components. To further facilitate reuse a careful model decomposition and parameterization are needed. A proper decomposition of models makes it easy to develop and reuse previously defined models and model components. We have seen that it is possible to make model decomposition in four ways.

- *Process structure decomposition* in a hierarchical submodel description.
- If a primitive model contains a description of momentum, heat or mass transfer we can make a *transport phenomenon decomposition.*
- A more advanced way of decomposition is the *medium and machine decomposition.* In this chapter we have seen that it is possible to make this factorization of a primitive model.
- *Model approximation decomposition* is used to decompose models with complex structure or complex behaviour.

Parameterization is another important method to increase the reusability of models.

- *Structure parameterization* is used in composite models for parameterization of the structures, submodel types and model and model component dimensions.
- In hierarchical submodel descriptions it is important to be able to do *super-model parameter assignment.* This makes it possible for the user to set design parameters on a convenient abstraction level.
- *Parameter propagation* is used to assign parameters through connections.

5. The Potential of Object-Oriented Modelling

In Chapter 4 object-oriented modelling of a chemical process is discussed. A number of ideas and concepts are presented and analyzed. In this chapter the object-oriented approach is discussed in a wider perspective. A comparison, result and methodology discussions and future development remarks are the subject of this chapter.

The chapter begins with a short comparison between Speedup, G2 and SEE. In the second section the main results from the object-oriented approach to modelling are discussed. In Section 5.3 this discussion continues in a brief methodology description and in Section 5.4 future development of the SEE model representation is suggested.

5.1 A Comparison

In this section we will make a brief discussion and comparison between the object-oriented modelling methodology discussed in this work and two commercial modelling systems, Speedup and G2. Speedup is a simulation package for simulation of dynamic behaviour in chemical processes (Perkins, 1986). G2 is a real-time expert system with a simulator for dynamic models (Moore et al, 1987, or G2, 1988). Both are state of the art in their fields.

Speedup

Speedup is developed at the department of Chemical Engineering at Imperial Collage in London (Perkins, 1986). The first versions were developed in the sixties and Speedup is based on Fortran. The Speedup model representation concepts include a number of the model structuring concepts discussed in Chapter 2. Below follows the most important ones.

- A process model can be described in a `FLOWSHEET` which is modularized into `UNITS`. The flow sheet is a kind of composite model and

the units represents submodels, which means that Speedup has only one level of abstraction.

- A unit (submodel) is described in a `MODEL` definition. Two similar units can share the same model definition. Consequently Speedup has a model type concept for development of common process submodels.
- Speedup has also a concept similar to terminal called `STREAM`. Input and output variables of a unit are structured in streams. On the flow sheet level the streams are connected to each other through connections.
- Speedup handles differential and algebraic equations and has good integration routines for simulations.

Speedup has been developed at a chemical engineering department, which is a reason why Speedup has relative good model structuring concepts. Speedup is probably the best system today for development and simulation of chemical process models. The technology behind Speedup is from the sixties and seventies, which makes the potential of Speedup limited.

G2

The G2 real-time expert system is developed at Gensym in Cambridge, Massachusetts. It includes a simulator of dynamic behaviour. G2 is based on an object-oriented representation and has a powerful graphical user interface. G2 is written in Lisp.

- A process model can be represented as a set of objects connected to each other. An object can have other objects as attributes. Connections are also represented as objects.
- An object is an instantiation of a class definition. G2 has single inheritance, which means that subclasses inherit attributes from their super-classes.
- G2 has a nice graphical interface. Objects are represented as icons and connections are made interactively by drawing a connection between two objects.
- The equations that describe the model behaviour are not structured. All equations and variables are global. It is up to the model developer to structure the descriptions of the primitive models. This makes it impossible to make consistency checks and it is possible for the user to create equations that contradict the interpretation

of the connections. On the other hand it is possible to have generic equations describing the behaviour of a super-class. These generic equations are then used in simulations of subclasses and instances.

- G2 contains an equation editor with a syntax help support. When the user types a word in the editor, it directly makes a consistency check and makes suggestions for possible words that can be used.

The simulator is the weakest part in the G2 system. It simulates one equation at the time and uses only simple integration routines. G2 has an object-oriented model representation and a nice model/user interface but does not support model structuring concepts in a wider extent. G2 is based on the latest in computer science and has therefore a great potential for future development.

A Comparison

Speedup has many of the model structuring concepts discussed in Chapter 2, but it lack hierarchical submodel description and inheritance compared with SEE. These concepts are important in model abstraction, development and reuse.

G2 has a well developed object-oriented model representation, but it lacks almost every model structuring concepts discussed in Chapter 2. An advanced user can create an environment in G2 that has some of the concepts, but it is not a part of the system. Examples of concepts that can be created are hierarchical submodel descriptions and terminals.

Speedup includes a number of the SEE model structuring concepts and G2 has the object-oriented approach. Without any doubt SEE model representation is superiour to both Speedup and G2. To make SEE a good modelling environment it requires a lot more than a superiour model representation. It requires a good model/user interface and a number of different tools.

5.2 Benefits of Object-Oriented Modelling

The main benefits of object-oriented process modelling are facilitated model development, model reuse, model refinement and model maintenance.

Model Development

The possibility to develop new models are important because a model library can never contain every possible model. Modelling is an engineering area dominated by individual taste and conventions.

Hierarchical submodel decomposition is important when modelling complex processes. Large process models require a number of different levels in order to have a convenient model description. Each level should have the right amount of abstraction. It is easier to develop and validate small submodels than large ones. The decomposition of models into model components makes it easier to develop new models by reusing components, like terminals from terminal libraries.

Inheritance is the other concept that facilitates model development. Specialization of predefined models and model components in order to fit a new application is convenient for the model developer. To use predefined models as examples in development of new models is one way to make fast prototyping, which decreases the modelling efforts. This can be done by inheritance of the properties of a super-class or by changing a copy of a class.

Model Reuse

Reuse of models and model components are of main interest because it means that the user does not have to invent and to develop old and known models.

Modularization into submodels and hierarchical submodel descriptions decompose large models into smaller submodels. This makes it easier to reuse parts of models. Also the decomposition of models into separate model components facilitates the reuse of predefined model components. The decomposition methods, discussed in Chapter 4, can help the user to make model decomposition in a way that increases the reusability of models.

Descriptions of common process objects can be reused as submodels in new models and this is done through inheritance. The powerful inheritance concept makes it possible to specialize and change old models into new similar models. Reuse and specialization of model components make process modelling easier.

Reusability of models is increased through various ways of parameterization as structure parameterization and parameter propagation through

connections. Some of the parameterization methods presented in Chapter 4 is based on the inheritance concept.

Model Refinement

Models need to be refined when they are tuned and validated to real data, when they are used in new applications or when they are used for new tasks. If a model of a particular physical object does not contain a description that satisfies the new demand, then it must be possible to refine it without changing the old model description. In order to refine predefined models one can use inheritance and multiple realizations.

One way to refine a model is to specialize it, change the description or to change parameters. This can be done through specialization of a predefined model. We then get a new model with modified properties.

Another way is to create new behaviour descriptions in the predefined model. The multiple realization concept makes this possible and it means that we keep the old model and just add a new behaviour description. There is now an opportunity to switch between different behaviours instead of changing between different model objects.

Model Maintenance

In industry processes are continuously changing. This means that process models must be changed all the time and adjusted to predict the process. Models with long lifetime must be maintained and used by different persons. This maintenance problem is composed of demands on model readability, model reuse and model refinement.

The model must be easy to understand and to read. The hierarchical submodel description makes it possible to have the right amount of abstraction on each level to increase readability. The model representation described in this work lacks a good concept for verbal or graphical model presentations.

Model reuse and model refinement are discussed above. Inheritance is a concept that facilitates model maintenance. Correction of a model can be done in the class description, which means that the correction is inherited by its subclasses.

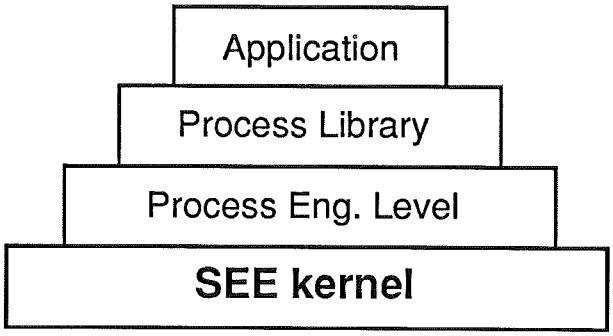


Figure 5.1 The process engineering environment with different layers on top of the SEE kernel.

5.3 A Process Modelling Environment

Modelling and simulation are gaining interest in industry. Simulation is used in many different tasks and used by different user categories. In order to spread the use of modelling and simulation the system must be customed for different users. The *library developer* develops generic models that can be reused in different applications. The *process model developer* uses predefined models, from libraries, to create process models. These process models are used by the *end user* category. To customed a process modelling environment we have to create specialized layers on top of the SEE kernel. One example of a layered design is seen in Figure 5.1. The first layer is the SEE kernel discussed in Chapter 3. The second layer contains special designed tools and mechanisms for process engineering. The third layer contains a set of process model libraries and the fourth layer contains the application and end user interfaces.

User Categories

Different user categories as the library developers, the process model developers and the end users and their need of tools are discussed below.

Library Developer: Model libraries containing generic and reusable models that can be used in different applications are developed by a library developer. The library developer uses inheritance, decomposition and parameterization methods, as discussed in Chapter 4, for creating model libraries. For this the developer requires a number of editors. A *graphical editor* for composite model development and a *primitive model editor*, which may be an ordinary text editor. These kind of editors can be specialized for process applications and this is discussed later in this section. Specially designed *editors for terminals* are also interesting

because these objects are small and have a given internal structure. The library developer must also have ways to investigate if there is a similar model that can be modified and reused.

Process Model Developer: The process model developer category uses some of the basic mechanisms in the SEE kernel and the models in the model library. The process model developer must therefore know about inheritance, decomposition and how to use parameterization. This user makes minor adaptations of library models but the major work is to configure process models for different simulation tasks. To do this the model developer needs advanced ways to find the best model in large libraries.

End User: The end user category should not be forced to know anything about the SEE system at all. Examples of how different end users are using the system are process operator for training, management personnal for production planning and other engineers for process performance studies. The end users want to interact with the simulator in a for them convenient and natural way. This demand requires a way to define interfaces that can be customed for particular applications and for different end user categories. Interactions with and presentations of the model should be described in a high level language.

Process Engineering Tools

In an object-oriented modelling environment the most important part is the model library. The development and reuse of models is based on predefined objects. The user must be able to search through large libraries in order to find an object that satisfies the demands. The developer requires sophisticated support during development of new models in order to increase model development speed. To make end user interfaces the process modelling environment must contain a way to define interaction and presentation of models.

Library Browser: A model library may contain a large number of different objects. It should be possible to structure large model libraries in order to increase the accessibility, readability and comprehensibility of the libraries. Fast search through large amount of decentralized information can be done with a *browser*, similar to the one in Smalltalk (Smalltalk/V, 1986, or Kaehler and Patterson, 1986). It should be possible to show detailed information of an object and on the same time

show an overview of the model data base. To find a desired object in the library is not easy, even with a browser. *Multiple presentations* of the objects are one way to increase the readability and understanding of the object. The information that presents the object, in the browser, can be of different kinds. Examples of different presentations are the Omola code, mathematical and physical descriptions of the model, short and detailed verbal presentations and different kinds of graphical representations. *Automatic search* can also be used to increase the speed for finding a desired model in a large library. One proposal of a search method is to have routines that find all objects that contain a particular keyword. Each object is characterized with a number of keywords, which the browser can search for.

Model Editors: We have already mentioned a number of different editors. A *composite model editor* based on a graphic interaction is a block diagram editor in its simplest form. Mechanisms for regular structure description is of course one important extension. Another extension is to allow different kinds of object representations (icons) in the composite model editor. This means that it is possible to create, for the end user, natural descriptions of the process. An extension like this requires yet another editor, namely an *icon editor.* Primitive models are developed in a *text editor.* A primitive model editor may help the developer with generation of model equations. The developer should be able to describe the model behaviour in a high level language like specialization of different balance equations. Specifications like dynamic mass and energy balance can be used by the editor in order to automatically generate the equations for the model.

End User Interface: A good modelling environment requires a well designed end user and model interface. It should be possible to generate specially designed interfaces for simulation of a model for different end users. It must therefore contain ways for defining interactions with and presentations of models. This requires mechanisms for multiple presentations, as discussed in the model library section above. It also requires methods for control of the end user interaction with the model, like definitions of push buttons and specially designed menus.

5.4 Development of the Model Representation

In the previous chapter there were some discussions about limitations in the SEE model representation. Some presentations of possible extensions were done. In this sections these are discussed in more detail.

Regular Structures

Regular structures are common in chemical process applications. In the previous chapter there were a suggestion of what the regular structure mechanism may look like. In process engineering multi dimensional partial differential descriptions are quite common. Discretization of a model like this ends up in a multi dimensional regular structure. An example is seen below.

```
Element[1..3;1..2;1..2] IS A Element_Model;
```

This represents a three dimensional structure of twelve elements with three in the x-direction, two in the y-direction and two in the z-direction. Connections of regular structures can be made in the same matrix notation spirit.

```
Element[1..2;1..2;1..2].OutX AT Element[2..3;1..2;1..2].InX;
Element[1..3;1;1..2].OutY    AT Element[1..3;2;1..2].InY;
Element[1..3;1..2;1].OutZ    AT Element[1..3;1..2;2].InZ;
```

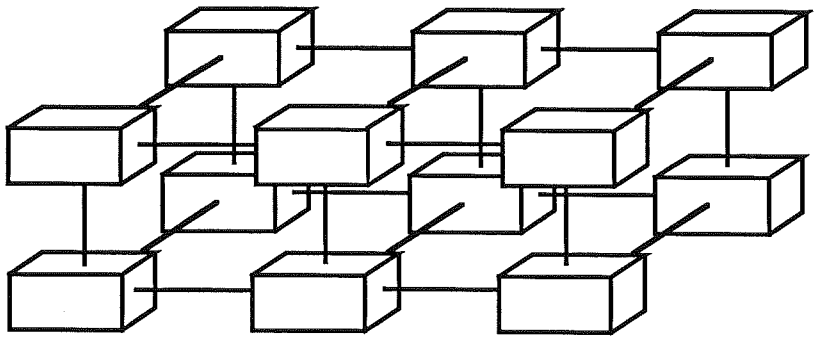


Figure 5.2 The tree dimensional example of a regular structure

In this example the elements are connected to each other in a net structure as can be seen in Figure 5.2. The need of this concept has already been discussed in a great detail in Chapter 4.

Deletion of Inherited Attributes

The possibility to deletion of inherited attributes will facilitate future reuse. In some cases it is convenient to reuse a predefined model by specialization, but the model may include some undesirable attributes. Inherited attributes can be overwritten by local attributes. If this local attribute gets the value of nil then it is neutralized.

```
Model1 IS A Model WITH
  terminal: Input IS A Terminal;
END;

Model2 IS A Model1 WITH
  terminal:
    Input NIL;
    Output IS A Terminal;
END;
```

In this example the inherited attribute is deleted in `Model2`. Inheritance of complex model components, like realizations, will sometimes contain undesired attributes that we want to delete. The problem is that connections and equations do not have any name, so it is impossible to delete them independently of other desired attributes.

Multiple inheritance

Multiple inheritance allows inheritance of attributes from more then one super-class. Some object-oriented programming systems have multiple inheritance. It is interesting to know how multiple inheritance can influence object-oriented modelling of the process model in Chapter 4. I did not find any good example where multiple inheritance could be used in a convenient and successful way.

Discrete Events

Discrete events are not handled in the SEE-simulator. To simulate batch processes or sequence control we need an event concept. Discrete event extensions have been described by Andersson (1989c). The syntax of an event concept is shown below:

```
WHEN condition action
```

This concept have great potential in process simulations because it can be used in some spectacular ways. In the first example below, when the inlet flow is larger then a maximum value, an alarm is caused. The second event description causes a change of realization, when the temperature is higher then a given value.

```
WHEN Input.Flow > FlowMax CAUSE alarm1

WHEN Temp > HighTemp DO
  HighTempModel=true;
  LowTempModel=false;
END;
```

Structuring of Behaviour Descriptions

Realization tags can be used to give names to nameless attributes, like equations and connections. It was mentioned that it is difficult to specialize realization objects. It is impossible to change inherited nameless attributes. This means that the model developer only can add new attributes and not overwrite or delete. Tags can be used to give names to attributes or groups of attributes.

```
Tank1 IS A Model WITH
  terminals:
    In IS A Flow_In_Terminal;
    Out IS A Flow_Out_Terminal;
  parameters:
    density, Cp IS A Parameter;
  realizations:
    Real1 IS A Primitive WITH
      variables:
        mass, energy TYPE Real;
      equations:
        mass_balance:
          DOT(mass) = density*(In.Flow - Out.Flow);
        energy_balance:
          DOT(energy) = density*Cp*(In.Flow*In.Temp -
            Out.Flow*Out.Temp);
```

```
            Out.Temp = energy/(mass*Cp);
    END;
END;
```

Another way to solve this problem is to give names to the realization subobjects. In that case inheritance can be used to simplify changes in realizations. This makes it possible to change one equation without rewriting the others.

```
Tank1 IS A Model WITH
  terminals:
    In IS A Flow_In_Terminal;
    Out IS A Flow_Out_Terminal;
  parameters:
    density, Cp IS A Parameter;
  realizations:
    Real1 IS A Primitive WITH
      variables:
        mass, energy TYPE Real;
      equations:
        MassBalance IS A Equation WITH
          DOT(mass) = density*(In.Flow - Out.Flow);
        END;
        EnergyBalance IS A Equation WITH
          DOT(energy) = density*Cp*(In.Flow*In.Temp -
            Out.Flow*Out.Temp);
          Out.Temp = energy/(mass*Cp);
        END;
    END;
END;
```

Attribute Validity Range

When using models form libraries it is of great interest to prevent the user from using the model in a wrong way. The strong connection consistency check is one step in this direction. Another is to have validity ranges on variables and parameter. If a parameter is set outside the validity range or a variable during simulation is outside the validity range the user gets a warning. The density parameter below can only be set by

values between 500 and 5000.

```
Density IS A Parameter WITH
  value := 998;
  range := [500 5000];
  unit  := "kg m^-3"
END;
```

Model Hierarchy Modifications

When working with composite models it is sometimes interesting to change the hierarchical submodel decomposition, both to increase the number of levels and to decrease it. To increase the number of hierarchical levels can be interesting when one composite model becomes to complex or one part of the composite model is of general interest. Then it should be possible to cluster this part into a new submodel. The opposite is to decrease the number of levels through removing the submodel wall and incorporate the submodel structure into the super-model structure.

An example of this is seen in Figure 5.3. If the reaction part model was developed as in Chapter 4 it could be convenient to create a new model hierarchy level by clustering the parallel reactors in a new model.

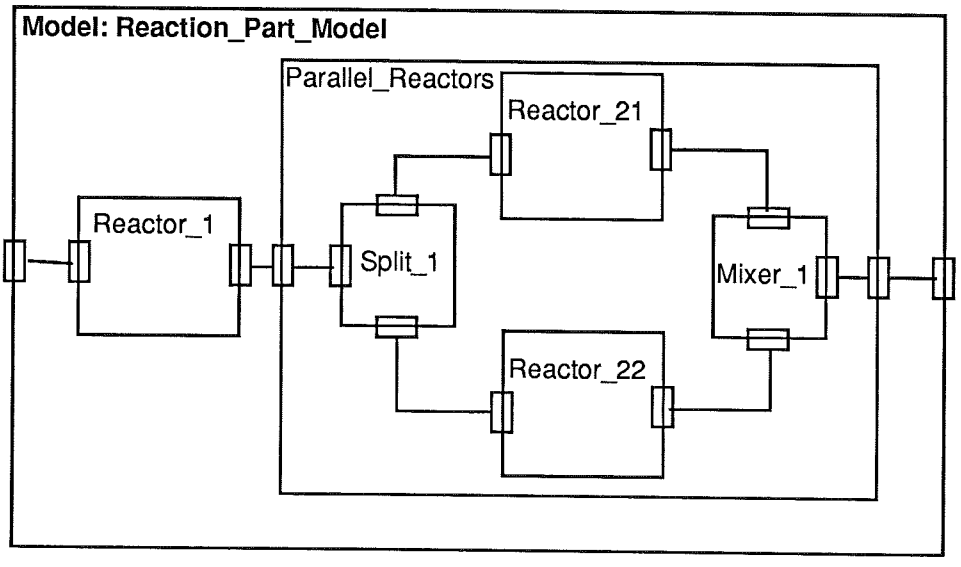


Figure 5.3 An example with modification of the model hierarchy.

Multiple Presentations

In this thesis we have only concentrated on process models. The process control system can of course also be described in this model. But a control system description will destroy the process model structure. If the control system is super imposed on the process structure then it will be process-oriented. If we want to have a control-oriented description then we have to create a new model representation. On the same time these descriptions are describing the same process. They are only different presentations of the same representation. In the SEE prototype there is no way to make two different presentations based on the same representation.

Multiple presentations can also be important in model libraries. To increase the model development speed we need mechanisms that make it easy for the user to find a particular model. To have a number of different presentations models will increase the readability and understanding of the presented library models.

To allow concepts for multiple presentations are not a contradiction with other concepts and are probably not difficult to include in the SEE system.

6. Conclusions

In this thesis an object-oriented approach to chemical process modelling was explored. Structured modelling of chemical processes requires a number of structuring concepts. In Chapter 2 these concepts was deduced out of examples. The object-oriented methodology was presented in Chapter 3. We also saw how the SEE model representation and Omola describe dynamic models. Experiences of a case study were discussed in Chapter 4. It did not only show how to describe process models in Omola but also some decomposition and parameterization methods. A more general discussion of the potential in object-oriented modelling was done in the previous chapter, Chapter 5. The major benefits of an object-oriented modelling approach is facilitated development, reuse, refinement and maintenance of models.

Contributions

The major contribution of this thesis is the case study, where the object-oriented modelling methodology is used successfully. Object-oriented modelling in general and the SEE representation in particular are good tools for describing a continuous chemical process. The process in the case study contains a wide range of different model types and model structures, which means that methods used in the object-oriented approach can be used as a modelling methodology for similar models.

Some *decomposition methods* for process model decompositions are presented. Decomposition is important in object-oriented modelling, because it increases reusability of models. The most important method is the medium and machine decomposition. It makes it possible to describe the medium and the machine independently, which is of particular interest in process applications.

Parameterization is also important in the aim of increased reusability of models. In an object-oriented methodology parameterization can be done in many different ways. Structure parameterization is perhaps the most important.

A vision of a *process modelling environment* was discussed and some ideas was presented. *Extensions* of the SEE model representations have

also been suggested and discussed. In process applications the regular structure mechanism is the most important.

Suggestions for Future Work

This thesis may be continued in several different directions. This work has been concentrated on the model representation needed for a chemical process model.

A "real" case study on a running SEE-system with a "real" chemical process is one natural continuation of the work. The requirements are then to create a running simulator and model representation that is convenient for a group of different users. A prototype like this also needs a study of the end user interface.

Another continuation can be to build a model library. A well designed decomposition and parameterization of model objects will be the main goal. Browsers and editors must also be studied to create a good process modelling environment.

Process control system representation is a third suggestion. There is no doubt that the SEE-representation can be used to represent process models and control systems. The problem is that the SEE-system of today only have one way to present the model structure. It would be interesting to have multiple presentations of the same model. For instance one process oriented presentation and one control system presentation.

7. References

ANDERSSON, M. (1989a): "An Object-Oriented Modelling Environment," in G. Iazeolla, A. Lehman, H.J. van den Herik (Eds.): *Simulation Methodologies, languages and architectures and AI and graphics for simulation*, 1989 European Simulation Multiconference, Rome, June 7–9, 1989, The Society for Computer Simulation International, pp. 77–82.

ANDERSSON, M. (1989b): "Omola – An Object-Oriented Modelling Language," Report TFRT-7417, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

ANDERSSON, M. (1989c): "Discrete Events," private communication.

ASPEN (1985): "Aspen Plus - Introductory Manual," First Ed., Aspen Tech., Cambridge, Mass., USA.

ÅSTRÖM, K.J. and W. KREUTZER (1986): "System Representations," *IEEE Third Symposium on Computer-Aided Control System Design, Arlington, Virginia.*

ÅSTRÖM, K.J. and S.E. MATTSSON (1987): "High-Level Problem Solving Languages for Computer Aided Control Engineering," Report TFRT-3187, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

ELMQVIST, H. (1978): "A Structured Model Language for Large Continuous Systems," Ph.D. thesis TFRT-1015, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

ELMQVIST, H. (1985): "LICS - Language for Implementation of Control Systems," Report TFRT-3179, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

ELMQVIST, H. and S.E. MATTSSON (1989): "Simulator for Dynamical Systems Using Graphics and Equations for Modeling," *IEEE Control Systems Magazine*, **9**, 1, 53–58.

G2 (1988): "G2 User's Manual," version 1.1, Gensym, Cambridge,

Mass., USA.

KAEHLER, T. and D. PATTERSON (1986): *A Taste of Smalltalk*, Norton & Company, Inc., New York, NY, USA.

KEE (1988): "KEE User's Guide," version 3.1, IntelliCorp Inc.

LUYBEN, W.L. (1973): *Process Modeling, Simulation and Control for Chemical Engineers*, McGraw-Hill Book Company, New York, USA.

MATTSSON, S.E. (1988): "On Model Structuring Concepts," *Preprints of the 4th IFAC Symposium on Computer-Aided Design in Control Systems (CADCS)*, August 23–25 1988, P.R. China, pp. 269–274.

MATTSSON, S.E. (1989): "Modeling of Interactions between Submodels," in G. Iazeolla, A. Lehman, H.J. van den Herik (Eds.): *Simulation Methodologies, languages and architectures and AI and graphics for simulation*, 1989 European Simulation Multiconference, Rome, June 7–9, 1989, The Society for Computer Simulation International, pp. 63–68.

MATTSSON, S.E. and M. ANDERSSON (1989a): "A Kernel for System Representation," Report TFRT-7429, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

MATTSSON, S.E. and M. ANDERSSON (1989b): "An Environment for Model Development and Simulation," Final Report TFRT-3205, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

MOLER, C., J. LITTLE and S. BANGERT (1987): *PRO-MATLAB, User's Guide*, The MathWorks, Inc, Sherborn, MA, USA.

MOORE, R.L., L.B.HAWKINSON, M.LEVIN, A.G.HOFFMAN, B.L.MATTEWS and M.H. DAVID (1987): "Expert system methodology for real-time process control," *Proceedings of the 10th IFAC Workshop on AI in real-time control, Swansea, Sep. 21-23.*

NILSSON, B. (1987): "Experiences of Describing a Distillation Column in some Modelling Language," Report TFRT-7362, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

NILSSON, B. (1989): "Object-oriented Modelling of a Controlled Chemical Process," Report TFRT-7428, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

NILSSON, B., S.E. MATTSSON and M. ANDERSSON (1989): "Tools for Model Development and Simulation," *Proceedings of the SAIS '89 Workshop*, AILU—The AI Group in Lund, Lund University and Lund Institute of Technology.

PERKINS, J.D. (1986): "Survey of existing systems for dynamic simulation of industrial processes," *Modeling, Identification and Control*, **7**, 2.

SMALLTALK/V (1986): *Smalltalk/V, Object-Oriented Programming System*, Digitalk, Inc., Los Angeles, CA, USA.

STEFIK, M. and D.G. BOBROW (1984): "Object-Oriented Programming: Themes and Variations," *The AI Magazine*, **6**, 4.

STEPHANOPOULOS, G. (1987): "A Knowledge-based Framework for Process Design and Control," *NSF-AAAI Workshop on Artificial Intelligence in Process Engineering*, Columbia University, NY, USA.

STEPHANOPOULOS, G. (1984): *Chemical Process Control: An Introduction to Theory and Practice*, Prentice-Hall, Englewood Cliffs, NJ, USA.

TÖRNQUIST, J. (1983): "Specialkemikalier och organiska basprodukter," (Special Chemicals and Organic Base Products), *Svensk Kemisk Industri Idag*, Ingenjörsförlaget AB, Stockholm, Sweden.

ZACCHI, G. and G. ALY (1984): *Fasjämvikter för Kemitekniker*, (Phase Equilibria for Chemical Engineers), Department of Chemical Engineering, Lund Institute of Technology, Lund, Sweden.

A. Nomenclature

Equation Symbols

The symbols in the equations defined in Chapter 2 are shortly described below.

A	interfacial area, cross area, Antoine factor
A, B, C	Antoine factor
c	concentration
C_p	heat capacitivity
δx	infinitesimal lenght
k	heat transfer coefficient
k_L, k_V	mass transfer coefficient (liquid resp. vapour)
k_{lf}	pressure loss factor
k_{pump}	pump flow factor
k_{reac}	reaction coefficient
k_y	reaction coefficient at surface(liquid)
m	mass
N	mole flux
P	total pressure
p^o	partial pressure
q	volumetric flow
Q_{max}	maximum flow
Q_{trans}	heat transfer
r	reaction velocity
R_v	reaction velocity per area
S	interfacial area
T	temperature
t	time
w	mass flow
V	volume
v	velocity
x	mass fraction in liquid
y	mass fraction in vapour
γ	activity factor

ΔH_{reac}	reaction enthalpy
ΔP	pressure drop
ρ	density

Index

A, B, C, D, E	chemical component
c	cold side or cold stream
f	feed stream
h	hot side or hot stream
in	variable at the in terminal
j	generic chemical component
L	variable or parameter describing the liquid
out	variable at the out terminal
V	variable or parameter describing the vapour

B. The Process Model

This appendix is a documentation of the process models developed in the thesis. The models are presentated in "extended Omola". In each subappendix the models are organized in an alphabetical order and in class hierarchy. An subappendix that describes a subprocess model begins with a description of the class hierarchy tree. Model objects and model components that are commonly used and of general interest are organized in special library appendices.

The appendix is organized as follows:

B1: The Process Model contains the model of the main chemical process and the subprocess models.

B2: Reaction Subprocess contains a collection of the submodels used in the reaction process part model.

B3: Separation Subprocess contains decriptions of the submodels in the separation part.

B4: Common Process Submodel Library is a collection of common and general submodel classes used in the chemical process model.

B5: Process Terminal Library contains terminals that are used in the process model.

B1 The Process Model and its Subprocesses

Process_Model is the model of the main chemical process and is composed of a number of submodels, namely preparation, reaction and separation.

```
Process_Model IS A Model WITH
  %
  %  This is a model of the main
  %  chemical process.
  %
  terminals:
    Feed_A, Feed_B       IS A Vapour_In_Pipe;
    Product_C            IS A Vapour_Out_Pipe;
    Product_D, Product_E IS A Liquid_Out_Pipe;
  realizations:
    Process_Flowsheet IS A Structure WITH
      submodels:
        Preparation IS A Preparation_Part_Model;
        Reaction    IS A Reaction_Part_Model;
        Separation  IS A Separation_Part_Model;
      connections:
        Feed_A                 AT Preparation.Feed_A;
        Feed_B                 AT Preparation.Feed_B;
        Preparation.Mixed_Feed AT Reaction.Feed;
        Reaction.Products      AT Separation.Feed;
        Separation.Recycle     AT Preparation.Recycle;
        Separation.To_Recovery AT Product_C;
        Separation.Product_D   AT Product_D;
        Separation.To_Refinary AT Product_E;
     END;
END;


Preparation_Part_Model IS A Model WITH
  %
  %  This is a model of the preparation
  %  subprocess in the Process_Model.
  %
  terminals:
    Feed_A, Feed_B, Recycle IS A Vapour_In_Pipe;
    Mixed_Feed              IS A Vapour_Out_Pipe;
  realizations:
```

```
    Prep_Flow_Sheet IS A Structure WITH
      submodels:
        Valve_1, Valve_2           IS A Control_Valve;
        Mixer_1, Mixer_2           IS A Mixer;
        Compressor_1, Compressor_2 IS A Compressor;
      connections:
        Feed_A               AT Valve_1.Inlet;
        Feed_B               AT Valve_2.Inlet;
        Valve_1.Outlet       AT Mixer_1.Inlet_1;
        Valve_2.Outlet       AT Mixer_1.Inlet_2;
        Mixer_1.Outlet       AT Compressor_1.Inlet;
        Compressor_1.Outlet AT Mixer_2.Inlet_1;
        Recycle              AT Compressor_2.Inlet;
        Compressor_2.Outlet AT Mixer_2.Inlet_2;
        Mixer_2.Outlet       AT Mixed_Feed;
    END;
END;

Reaction_Part_Model IS A Model WITH
  %
  %  This is a model of the reaction
  %  subprocess used in the Process_Model.
  %  It contains one tank reactor and two
  %  tubular reactors.
  %
  terminals:
    Feed     IS A Vapour_In_Pipe;
    Products IS A Vapour_Out_Pipe;
  realizations:
    Reac_Flowsheet IS A Structure WITH
      submodels:
        Reactor_1              IS A Tank_Reactor;
        Reactor_21, Reactor_22 IS A Tube_Reactor;
        Split_1                IS A Spliter;
        Mixer_1                IS A Mixer;
      connections:
        Feed               AT Reactor_1.Inlet;
        Reactor_1.Outlet   AT Split_1.Inlet;
        Split_1.Outlet_1   AT Reactor_21.Inlet;
        Reactor_21.Outlet AT Mixer_1.Inlet_1;
        Split_1.Outlet_2   AT Reactor_22.Inlet;
        Reactor_22.Outlet AT Mixer_1.Inlet_2;
        Mixer_1.Outlet     AT Products;
```

```
    END;
END;

Separation_Part_Model IS A Model WITH
  %
  %  This is the model of the separation subprocess.
  %  Three distillation units are connected in serie.
  %  (Transform_Mass_Vol transform the volumetric
  %   inflows and outflows to internal flows based
  %   on mass fractions.)
  %
  terminals:
    Feed                    IS A Vapour_In_Pipe;
    Recycle, To_Recovery    IS A Vapour_Out_Pipe;
    Product_D, To_Refinary IS A Liquid_Out_Pipe;
  realizations:
    Reac_Flowsheet IS A Structure WITH
      submodels:
        T1, T2, T3, T4, T5 IS A Transform_Mass_Vol;
        Column_1 IS A Flash_Column_1;
        Column_2 IS A Distillation_Column_2;
        Column_3 IS A Distillation_Column_3;
      connections:
        Feed              AT T1.Vol
        T1.Mass           AT Column_1.Feed;
        Column_1.Top      AT T2.Mass
        T2.Vol            AT To_Recovery;
        Column_1.Bottom AT Column_2.Feed;
        Column_2.Top      AT T3.Mass
        T3.Vol            AT Recycle;
        Column_2.Bottom AT Column_3.Bottom;
        Column_3.Top      AT T4.Mass
        T4.Vol            AT Product_D;
        Column_3.Bottom AT T5.Mass
        T5.Vol            AT To_Refinary;
    END;
END;
```

B2 The Reaction Subprocess

This subappendix contains the submodels in the reaction subprocess. It is done after the model class hierarchy tree below. The submodels are stored as the hierarchy tree indicates. Models in parenthesis can be found in the common process model library in Appendix B4.

Class Hierarchy Tree

```
Model
   Reactor_Vessel
   RegStructure1N
   Tank_Reactor
   Tube_Reaction_Data_Model
   TubeSideElement
      ColdTubeSide
      HotTubeSide
      Tube_Machine_Model
   (Heat_Exchanger_Model)
      Tube_Slice
   Tubular_Reactor

RecordTerminal
   Tube_Reaction_Data_Terminal
```

Model Descriptions

```
Reactor_Vessel IS A Model WITH
  %
  %  This is a model for the bubble tank reactor vessel.
  %  It is based on dynamic component mass balances
  %  (composition) and a dynamic energy balance
  %  (temperature). The products evaporate from the
  %  liquid (evap_flux) and the chemical reaction is
  %  assumed to be of the first order with repect to the
  %  concentrations of A and B.
  %
  terminals:
    V_In  IS A Vapour_In_Pipe;
```

```
  V_Out IS A Vapour_Out_Pipe;
  L_In  IS A Liquid_In_Pipe;
  L_Out IS A Liquid_Out_Pipe;
parameters:
  No_Comp IS A Parameter WITH value:=5; END;
  Volume, Area, Density, K_reac, HeatCap, GasConst
    Reac_Heat, Pressure IS A Parameter;
  KL          TYPE Matrix[No_Comp,No_Comp];
  comp_equil TYPE Column[No_Comp];
constraints:
  V_In.Composition.Lenght :- V_Out.Composition.Lenght :-
    L_In.Composition.Lenght :-
    L_Out.Composition.Lenght :- No_Comp;
realizations:
  Ideal_Tank_Real IS A Primitive WITH
    variables:
      composiiton          TYPE Column[No_Comp];
      temperature, r_velo TYPE Real;
    equations:
      % Component Mass Balance with Reaction
      Volume*DOT(composition) =
        V_In.Flow*V_In.Compositon -
        evap_flux + Volume*Reac_Velocity;
      % Energy Balance with Reaction Heat
      Density*Volume*HeatCap*DOT(temperature) =
        Density*L_In.Flow*L_In.Temperature -
        Density*L_Out.Flow*L_Out.Temperature +
        Reac_Heat*Volume*r_velo;
      % Evaporation
      evap_flux = Area*KL*(comp_equil - composition);
      % Reaction
      r_velo = K_reac*composition[1]*composition[2];
      Reac_Velocity = [-R_V; -R_V; R_V; R_V; 0];
      % Static Momentum Balance
      V_In.Pressure  = Pressure;
      V_Out.Pressure = Pressure;
      L_In.Pressure  = Pressure;
      L_Out.Pressure = Pressure;
      % Liquid Out Flow
      L_Out.Temperature = temperature;
      L_Out.Composiiton = composition;
      % Vapour Out Flow
      V_Out.Temperature = temperature;
```

```
        V_Out.Composition = evap_flux/V_Out.Flow;
    END;
END;

RegStructure1N IS A Model WITH
  %
  %  A general purpose one dimensional
  %  regular structure model.
  %
  terminals:
    InBottom, InTop   IS A Process_In_Pipe;
    OutBottom, OutTop IS A Process_Out_Pipe;
  parameters:
    m IS A Parameter WITH value:=1; END;
    n IS A Parameter WITH value:=5; END;
  realization:
    Regular IS A Structure WITH
      submodels:
        Element[m..n] IS A Element_Model;
      connections:
        Element[m..n-1].OutTop AT Element[m+1..n].InBottom;
        Element[m..n-1].InTop  AT Element[m+1..n].OutBottom;
        InBottom               AT Element[m].InBottom
        OutBottom              AT Element[m].OutBottom
        InTop                  AT Element[n].InTop;
        OutTop                 AT Element[n].OutTop;
    END;
END;

Tank_Reactor IS A Model WITH
  %
  %  A model describing the tank reactor configuration.
  %  It contains submodels of Reactor_Vessel,
  %  Centrifugal_Pump and Heat_Exchanger_Model.
  %
  terminals:
    Feed     IS A Vapour_In_Pipe;
    Products IS A Vapour_Out_Pipe;
  realizations:
    Bubble_Tank_Real IS A Structure WITH
      submodels:
        Vessel IS A Reactor_Vessel;
        Pump_1 IS A Centrifugal_Pump;
```

```
        Hex_1  IS A Heat_Exchanger_Model;
      connections:
        Feed               AT Vessel.Vapour_In;
        Vessel.Vapour_Out  AT Products;
        Vessel.Liquid_Out  AT Pump_1.Inlet;
        Pump_1.Outlet      AT Hex_1.Inlet;
        Hex_1.Outlet       AT Vessel.Liquid_In;
    END;
END;

TubeSideElement IS A Model WITH
  %
  %  A super-class for elements in
  %  the tubular reactor model.
  %
  terminals:
    Inlet   IS A Vapour_In_Pipe;
    Outlet  IS A Vapour_Out_Pipe;
    Heat_In IS A Heat_Transfer_In;
  parameter:
    No_Comp IS A Parameter WITH value:=5; END;
    Lenght, Area, Heat_Surface IS A Parameter;
  constraints:
    Inlet.Composition.Length :-
      Outlet.Composition.Length :- No_Comp;
END;

ColdTubeSide IS A TubeSideElement WITH
  %
  %  A model describing the cold side of the tube.
  %  A constant volume vessel with perfect mixing and
  %  energy interaction.
  %
  parameter:
    GasConst, HeatCap, loss_factor IS A Parameter;
    MoleWeight IS A Row[No_Comp];
  realization:
    Tube_Behaviour IS A Primitive WITH
      variables:
        composition TYPE Column[No_Comp];
        density, pressure, temperature TYPE Real;
      equations:
        % Component Mass Balance
```

```
        Volume*DOT(composition) =
          Inlet.Flow*Inlet.Composition -
          Outlet.Flow*Outlet.Composition;
        Outlet.Composition = composition;
        % general gas law
        pressure*Volume =
          GasConst*SUM(composition)*temperature;
        density = MoleWeight*composition
        % Energy Balance
        DOT(energy) =
          density*Inlet.Flow*HeatCap*Inlet.Temperature +
          density*Outlet.Flow*HeatCap*Outlet.Temperature +
          Heat_In.Flux*Heat_Surface;
        energy = density*Volume*HeatCap*temperature;
        Heat_In.Temperature = temperature;
        Outlet.Temperature  = temperature;
        % Static Momentum Balance
        Outlet.Flow = Inlet.Flow;
        Inlet.Pressure - Outlet.Pressure =
          loss_factor*(Inlet.Flow/Area)^2/2;
        Outlet.Pressure = pressure;
    END;
END;

HotTubeSide IS A TubeSideElement WITH
  %
  %  A model for the hot side of the tube in
  %  the tubular reactor.
  %  The model is medium and machine decomposed into
  %  Tube_Machine_Model and Tube_Reaction_Data_Model.
  %
  parameter:
    Reac_Surface IS A Parameter;
  realization:
    MM_Structure IS A Structure WITH
      submodels:
        Tube   IS A Tube_Machine_Model;
        Medium IS A Tube_Reaction_Data_Model:
      constraint:
        Tube.Length       :- Length;
        Tube.Area         :- Area;
        Tube.Heat_Surface :- Heat_Surface;
        Tube.Reac_Surface :- Reac_Surface;
```

```
      connection:
        Inlet       AT Tube.Inlet;
        Outlet      AT Tube.Outlet;
        Heat_In     AT Tube.Heat_In;
        Tube.MData AT Medium.MData;
    END;
END;

Tube_Machine_Model IS A TubeSideElement WITH
  %
  %  A machine model of hot tube side based on constant
  %  volume vessel with perfect mixing and heat interaction.
  %  Description of the gas-solid reaction is done in the
  %  medium model through the MData terminal.
  %
  terminal:
    MData IS A Tube_Reaction_Data_Terminal;
  parameters:
    loss_factor, GasConst IS A Parameter;
    MoleWeight IS A Row[No_Comp];
  constraint:
    MData.Comp.Lenght :- No_Comp;
  realization:
    Tube_Behaviour IS A Primitive WITH
      variables:
        composition TYPE Column[No_Comp];
        volume, density, pressure, temperature TYPE Real;
      equations:
        % Component Mass Balance
        Volume*DOT(composition) =
          Inlet.Flow*Inlet.Composition -
          Outlet.Flow*Outlet.Composition +
          Reac_Surface*MData.Mole_Flux;
        % general gas law
        pressure*volume =
          GasConst*SUM(composition)*temperature;
        density = MoleWeight*composition;
        volume = Length*Area;
        % Energy Balance
        DOT(energy) =
          density*Inlet.Flow*HeatCap*Inlet.Temperature +
          Heat_Surface*Heat_In.Flux +
          Reac_Surface*MData.Reac_Heat -
```

```
          density*Outlet.Flow*HeatCap*Outlet.Temperature;
        energy = density*Volume*HeatCap*temperature;
        Heat_In.Temperature = temperature;
        Outlet.Temperature  = temperature;
        % Static Momentum Balance
        Outlet.Flow = Inlet.Flow;
        Inlet.Pressure - Outlet.Pressure =
          loss_factor*(Inlet.Flow/Area)^2/2;
        Outlet.Pressure = pressure;
        % Medium Model Communication
        MData.Pressure = pressure;
        MData.Temp = temperature;
        MData.comp = composition;
    END;
END;

Tube_Reaction_Data_Model IS A Model WITH
  %
  %  This is the medium model which is used together with
  %  Tube_Machine_Model.
  %  It describe the gas-solid reaction and the static mass
  %  transfer phenomenon on the catalyst surface.
  %
  terminals:
    MData IS A Tube_Reaction_Data_Terminal;
  parameters:
    No_Comp := 5
    Kag TYPE Matrix[No_Comp,No_Comp];
    Ky, Heat_Reaction IS A Parameter;
  constraints:
    MData.No_Comp :- MData.Comp.Length :-
      MData.Mole_Flux.Length :- No_Comp;
  realizations:
    Real1 IS A Primitive WITH
      variables:
        Surface_Comp, Reac_Velo TYPE Column[No_Comp];
        Rv TYPE Real;
      equations:
        MData.Mole_Flux = Kag*(MData.Comp .- Surface_Comp);
        Rv = Ky*Surface_Comp[1]*Surface_Comp[4];
        Reac_Velo = [-Rv; 0; Rv; -Rv; Rv];
        MData.Mole_Flux = Reac_Velo;
        MData.Reac_Heat = Heat_Reaction*Rv;
```

```
    END;
END;

Tube_Slice IS A Heat_Exchanger_Model WITH
  %
  %  This is composite model connecting the hot tube side
  %  and the cold tube side together with a heat transfer
  %  model. This is therefore a specialization of the heat
  %  exchanger model based on a transport phenomenon
  %  decomposition.
  %
  parameters:
    Length, Area_Cold, Area_Hot, Heat_Surface,
      Reac_Surface IS A Parameter;
  realization:
    Tube_Real IS A Hex_Real WITH
      submodels:
        Hot_Side  IS A HotTubeSide;
        Cold_Side IS A ColdTubeSide;
      constraints:
        Cold_Side.Lenght :- Hot_Side.Lenght :- Lenght;
        Cold_Side.Area :- Area_Cold;
        Hot_Side.Area :- Area_Hot;
        Cold_Side.Heat_Surface :-
          Hot_Side.Heat_Surface :- Heat_Surface;
        Hot_Side.Reac_Surface :- Reac_Surface;
    END;
END;

Tubular_Reactor IS A Model WITH
  %
  %  The super-model using the regular structure model
  %  in order to structure a number of tube slices in
  %  a serie.
  %
  terminals:
    Inlet  IS A Vapout_In_Pipe;
    Outlet IS A Vapour_Out_Pipe;
  parameters:
    Tube_Length, Total_Heat_Area, Total_Reac_Surface,
      Cold_Cross_Area, Hot_Cross_Area IS A Parameter;
  realizations:
    ReactorRealization IS A Structure WITH
```

```
    parameters:
      No_slices IS A Parameter WITH value:= 10; END;
    submodels:
      Reactor_Strucure IS A RegStructure1N WITH
        Element_Model IS A Tube_Slice;
      END;
    constraints:
      Reactor_Structure.n :- No_Slices;
      Reactor_Structure.Element[1..n].Lenght :-
        Tube_Lenght/No_Slices;
      Reactor_Structure.Element[1..n].Heat_Surface :-
        Total_Heat_Area/No_Slices;
      Reactor_Structure.Element[1..n].Reac_Surface :-
        Total_Reac_Surface/No_Slices;
      Reactor_Structure.Element[1..n].Area_Cold :-
        Cold_Cross_Area;
      Reactor_Structure.Element[1..n].Area_Hot :-
        Hot_Cross_Area;
    connections:
      Inlet                          AT
        Reactor_Structure.InBottom;
      Reactor_Structure.OutTop       AT
        Reactor_Structure.InTop;
      Reactor_Structure.OutBottom AT Outlet;
  END;
END;

Tube_Reaction_Data_Terminal IS A RecordTerminal WITH
  %  This is a terminal descrbing the interaction between
  %  the machine (tube) and the medium (reaction) models
  %  the tubular reactor.
  components:
    No_Comp   IS A SimpleTerminal;
    Pressure  IS A Pressure_Terminal;
    Temp      IS A Temperature_Terminal;
    Comp      IS A Composition_Terminal;
    Mole_Flux IS A Mole_Flux_Terminal;
    Reac_Heat IS A Heat_Data_Terminal;
END;
```

B3 Separation Subprocess

This subappendix is organized as the previous one. First is the model class hierarchy listed. After that is the submodel described in alphabetical order.

Class Hierarchy

```
Model
   Condenser_Gen_Model
      Partial_Condenser_Generic
      Total_Condenser_Generic
   Distillation_Column_BS
      Distillation_Column_with_PC
         Distillation_Column_2
         Flash_Column_1
      Distillation_Column_with_TC
         Distillation_Column_3
   Distillation_Generic_Element
      Column_Model
      Column_Part_Model
      Tray_Generic_Model
      Tray_Model
         Feed_Tray_Model
   Distillation_Media_Model
   Reboiler_Machine_Model
   Reboiler_Model
   Reflux_Drum
   Total_Condenser
      Partial_Condenser

RecordTerminal
   Media_Data_Terminal
```

Model Descriptions

```
Condenser_Gen_Model IS A Model WITH
  %
  % A super-class for condensers.
```

```
  %
  terminals:
    VIn   IS A Vapour_In;
    LOut  IS A Liquid_Out;
    MData IS A Media_Data_Terminal;
  parameters:
    Cooling_Heat, pressure IS A Parameter;
  constraints:
    VIn.Comp.Lenght :- LOut.Comp.Lenght :- MData.No_Comp;
END;

Partial_Condenser_Generic IS A Condenser_Gen_Model WITH
  %
  % A model for partial condensors.
  %
  terminals:
    VOut IS A Vapour_Out;
  constraint:
    VOut.Comp.Lenght :- MData.No_Comp;
  realizations:
    Partial_Condens_Realization IS A Primitive WITH
      variable:
        temp TYPE Real;
      equations:
        VIn.Flow*VIn.Comp - LOut.Flow*LOut.Comp -
          VOut.Flow*VOut.Comp = 0;
        VIn.Flow*VIn.Temp - LOut.Flow*LOut.Temp -
          VOut.Flow*VOut.Temp +
          LOut.Flow*MData.Cond_Heat - Cooling_Heat = 0;
        SUM(LOut.Comp) = 1;
        SUM(VOut.Comp) = 1;
        MData.Temp = LIn.Temp;
        MData.Pressure = pressure;
        MData.X = LOut.Comp;
        LOut.Comp = MData.X;
        LOut.Temp = temp;
        LOut.Pressure = pressure;
        VOut.Comp = MData.Y;
        VOut.Temp = temp;
        VOut.Pressure = pressure;
    END;
END;
```

```
Total_Condenser_Generic IS A Condenser_Gen_Model WITH
  %
  % A model for a total condenser.
  %
  realizations:
    Total_Condens_Realization IS A Primitive WITH
      equations:
        VIn.Flow*VIn.Comp - LOut.Flow*LOut.Comp = 0;
        VIn.Flow*VIn.Temp - LOut.Flow*LOut.Temp +
          LOut.Flow*MData.Cond_Heat - Cooling_Heat = 0;
        SUM(LOut.Comp) = 1;
        LOut.Pressure = VIn.Pressure;
    END;
END;

Distillation_Column_BS IS A Model WITH
  %
  % A super-class for distillation units.
  % It describe the basic structure (BS)
  % of the reboiler contguration.
  %
  terminals:
    Feed   IS A Liquid_In;
    Bottom IS A Liquid_Out;
  constraints:
    Feed.Comp.Lenght :- Bottom.Comp.Lenght :- No_Comp;
  parameters:
    No_Comp      := 5;
    No_trays     := 5;
    Feed_tray_at := 4;
  realizations:
    Column_Structure IS A Structure WITH
      submodels:
        Column       IS A Column_Model;
        Reboiler     IS A Reboiler_Model;
        Bottom_Valve IS A Control_Valve_D;
      constraints:
        Column.No_Comp :- Reboiler.No_Comp :-
          Bottom_Valve.No_Comp :- No_Comp;
        Column.No_trays :- No_trays;
        Column.Feed_tray_at :- Feed_tray_at;
      connections:
        Feed          AT Feed OF Column;
```

```
        Reboiler.LIn  AT Column.LOut;
        Reboiler.VOut AT Column.VIn;
        Reboiler.LOut AT Bottom_Valve.Inlet;
        Bottom        AT Bottom_Valve.Outlet;
    END;
END;

Distillation_Column_With_PC IS A Distillation_Column_BS WITH
  %
  % A distillation unit with a partial condenser.
  %
  terminals:
    Top IS A Vapour_Out;
  constraint:
    Top.Comp.Lenght :- No_Comp;
  realizations:
    Column_Structure_PC IS A Column_Structure WITH
      submodels:
        Condenser IS A Partial_Condenser;
        Top_Valve IS A Control_Valve_D;
      constraint:
        Condenser.No_Comp :- Top_Valve.No_Comp :- No_Comp;
      connections:
        Column.VOut     AT Condenser.VIn;
        Condenser.LOut  AT Column.LIn;
        Top_Valve.Inlet AT Condenser.VOut;
        Top             AT Top_Valve.Outlet;
    END;
END;

Distillation_Column_2 IS A Distillation_Column_With_PC WITH
  %
  % A distillation unit with partial condenser
  % describing the second unit in the chemical
  % process.
  %
  parameters:
    No_Comp      := 5 ;
    No_trays     := 22;
    Feed_tray_at := 12;
END;

Flash_Column_1 IS A Distillation_Column_With_PC WITH
```

```
  %
  % A distillation unit with partial condeser
  % describing the first unit in the process.
  %
  parameters:
    No_Comp      := 5;
    No_trays     := 5;
    Feed_tray_at := 4;
END;

Distillation_Column_With_TC IS A Distillation_Column_BS WITH
  %
  % A distillation unit model with total condenser
  % configuration.
  %
  terminals:
    Top IS A Liquid_Out;
  constraint:
    Top.Comp.Lenght :- No_Comp;
  realizations:
    Column_Structure_TC IS A Column_Structure WITH
      submodels:
        Condenser    IS A Total_Condenser;
        Reflux_Drum  IS A Reflux_Drum_Model;
        Reflux_Pump  IS A Cent_Pump_D;
        Spliter_1    IS A Spliter;
        T1, T2, T3   IS A Transform_Mass_Vol;
        Reflux_Valve IS A Control_Valve_D;
        Top_Valve    IS A Control_Valve_D;
      constraint:
        Condenser.No_Comp :- Top_Valve.No_Comp :-
          Reflux_Drum.No_Comp :- Reflux_Pump.No_Comp :-
          Spliter_1.No_Comp :- Reflux_Valve.No_Comp :-
          No_Comp;
      connections:
        Top                  AT Top_Valve.Outlet;
        Column.VOut          AT CondenserVIn;
        Condenser.LOut       AT Reflux_Drum.LIn;
        Reflux_Drum.LOut     AT Reflux_Pump.Inlet;
        Reflux_Pump.Outlet AT T1.Mass;
        T1.Vol               AT Spliter_1.Inlet;
        Spliter_1.Outlet_1 AT T2.Vol;
        T2.Mass              AT Reflux_Valve.Inlet;
```

```
        Column.LIn          AT Reflux_Valve.Outlet;
        Spliter_1.Outlet_2  AT T3.Vol;
        T3.Mass             AT Top_Valve.Inlet;
    END;
END;

Distillation_Column_3 IS A Distillation_Column_With_TC WITH
  %
  % A distillation unit with total condenser describing
  % the third unit in the process.
  %
  parameters:
    No_Comp      := 5 ;
    No_trays     := 50;
    Feed_tray_at := 35;
END;

Distillation_Generic_Element IS A Model WITH
  %
  % A super-class for objects in the distillation units.
  %
  terminals:
    LIn  IS A Liquid_In;
    VOut IS A Vapour_Out;
    LOut IS A Liquid_Out;
    VIn  IS A Vapour_In;
  parameter:
    No_Comp := 5;
  constraints:
    LIn.Comp.Lenght :- VIn.Comp.Length :-
      LOut.Comp.Length :- VOut.Comp.Length :- No_Comp;
END;

Column_Model IS A Distillation_Generic_Element WITH
  %
  % A model describing the internal structure
  % of a distillation column.
  %
  terminals:
    Feed IS A Liquid_In;
  parameters:
    No_trays     := 10;
    Feed_tray_at := 5;
```

```
  constraint:
    Feed.Comp.Lenght :- No_Comp;
  realizations:
    Column_Structure IS A Structure WITH
      submodels:
        Stripper IS A Column_Part_Model WITH
          parameters:
            bottom := 1;
            top    := Feed_tray_at - 1;
        END;
        Rectifyer IS A Column_Part_Model WITH
          parameters:
            bottom := Feed_tray_at + 1;
            top    := No_tray;
        END;
        Feed_Tray IS A Feed_Tray_Model;
      constraint:
        Stripper.No_Comp :- Rectifyer.No_Comp :-
          Feed_Tray.No_Comp :- No_Comp;
      connections:
        LIn             AT Restifyer.LIn;
        Restifyer.LOut  AT Feed_Tray.LIn;
        Feed            AT Feed_Tray.Feed;
        Feed_Tray.LOut  AT Stripper.LIn;
        Stripper.LOut   AT LOut;
        VIn             AT Stripper.VIn;
        Feed_Tray.VIn   AT Stripper.VOut;
        Restifyer.VIn   AT Feed_Tray.VOut;
        VOut            AT Restifyer.VOut;
END;

Column_Part_Model IS A Distillation_Generic_Element WITH
  %
  % A regular structure model for parts
  % of distillation column.
  %
  parameters:
    bottom := 1;
    top    := 5;
  realizations:
    Column_Regular_Structure IS A Structure WITH
      submodels:
        Tray[bottom..top] IS A Tray_Model;
```

```
      constraint:
        No_Comp :- Tray[bottom].No_Comp;
      connections:
        LIn                       AT Tray[top].LIn;
        Tray[bottom..top-1].LIn AT Tray[bottom+1..top].LOut;
        LOut                      AT Tray[bottom].LOut;
        VIn                       AT Tray[bottom].VIn;
        Tray[bottom+1..top].VIn AT Tray[bottom..top-1].VIn;
        VOut                      AT Tray[top].VOut;
    END;
END;

Tray_Generic_Model IS A Distillation_Generic_Element WITH
  %
  % A tray machine model which can be connected
  % with a distillation media model in a tray model.
  % It is composed of dynamic mass, component and
  % energy balances.
  %
  terminals:
    MData IS A Media_Data_Terminal;
  parameters:
    lmin, Mmin, beta, pressure IS A Parameter;
  constraint:
    No_Comp :- MData.NoComp;
  realizations:
    Tray_Realization IS A Primitive WITH
      variables:
        Mcomp, comp TYPE Column No_Comp;
        Mtemp, M    TYPE Real;
      equations:
        DOT(Mcomp) = LIn.Flow*LIn.Comp + VIn.Flow*VIn.Comp -
          LOut.Flow*LOut.Comp - VOut.Flow*VOut.Comp;
        DOT(Mtemp) = LIn.Flow*LIn.Temp + VIn.Flow*VIn.Temp -
          LOut.Flow*LOut.Temp - VOut.Flow*VOut.Temp;
        M = SUM(Mcomp);
        comp = Mcomp/M;
        temp = Mtemp/M;
        MData.X = comp;
        MData.Pressure = pressure;
        LOut.Flow = lmin + (M - Mmin)/beta;
        LOut.Pressure = pressure;
        LOut.Comp = comp;
```

```
        LOut.Temp = temp;
        VOut.Flow = VIn.Flow;
        VOut.Pressure = pressure;
        VOut.Comp = MData.Y;
        VOut.Temp = temp;
    END;
END;

Tray_Model IS A Distillation_Generic_Element WITH
  %
  % A composite model connecting a tray machine model
  % and a distillation media model together.
  %
  realizations:
    Tray_Structure IS A Structure WITH
      submodels:
        Tray   IS A Tray_Generic_Model;
        Medium IS A Distillation_Media_Model;
      constraint:
        No_Comp :- Media.No_Comp;
      connections:
        LIn         AT Tray.LIn;
        LOut        AT Tray.LOut;
        VIn         AT Tray.VIn;
        VOut        AT Tray.VOut;
        Tray.MData AT Media.Op_Cond;
    END;
END;

Feed_Tray_Model IS A Tray_Model WITH
  %
  % A special tray model for feed inlet.
  % The tray machine model is specialized
  % to capture an additional inlet.
  %
  terminals:
    Feed IS A Liquid_In;
  realizations:
    Feed_Tray_Structure IS A Tray_Structure WITH
      submodels:
        Tray IS A Tray_Generic_Model WITH
          terminals:
            Feed IS A Liquid_In;
```

```
        realizations:
          Tray_Realization IS A Primitive WITH
            variables:
              Mcomp, comp TYPE Column No_Comp;
              Mtemp, M TYPE Real;
            equations:
              DOT(Mcomp) = LIn.Flow*LIn.Comp +
                VIn.Flow*VIn.Comp + Feed.Flow*Feed.Comp -
                LOut.Flow*LOut.Comp - VOut.Flow*VOut.Comp;
              DOT(Mtemp) = LIn.Flow*LIn.Temp +
                VIn.Flow*VIn.Temp + Feed.Flow*Feed.Temp -
                LOut.Flow*LOut.Temp - VOut.Flow*VOut.Temp;
              M = SUM(Mcomp);
              comp = Mcomp/M;
              MData.X = comp;
              MData.Pressure = pressure;
              MData.Temp = Mtemp/M;
              LOut.Flow = lmin + (M - Mmin)/beta;
              LOut.Comp = comp;
              LOut.Temp = Mtemp/M;
              LOut.Pressure = pressure;
              VOut.Flow = VIn.Flow;
              VOut.Comp = MData.Y;
              VOut.Temp = LOut.Temp;
              VOut.Pressure = pressure;
          END;
        END;
      constraint:
        No_Comp :- MData.No_Comp;
      connections:
        Feed AT Tray.Feed;
    END;
END;

Distillation_Media_Model IS A Model WITH
  %
  % A distillation media model.
  % It can be connected to a tray machine model
  % or a reboiler machine model.
  %
  terminals:
    Op_Cond IS A Media_Data_Terminals;
  parameters:
```

```
    No_Comp := 5;
    Cond_Heat, Evap_Heat IS A Parameter;
    A, B, C TYPE Column No_Comp;
  constraint:
    Op_Cond.No_Comp :- Op_Cond.X.Lenght :-
      Op_Cond.Y.Lenght :- No_Comp;
  realizations:
    Ideal_Four_Comp IS A Primitive WITH
      equations:
        LogPo = A - B ./ (C + Op_Cond.Temp)';
        Op_Cond.Y =
          EXP(LogPo)/Op_Cond.Pressure .* Op_Cond.X;
        Op_Cond.Cond_Heat = Cond_Heat;
        Op_Cond.Evap_Heat = Evap_Heat;
    END;
END;

Reboiler_Machine_Model IS A Model WITH
  %
  % A reboiler machine model which can be connected
  % to a distillation media model.
  %
  terminals:
    LIn         IS A Liquid_In;
    LOut        IS A Liquid_Out;
    VOut        IS A Vapour_Out;
    MData       IS A Distillation_Media_Model
    Heat_Apply IS A Heat_Transfer_In;
  constraints:
    LIn.Comp.Length :- LOut.Comp.Length :-
      VOut.Comp.Length :- MData.No_Comp;
  parameter:
    pressure TYPE Real;
  realization:
    Boiler_Realization IS A Primitive WITH
      variables:
        Mcomp TYPE Column MData.No_Comp;
        Mtemp TYPE Real;
      equations:
        DOT(Mcomp) = LIn.Flow*LIn.Comp -
          VOut.Flow*VOut.Comp - LOut.Flow*LOut.Comp;
        DOT(Mtemp) = LIn.Flow*LIn.Temp + Heat_Apply.Flux -
          VOut.Flow*VOut.Temp - LOut.Flow*LOut.Temp -
```

```
          VOut.Flow*MData.Evap_Heat;
        M = SUM(Mcomp);
        comp = Mcomp/M;
        temp = Mcomp/M;
        LOut.Comp = comp;
        LOut.Temp = temp;
        LOut.Pressure = pressure;
        MData.X = comp;
        MData.Temp = temp;
        Heat_Apply.Temp = temp;
        MData.Pressure = pressure;
        VOut.Pressure = pressure;
    END;
END;

Reboiler_Model IS A Model WITH
  %
  % A composite model connecting a reboiler
  % machine model to a distillation media model.
  %
  terminals:
    LIn  IS A Liquid_In;
    LOut IS A Liquid_Out;
    VOut IS A Vapour_Out;
  realizations:
    Boiler_Structure IS A Structure WITH
      submodels:
        Boiler_Side IS A Reboiler_Machine_Model;
        Medium      IS A Distillation_Media_Model;
        Heat_Trans  IS A Heat_Transfer_Model;
        Hot_Side    IS A PerfectMixing;
      constraint:
        LIn.Comp.Lenght :- LOut.Comp.Lenght :-
          VOut.Comp.Lenght :- Media.No_Comp;
      connections:
        LIn                    AT Boiler_Side.LIn;
        LOut                   AT Boiler_Side.LOut;
        VOut                   AT Boiler_Side.VOut;
        Boiler_Side.MData      AT Media.Op_Cond;
        Boiler_Side.Heat_Apply AT Heat_Trans.Heat_Out;
        Hot_Side.Heat_In       AT Heat_Trans.Heat_In;
    END;
END;
```

```
Reflux_Drum IS A Model WITH
  %
  % A reflux drum model.
  %
  terminals:
    LIn  IS A Liquid_In;
    LOut IS A Liquid_Out;
  constraints:
    LIn.Comp.Length :- LOut.Comp.Length :- No_Comp;
  parameters:
    No_Comp := 5;
    pressure IS A Parameter;
  realizations:
    Drum_Realization IS A Primitive WITH
      variables:
        Mcomp TYPE Column No_Comp;
        Mtemp TYPE Real;
      equations:
        DOT(Mcomp) = LIn.Flow*LIn.Comp -
          LOut.Flow*LOut.Comp;
        DOT(Mtemp) = LIn.Flow*LIn.Temp -
          LOut.Flow*LOut.Temp;
        M = SUM(Mcomp);
        LOut.Comp = Mcomp/M;
        LOut.Temp = Mtemp/M;
        LOut.Pressure = pressure;
    END;
END;

Total_Condenser IS A Model WITH
  %
  % A model of a total condenser.
  %
  terminals:
    Vin  IS A Vapour_In;
    LOut IS A Liquid_Out;
  realizations:
    Condenser_Structure IS A Structure WITH
      submodels:
        Condenser  IS A Total_Condenser_Generic;
        Medium     IS A Distillation_Media_Model;
      constraints:
```

```
        VIn.Comp.Lenght :- LOut.Comp.Lenght :-
          Media.No_Comp;
      connections:
        LOut             AT Condenser.LOut;
        VIn              AT Condenser.VIn;
        Condenser.MData AT Media.Op_Cond;
    END;
END;

Partial_Condenser IS A Total_Condenser WITH
  %
  % A model of a partial condenser which
  % is a subclass of the total condenser
  % model.
  %
  terminals:
    VOut IS A Vapour_Out;
  realizations:
    Part_Cond_Structure IS A Condenser_Structure WITH
      submodels:
        Condenser IS A Partial_Condenser_Generic;
      constraint:
        VOut.Comp.Length :- Media.No_Comp;
      connections:
        VOut AT Condenser.VOut;
    END;
END;

Media_Data_Terminal IS A Record_Terminal WITH
  %
  % A definition of a terminal for machine and
  % distillation media connections.
  %
  components:
    No_Comp   IS A Simple_Terminal;
    Cond_Heat IS A Heat_Data_Terminal;
    Evap_Heat IS A Heat_Data_Terminal;
    Temp      IS A Temperature_Terminal;
    Pressure  IS A Pressure_Terminal;
    X, Y      IS A Mass_Composition_Terminal;
END;
```

B4 Common Process Submodel Library

Models of general interest and commonly used are stored in this library. The models can be seen below but first the model class hierarchy is shown and the model are then found in that order.

Class Hierarchy

```
Model
   Cent_Pump_D
   Control_Valve
      Control_Valve_D
   Heat_Exchanger_Model
   Heat_Transfer_Model
   Mixer
   Multi_Flow_Element
   Servo_Motor
   Spliter
   Static_Flow_Object
      Centrifugal_Pump
         Compressor
      Static_Valve
      Static_Pipe
      Valve_Lin
   Transform_Mass_Vol
   Vessel
      GasVessel
      PerfectMixing
      Tank
```

Submodels

```
Cent_Pump_D IS A Model WITH
  %
  % A centrifugal pump model with mass fraction
  % based termianls. The model transform the
  % terminals to concentrations and uses the
  % concentration based centrifugal pump model.
  %
  terminals:
```

```
    Inlet  IS A Liquid_In;
    Outlet IS A Liquid_Out;
  parameters:
    No_Comp := 5;
  constraints:
    Inlet.Comp.Lenght :- Outlet.Comp.Lenght :- No_Comp;
  realizations:
    Pump_Real_D IS A Structure WITH
      submodels:
        Pump     IS A Centrifugal_Pump;
        T_1, T_2 IS A Transform_Mass_Vol;
      constraint:
        Pump.No_Comp :- T_1.No_Comp :-
          T_2.No_Comp :- No_Comp;
      connections:
        Inlet   AT T_1.Mass;
        T_1.Vol AT Pump.Inlet;
        Outlet  AT T_2.Mass;
        T_2.Vol AT Pump.Outlet;
    END;
END;

Control_Valve IS A Model WITH
  %
  % A composite model that connect a valve
  % to a servo motor.
  %
  terminals:
    Control_Signal IS A SimpleTerminal;
    Inlet          IS A In_Pipe;
    Outlet         IS A Out_Pipe;
  realizations:
    RealCV IS A Structure WITH
      submodels:
        Servo IS A Servo_Motor;
        Valve IS A Valve_Lin;
      connections:
        Control_Signal AT Servo.Control_Signal;
        Servo.Position AT Valve.Position;
        Inlet          AT Valve.Inlet;
        Outlet         AT Valve.Outlet;
    END;
END;
```

```
Control_Valve_D IS A Control_Valve WITH
  %
  % A control valve with mass fraction based terminals.
  % Uses the concentration based control valve.
  %
  terminals:
    Inlet  IS A Liquid_In;
    Outlet IS A Liquid_Out;
  parameter:
    No_Comp := 5;
  constraints:
    Inlet.Comp.Lenght :- Outlet.Comp.Lenght :- No_Comp;
  realizations:
    RealCV_D IS A RealCV WITH
      submodels:
        Trans_1, Trans_2 IS A Transform_Mass_Vol;
      connections:
        Inlet         AT Trans_1.Mass;
        Trans_1.Vol   AT Valve.Inlet;
        Valve.Outlet AT Trans_2.Mass;
        Trans_2.Vol   AT Valve.Outlet;
    END;
END;

Heat_Exchanger_Model IS A Model WITH
  %
  % A heat exchanger model.
  % This is a composite model based on
  % a transport phenomenon decomposition.
  %
  terminals:
    Inlet  IS A In_Pipe;
    Outlet IS A Out_Pipe;
  realizations:
    Hex_Real IS A Structure WITH
      submodels:
        Hot_Side, Cold_Side IS A PerfectMixing
        HT_Model            IS A Heat_Transfer_Model;
      connections:
        Inlet              AT Cold_Side.Inlet;
        Outlet             AT Cold_Side.Outlet;
        Cold_Side.Heat_In AT HT_Model.Heat_Out;
```

```
        Hot_Side.Heat_In  AT HT_Model.Heat_In;
    END;
END;

Heat_Transfer_Model IS A Model WITH
  %
  % A model of static heat transfer.
  %
  terminals:
    Heat_In  IS A Heat_Transfer_In;
    Heat_Out IS A Heat_Transfer_Out;
  parameters:
    k, Area IS A Parameter;
  realizations:
    Constant_Heat_Transfer IS A Primitive WITH
      equations:
        Heat_In.Flux = Area*k*
          (Heat_Out.Temp - Heat_In.Temp);
        Heat_Out.Flux = Heat_In.Flux;
    END;
END;

Mixer IS A Model WITH
  %
  % This model describe a blending process
  % of two inlets which are mixed to one outlet.
  %
  terminals:
    Inlet_1, Inlet_2 IS A In_Pipe;
    Outlet           IS A Out_Pipe;
  realizations:
    RealMixer IS A Structure WITH
      submodels:
        Mix IS A Multi_Flow_Element WITH
          realizations:
            Flow_Mix IS A Three_Flow_Join WITH
              Flow_3.Flow*Flow_3.Temp =
                Flow_1.Flow*Flow_1.Temp +
                Flow_2.Flow*Flow_2.Temp;
              Flow_3.Flow*Flow_3.Comp =
                Flow_1.Flow*Flow_1.Comp +
                Flow_2.Flow*Flow_2.Comp;
            END;
```

```
        END;
      connections:
        Inlet_1 AT Mix.Flow_1;
        Inlet_2 AT Mix.Flow_2;
        Outlet  AT Mix.Flow_3;
    END;
END;

Multi_Flow_Element IS A Model WITH
  %
  % This is a super-class for objects that
  % mix or split streams.
  %
  terminals:
    Flow_1, Flow_2, Flow_3 IS A In_Pipe;
  parameter:
    No_Comp := 5;
  constriants:
    Flow_1.Comp.Lenght :- Flow_2.Comp.Lenght :-
      Flow_3.Comp.Lenght :- No_Comp;
  realizations:
    Three_Flow_Join IS A Primitive WITH
      equations:
        Flow_2.Pressure = Flow_1.Pressure;
        Flow_3.Pressure = Flow_1.Pressure;
        Flow_1.Flow + Flow_2.Flow + Flow_3.Flow = 0;
    END;
END;

Servo_Motor IS A Model WITH
  %
  % A model of a servo motor.
  % It is used in the control valve model.
  %
  terminals:
    Control_Signal IS A SimpleTerminal;
    Position       IS A Position_Terminal:
  parameters:
    Km, Tm IS A Parameter;
  realizations:
    Motor_Dynamics IS A Primitive WITH
      variables:
        velocity TYPE Real;
```

```
      equations:
        Tm*DOT(velocity) + velocity = Km*Control_Signal;
        DOT(Position) = velocity;
    END;
END;

Spliter IS A Model WITH
  %
  % A model for describing a split of one stream
  % into two outlet streams.
  %
  terminals:
    Inlet               IS A In_Pipe;
    Outlet_1, Outlet_2 IS A Out_Pipe;
  parameters:
    No_Comp := 5;
  constraints:
    Inlet.Comp.Lenght :- Outlet_1.Comp.Lenght :-
      Outlet_2.Comp.Lenght :- No_Comp;
  realizations:
    RealSplite IS A Structure WITH
      submodels:
        Flow_Splite IS A Multi_Flow_Element WITH
          realizations:
            Flow_Splite IS A Three_Flow_Join WITH
              Flow_2.Temp = Flow_1.Temp;
              Flow_3.Temp = Flow_1.Temp;
              Flow_2.Comp = Flow_1.Comp;
              Flow_3.Comp = Flow_1.Comp;
            END;
        END;
      constraint:
        Flow_Splite.No_Comp := No_Comp;
      connections:
        Inlet    AT Flow_1 OF Spl;
        Outlet_1 AT Flow_2 OF Spl;
        Outlet_2 AT Flow_3 OF Spl;
    END;
END;

Static_Flow_Object IS A Model WITH
  %
  % A super-class for objects with static
```

```
  % mass, component, energy and momentum balances.
  %
  terminals:
    Inlet  IS A In_Pipe;
    Outlet IS A Out_Pipe;
  parameters:
    No_Comp := 5;
  constraints:
    Inlet.Comp.Lenght :- Outlet.Comp.Lenght :- NoComp;
  realizations:
    Statics IS A Primitive WITH
      variable:
        PressureDrop IS A Parameter;
      equations:
        Outlet.Flow  = Inlet.Flow;
        Outlet.Temp  = Inlet.Temp;
        Outlet.Comp  = Inlet.Comp;
        PressureDrop = Inlet.Pressure - Outlet.Pressure;
    END;
END;

Centrifugal_Pump IS A Static_Flow_Element WITH
  %
  % A model of a centrifugal pump with static behaviour.
  %
  parameters:
    pump_factor, Max_Flow IS A Parameter;
  realizations:
    PumpStatics IS A Statics WITH
      equations:
        Outlet.Flow = Max_Flow -
          (PressureDrop/pump_factor)^2;
    END;
END;

Compressor IS A Centrifugal_Pump;
  %
  % A compressor model which is a subclass of an
  % centrifugal pump without future specialization.
  %

Static_Valve IS A Static_Flow_Object WITH
  %
```

```
  % Model of a valve with static behaviour.
  %
  parameters:
    loss_factor, Cross_Area IS A Parameter;
  realizations:
    ValveStatics IS A Statics WITH
      equations:
        PressureDrop =
          loss_factor/(2*Cross_Area^2)*
          Outlet.Flow*ABS(Outlet.Flow);
    END;
END;

Static_Pipe IS A Static_Flow_Object WITH
  %
  % A model of a pipe with a plugg-flow.
  % The momentum balance is assumed to be
  % static. Composition and temperature
  % are delayed in the model.
  %
  parameters:
    Diameter, Length, loss_factor IS A Parameter;
  realizations:
    Statics IS A Primitive WITH
      variables:
        PressureDrop, CrossArea, DeadTime TYPE Real;
      equations:
        Outlet.Flow  = Inlet.Flow;
        Outlet.Temp  = DELAY(Inlet.Temp,DeadTime);
        Outlet.Comp  = DELAY(Inlet.Comp,DeadTime);
        PressureDrop = Inlet.Pressure - Outlet.Pressure;
        PressureDrop =
          loss_factor*Length/(Diameter*Cross_Area^2)*
          Outlet.Flow*ABS(Outlet.Flow);
        CrossArea = (Diameter/2)^2*PHI;
        DeadTime = Length/(Outlet.Flow/CrossArea);
    END;
END;

Valve_Lin IS A Static_Flow_Object WITH
  %
  % Valve_Lin is a model used in a control valve
  % configuration.
```

```
  %
  terminal:
    Position IS A Position_Terminal;
  parameter:
    Cross_Area, lin_factor IS A Parameter;
  realizations:
    LinValveChar IS A ValveStatics WITH
      equations:
        PressureDrop = lin_factor*Position/(2*Cross_Area^2)*
          Outlet.Flow*ABS(Outlet.Flow);
    END;
END;

Transform_Mass_Vol IS A Model WITH
  %
  % A useful model for transformation of
  % a mass fraction based terminal to a
  % concentration based terminal.
  %
  terminals:
    Mass IS A Liquid_In WITH Phase := nil; END;
    Vol  IS A Liquid_Out_Pipe WITH Phase := nil; END;
  parameters:
    No_Comp := 5;
    Mole_Weight TYPE Column No_Comp;
    dens        TYPE Real;
  constraints:
    Mass.Comp.Lenght :- Vol.Comp.Lenght :- No_Comp;
  realizations:
    r1 IS A Primitive WITH
      equations:
        Mass.Pressure = Vol.Pressure;
        Mass.Temperature = Vol.Temperature;
        dens*Mass.Comp = Mole_Weight .* Vol.Comp;
        Mass.Flow = dens*Vol.Flow;
    END;
END;

Vessel IS A Model WITH
  %
  % A super-class for vessel objects with
  % dynamic mass, composition or energy balances.
  %
```

```
  terminals:
    Inlet  IS A Process_In_Pipe;
    Outlet IS A Process_Out_Pipe;
  parameters:
    No_Comp IS A Parameter WITH value:=2 END;
    Phase   IS A Parameter;
  constraints:
    Inlet.Composition.Lenght :-
      Outlet.Composition.Lenght :- No_Comp;
    Inlet.Phase :- Outlet.Phase :- Phase;
END;

GasVessel IS A Vessel WITH
  %
  % A model of a gas vessel with dynamic
  % mass, composition and energy balances.
  % Static momentum balance.
  %
  parameters:
    Volume, GasConst, HeatCap IS A Parameter;
    MoleWeight                IS A Row No_Comp;
    Phase := "Vapour";
  realizations:
    Behaviour IS A Primitive WITH
      variables:
        composition IS A Column No_Comp;
        pressure, temperature, density, energy IS A Real;
      equations:
        pressure*Volume =
          GasConst*SUM(composition)*temperature
        Volume*DOT(composition) =
          Inlet.Flow*Inlet.Composition -
          Outlet.Flow*Outlet.Composition;
        density = MoleWeight*composition;
        DOT(energy) =
          Inlet.Flow*density*HeatCap*Inlet.Temperature -
          Outlet.Flow*density*HeatCap*Outlet.Temperature;
        energy = density*Volume*HeatCap*temperature;
        Inlet.Pressure = pressure;
        Outlet.Pressure = pressure;
        Outlet.Temperature = temperature;
        Outlet.Composition = composition;
    END;
```

```
END;

PerfectMixing IS A Vessel WITH
  %
  % A special model describing a fixed sized
  % gas element with dynamic properties.
  %
  terminal:
    Heat_Input IS A Heat_Transfer_In WITH
      Flux.default_value := 0
    END;
  parameters:
    Lenght, Cross_Area, Density, HeatCap IS A Parameter;
    Phase := "Liquid";
  realizations:
    Behaviour IS A Primitive WITH
      variables:
        totalcomposition, composition IS A Column No_Comp;
        volume, mass, energy, temperature,
          pressure IS A Real;
      equations:
        % Mass Balance
        volume = Cross_Area*Lenght;
        mass = Density*volume;
        % Component Mass Balance
        DOT(totalcomposition) =
          Inlet.Flow*Inlet.Composition -
          Outlet.Flow*Outlet.Composition;
        composition = totalcomposition/volume;
        % Energy Balance
        DOT(energy) =
          Inlet.Flow*Density*HeatCap*Inlet.Temperature -
          Outlet.Flow*Density*HeatCap*Outlet.Temperature +
          Heat_Input.Flux;
        energy = mass*HeatCap*temperature;
        % Static Momentum Balance
        Outlet.Flow = Inlet.Flow;
        Inlet.Pressure = pressure;
        Outlet.Pressure = pressure;
        Outlet.Temperature = temperature;
        Outlet.Composition = composition;
        Heat_Input.Temperature = temperature;
    END;
```

```
END;

Tank IS A Vessel WITH
  %
  % A model of a liquid tank with dynamic behaviour.
  %
  parameters:
    Cross_Area, density, HeatCap IS A Parameter;
    Phase := "Liquid";
  realizations:
    Behaviour IS A Primitive WITH
      variables:
        totalcomposition, composition IS A Column[No_Comp];
        height, Volume, mass, energy, temperature,
          pressure IS A Real;
      equations:
        % Mass Balance
        Cross_Area*DOT(height) = Inlet.Flow - Outlet.Flow;
        Volume = Cross_Area*height;
        mass = density*Volume;
        % Component Mass Balance
        DOT(totalcomposition)=Inlet.Flow*Inlet.Composition-
          Outlet.Flow*Outlet.Composition;
        composition = totalcomposition/Volume;
        % Energy Balance
        DOT(energy) =
          Inlet.Flow*density*HeatCap*Inlet.Temperature -
          Outlet.Flow*density*HeatCap*Outlet.Temperature;
        energy = mass*HeatCap*temperature;
        Inlet.Pressure = pressure;
        Outlet.Pressure = pressure;
        Outlet.Temperature = temperature;
        Outlet.Composition = composition;
    END;
END;
```

B5 Process Terminal Library

In this library commonly used terminal is found. They are first described in a terminal class hierarchy and after that the Omola description is shown.

Class Hierarchy

```
Terminal

  SimpleTerminal
     Concentration_Terminal
     Flow_Terminal
        Flow_In_Terminal
        Flow_Out_Terminal
     Heat_Data_Terminal
     Heat_Flux_Terminal
        Heat_Flux_In
        Heat_Flux_Out
     Mass_Flow_In_Terminal
        Mass_Flow_Out_Terminal
     Mass_Frac_Terminal
     Terminal_Tag
     Position_Terminal
     Pressure_Terminal
     Temperature_Terminal

  VectorTerminal
     Composition_Terminal
     Mass_Composition_Terminal
     Mole_Flux_Terminal

  RecordTerminal
     Flow_In_Pipe
        Heat_In_Pipe
        In_Pipe
           Process_In_Pipe
     Flow_Out_Pipe
        Heat_Out_Pipe
        Out_Pipe
           Process_Out_Pipe
     Heat_Transfer_In
```

```
      Heat_Transfer_Out
      Liquid_In
         Liquid_Out
            Vapour_Out
         Vapour_In
      Liquid_In_Pipe
         Liquid_Out_Pipe
            Vapour_Out_Pipe
         Vapour_In_Pipe
```

Simple terminals

```
Concentration_Terminal IS A SimpleTerminal WITH
  quantity  := mole_concentration;
  unit      := "kmole m^-3";
  direction := across;
END;

Flow_Terminal IS A SimpleTerminal WITH
  quantity := volumetric_flow;
  unit     := "m^3 s^-1";
END;

  Flow_In_Terminal IS A Flow_Terminal WITH
    direction := in;
  END;

  Flow_Out_Terminal IS A Flow_Terminal WITH
    direction := out;
  END;

Heat_Data_Terminal IS A SimpleTerminal WITH
  unit        := "kJ kg^-1";
  variability := bound;
END;

Heat_Flux_Terminal IS A SimpleTerminal WITH
  quantity  := heat_flux;
  unit      := "J s^-1 m^-2";
END;
```

```
  Heat_Flux_In IS A Heat_Flux_Terminal WITH
    direction := in;
  END;

  Heat_Flux_Out IS A Heat_Flux_Terminal WITH
    direction := out;
  END;

Mass_Flow_In_Terminal IS A SimpleTerminal WITH
  quantity  := mass_flow;
  unit      := "kg s^-1";
  direction := in;
END;

  Mass_Flow_Out_Terminal IS A Mass_Flow_In_Terminal WITH
    direction := out;
  END;

Mass_Frac_Terminal IS A SimpleTerminal WITH
  quantity  := mass_fraction;
  direction := across;
END;

Terminal_Tag IS A SimpleTerminal WITH
  value TYPE String;
  variablity := constant;
END;

Position_Terminal IS A SimpleTerminal WITH
  quantity := position;
  unit     := "m";
END;

Pressure_Terminal IS A SimpleTerminal WITH
  quantity  := pressure;
  unit      := "kPa";
  direction := across;
END;

Temperature_Terminal IS A SimpleTerminal WITH
  quantity  := temperature;
  unit      := "K";
  direction := across;
```

```
END;
```

Vector terminals

```
Composition_Terminal IS A VectorTerminal WITH
  Lenght := 4;
  CompType IS A Concentration_Terminal;
END;

Mass_Composition_Terminal IS A VectorTerminal WITH
  Lenght := 4;
  CompType IS A Mass_Frac_Terminal;
END;

Mole_Flux_Terminal IS A VectorTerminal WITH
  Lenght := 5;
  CompType IS A SimpleTerminal WITH
    quantity := mole_flux;
    unit     := "mole m^-2 s^-1";
  END;
END;
```

Record terminals

```
Flow_In_Pipe IS A RecordTerminal WITH
  components:
    Flow     IS A Flow_In_Terminal;
    Pressure IS A Pressure_Terminal;
END;

  Heat_In_Pipe IS A Flow_In_Pipe WITH
    components:
      Temp IS A Temperature_Terminal;
  END;

    In_Pipe IS A Flow_In_Pipe WITH
      components:
        Temp IS A Temperature_Terminal;
```

```
        Comp IS A Composition_Terminal;
    END;

      Process_In_Pipe IS A In_Pipe WITH
        component:
          Phase IS A Terminal_Tag;
      END;

Flow_Out_Pipe IS A RecordTerminal WITH
  components:
    Flow     IS A Flow_Out_Terminal;
    Pressure IS A Pressure_Terminal;
END;

  Heat_Out_Pipe IS A Flow_Out_Pipe WITH
    components:
      Temp IS A Temperature_Terminal;
  END;

    Out_Pipe IS A Flow_Out_Pipe WITH
      components:
        Temp IS A Temperature_Terminal;
        Comp IS A Composition_Terminal;
    END;

      Process_Out_Pipe IS A Out_Pipe WITH
        component:
          Phase IS A Terminal_Tag;
      END;

Heat_Transfer_In IS A RecordTerminal WITH
  components:
    Flux IS A Heat_Flux_In;
    Temp IS A Temperature_Terminal;
END;

Heat_Transfer_Out IS A RecordTerminal WITH
  components:
    Flux IS A Heat_Flux_Out;
    Temp IS A Temperature_Terminal;
END;

Liquid_In IS A RecordTerminal WITH
```

```
   components:
     Phase IS A Terminal_Tag WITH
       value := "Liquid";
     END;
     Pressure IS A Pressure_Terminal;
     Flow     IS A Mass_Flow_In_Terminal;
     Temp     IS A Temperature_Terminal;
     Comp     IS A Mass_Composition_Terminal;
END;

   Liquid_Out IS A Liquid_In WITH
     components:
       Flow IS A Flow_Out_Terminal;
   END;

     Vapour_Out IS A Liquid_Out WITH
         Phase := "Vapour";
     END;

   Vapour_In IS A Liquid_In WITH
       Phase := "Vapour";
   END;

Liquid_In_Pipe IS A Record_Terminal WITH
   components:
     Phase IS A Terminal_Tag WITH
       value := "Liquid";
     END;
     Pressure    IS A Pressure_Terminal;
     Flow        IS A Flow_In_Terminal;
     Temperature IS A Temperature_Terminal;
     Composition IS A Composition_Terminal;
END;

   Liquid_Out_Pipe IS A Liquid_In_Pipe WITH
     components:
       Flow IS A Flow_Out_Terminal;
   END;

     Vapour_Out_Pipe IS A Liquid_Out_Pipe WITH
         Phase := "Vapour";
     END;
```

```
Vapour_In_Pipe IS A Liquid_In_Pipe WITH
    Phase := "Vapour";
END;
```