Development and Application of a Library of Elementary Model Entities for Vapor-Liquid Chemical Processes

Dissertation

zur Erlangung des akademischen Grades

Doktoringenieur
(Dr.-Ing.)

von M.Sc. Odón de Jesús Angeles-Palacios
geb. am 31. Oktober 1966 in Omitlán de Juárez, Hidalgo, Mexiko

gemehmigt durch die Fakultät für Elektrotechnik und Informationstechnik
der Otto-von-Guericke-Universität Magdeburg

Gutachter:
Prof. Dr.-Ing. Achim Kienle
Prof. Dr.-Ing. Kai Sundmacher

Promotionskolloquium am 17. November 2005

Forschungsberichte aus dem Max-Planck-Institut
für Dynamik komplexer technischer Systeme

Band 13

Odón de Jesús Angeles Palacios

Development and Application of a Library of Elementary Model Entities for Vapor-Liquid Chemical Processes

Shaker Verlag
Aachen 2006

Bibliographic information published by Die Deutsche Bibliothek
Die Deutsche Bibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data is available in the internet at http://dnb.ddb.de.

Zugl.: Magdeburg, Univ., Diss., 2005

ISBN 3-8322-4765-3
ISSN 1439-4804

Shaker Verlag GmbH • P.O. BOX 101818 • D-52018 Aachen
Phone: 0049/2407/9596-0 • Telefax: 0049/2407/9596-9
Internet: www.shaker.de • eMail: info@shaker.de

Acknowledgments

I express my gratitude to the German people for giving me the opportunity of working in the Max-Planck-Institut für Dynamik komplexer technischer Systeme in Magdeburg under ideal scientific conditions. In the institute I met the most brilliant colleagues I have ever known and I was favored by stimulating discussions and warm friendship with some of them.

I would like to express my gratitude to Prof. Dr.-Ing. Dr.h.c. mult. E.D. Gilles, the founding director of the institute, for making possible for me to work under such ideal conditions.

I gratefully acknowledge Prof. Dr.-Ing. Achim Kienle for the enthusiastic discussions and supervising tasks during the development of this research project. I also wish to express my gratitude to Prof. Dr.-Ing. Kai Sundmacher for his accurate reading and reviewing of this work.

I wish to express my gratitude to Dr. Michael Mangold, Dr. Ulrike Krewer and Martin Ginkel for their comments and careful attention to detail. Their valuable contribution has enabled me to free the text from various errors and some inconsistencies, but the errors that remain are my sole responsibility.

For being always very near to me despite the long distance between Magdeburg and Mexico City, during my stay in Germany, I am in debt with Mr. Odón Angeles R. and Mrs. Marta Palacios D., my parents.

I express my gratitude to Mariana Domínguez, my wife, as well as Max and Ethel, my children for filling of sense my life the whole time.

For you:

My parents,
Mariana, Max and Ethel

Preface

This work was developed during my research fellowship in the Process Synthesis and Process Dynamics Group of the Max-Planck-Institut für Dynamik komplexer technischer Systeme in Magdeburg.

The development of a library for modeling vapor-liquid chemical processes is the subject of this work. Focus of this contribution is on lumped systems described by ordinary differential equations of first order and algebraic equations for vapor-liquid systems.

Modeling is presented as a common activity in different scientific disciplines, with special emphasis on modeling in chemical engineering. Domain engineering activities are discussed, namely analysis domain and architecture development. These activities lead us to the software pattern proposed in this thesis and applied for the implementation of the library. The discussion emphasizes that modeling chemical engineering systems by reusing modules of a library is different than modeling from scratch and that preparing modules for reuse is a process on its own. In this context it was shown how computer-aided modeling impacts the whole modeling task.

The formulation of theoretical aspects are applied here for the development of concrete modules, namely, the network approach, the proposed phenomenological framework, and the simple-to-complex implementation strategy.

The modeler of chemical process should find a conceptually well structured knowledge space in which entities can be systematically found, used, and further developed. For that, notational aspects for creating names are considered, a conceptual framework is developed to set phenomenological assumptions for the formulation of mass and energy balances, which find a very wide spectrum of application. This conceptual frame is proposed as a starting point for more detailed phenomenological developments of the library.

Although the library is a final product itself, its development would be pointless without applications associated with it. The applications consist of two different industrial chemical processes: the first one is a plant for the production of butyl acetate and the second one is a plant for the production of acetic acid. Simulation scenarios serve to show the response of the plants. Each application is a completely independent project with the library of models as a common root. It is shown that the library can be adapted to particular requirements of the modeler and it can be further developed by either creating new modules or by modifying some others. After a validation and reuse analysis, these new models can be added to the library to make them available for different users.

Odón Angeles Palacios

Freiburg, December 2005

Contents

Notation

c	Speed	m/s
c_p	Molar heat capacity	kJ/kmol
CO	Controller output	-
g	Gravitational acceleration	m/s^2
h	Molar enthalpy	kJ/kmol
H	Total enthalpy	kJ
$J[n]$	Molar flow	kmol/s
$J[q]$	Heat flow	kJ/s
n	Number of moles	kmol
P	Pressure	bar
P_d	Dew point pressure	bar
P_b	Bubble point pressure	bar
P^o	Vapor pressure	bar
PV	Process variable	-
R_g	Universal gas constant	kJ/(kmol K)
rhs_cmb	Right hand side of the component material balance	kmol/s
rhs_eb	Right hand side of the energy balance	kJ/s
rhs_tmb	Right hand side of the total material balance	kmol/s
T	Temperature	K
T_d	Dew point temperature	K
T_b	Bubble point temperature	K
t	Time	s
u	Molar internal energy	kJ/kmol
U	Total internal energy	kJ
V	Total volume	m^3
v	Molar volume	m^3/kmol
x	Liquid mole fraction	[-]

y	Vapor or gas mole fraction	[-]
z	Overall mole fraction	[-]
z	Elevation with respect to a reference level	m

Greek Symbols

γ	Activity coefficient	[-]
ρ	Molar density	$kmol/m^3$
η	Murphree efficiency	[-]
τ	Integral time	[-]
ψ	Vapor fraction	[-]

Subscripts

a	Outlet
e	Inlet
i	Component
g	Generation

Superscripts

nc	Number of components
0	Pure component
’	Liquid
”	Vapor

Chapter 1

Introduction

This work addresses the task of mathematical modeling of industrial chemical processes.

With the arrival of computer aided tools in many realms of human activity, the modeling task, a fundamental source of scientific knowledge, is now considered an activity suitable to be computer aided. It is expected that the contribution of a computer aided modeling tool enhances the capacity for developing models, storing and further reusing them. The process modeling tool, **ProMoT** (Tränkle, 2000), was used in this work for such purposes. Using the Model Definition Language (MDL) of ProMoT, a library of modules was developed and applied to model different industrial processes. The intention was not to model any kind of industrial chemical processes, since it would be a never ending task. The main objective was to set up the conceptual structure to manage the library, i.e. retrieving, storing, and further developing models, with a comprehensive application spectrum and with that, contribute to the systematization of the modeling task (Mangold et al., 2004).

ProMoT has been developed and applied in the recent years at the Institut für Systemdynamik und Regelungstechnik (ISR) at the University of Stuttgart and at the Max Planck Institute for Dynamics of Complex Technical Systems in Magdeburg, for modeling different chemical engineering processes. After a history of continuous development, ProMoT has become a robust tool for assisting the modeling task.

During the first stages of development of ProMoT, particular applications were implemented trying to systematize the modeling process. Once each application was finished, even though the main particular modeling objectives were achieved, it became clear that further efforts had to be done in the future towards systematizing the modeling task. In the first publications (see Tränkle et al. (1997b) for example), ProMoT was introduced as a knowledge-based modeling tool, emphasizing the importance of creation of hierarchies of structural model entities[1]. Based on this evolutionary transformation for conceiving the modeling process, we conclude that the modeling process is determined by three main factors, namely,

- the aims of the modeler,
- the software tools currently available, and
- the accumulated experience coming from previous generations of modelers.

Knowledge coming from chemical engineering will contribute to set up the model, while software engineering considerations will contribute to implement the model for its computational manipulation.

1.1 Why to develop a library for modeling chemical engineering processes?

Chemical engineering models have to be translated into a programming language. Hence, once the modeler has finished the modeling, he has to concentrate his attention on the computer implementation. The ideal scenario is that, in which the chemical engineer concentrates on understanding the phenomenological problem rather than on computational issues.

Applications of the modeling tool ProMoT, like that reported by Osigus (1998), made use of the ideas on modeling strategies. However, in practice, real implementation problems were found when a modeler tried to reuse and further develop modules created for a particular application. The creation of libraries of models was always the target, but serious difficulties were found while reusing these modules without affecting dependencies between modules when those modules were used in different applications.

Models implemented by an engineering team represent a valuable source of knowledge. Having access to this wealth is possible only if the information is systematically structured and properly prepared to be reused. This can be achieved through the implementation of libraries that collect knowledge in a systematic fashion.

[1]More about the parallel development of ProMoT and the evolution the modeling concepts can be followed in Tränkle et al. (1997a, 1999, 2000).

1.2 Challenges in knowledge processing

Knowledge based systems are more a promise than a reality. Such systems are supposed to manipulate (or to process) systematically stored pieces of information and produce knowledge after the application of logical operations. This task has been achieved only for very simplified scopes of important academic interest but is of limited application in practice.

Currently, engineers work with data bases but perform operations and take decisions considering their own practical experience and academic background. In other words, computer systems store the information, but the engineer interprets it and takes decisions, perhaps assisted by information systems.

Even though achievements of computer science continuously surprise the society, much more effort has to be done for processing knowledge and solve problems that emerge in the realm of chemical engineering. For example, given a complex mixture of hydrocarbons, what can be done to separate them to fulfill quality specifications? Though the engineer can be assisted by computer systems to solve the problem, design decisions and process modeling can currently be executed only by creative and experienced engineers.

The following challenges remain to be tackled concerning automatic knowledge processing:

1. **Lack of a suitable knowledge representation for computational processing.**

   If we look at a classical text book of chemical engineering, say for example, (McCabe et al., 1985), we cannot conclude that knowledge is not systematically stored. There is a conceptual coherence between the different subjects, however, the information is not represented in a computer language for solving specific engineering problems. Chemical engineers need to set mathematical models that describe specific physical situations and solve the resulting sets of equations numerically. This process requires data, heuristics, mathematical algorithms, and the implementation of all these requirements into a computer program to be accomplished. Currently the engineer has to dedicate much efforts to translate the physical description of the system (equations) into programming languages for its numerical treatment. It is not enough for the engineer to understand the physical problem, but he also has to struggle with computational issues.

2. **Only pieces of code are reused.**

The work dedicated to the implementation of models is currently difficult to be reused. Usually, the reusability limits to copying some previous implemented code and adapt it to new situations. This method is error prone, since very much attention has to be paid for making the suitable adaptations. Selected pieces of code that are to be reused have to be searched any time they are required, since they are not systematically stored and very frequently these pieces of code are not properly documented.

3. **Experienced engineers do not have a systematic way to transmit their knowledge to non experienced generations**.

   Traditionally, books are the best way to leave knowledge to new generations. Of course, books are usually well structured and make knowledge available, but writing a book is a documentation effort in itself. The experienced engineer has to go to a quite place and dedicate himself to document his experience. This documentation process is usually an expected product in the academy, but in the industrial practice, which is also a very rich source of experience and knowledge, time constraints have a very different meaning than in academic environments (Quibeldey-Cirkel, 1999).

   Unfortunately, the rapid changes of technology and economic trends make that knowledge stored in books be very frequently not updated.

4. **Lack of documentation or large amounts of it makes it difficult to reuse knowledge.**

   If some systematic storage of information is achieved and libraries of reusable modules are implemented, there exists a very close dependency between these modules and their developers that makes it difficult to reused knowledge.

1.3 Contributions of this work

Automatic knowledge processing requires that the information is represented properly. This representation exhibits a strong parallelism with the form in which engineers represent knowledge and allow them to take decisions for solving every day problems. Assisting the engineer with a computer system that allows to take better decisions paves the way for tackling the challenges posed in the last section. Simultaneously, these systems can be used for the engineer and take advantage of the representation of the information. This is what was done with the library proposed in this work. The following contributions are proposed for tackling the above described problems.

Problem: Lack of a suitable knowledge representation.
Contribution: Software engineering patterns are proposed.
The proposed software pattern guides the modeler in the implementation of modules, reducing to a minimum the necessity of taking software design decisions. The proposed pattern has a strong parallelism with the conceptual formulation of models in the realm of chemical engineering. The proposed pattern is discussed up to page 30 and illustrated in Fig. 3.5.

Problem: Only pieces of code are reused.
Contribution: Structural reuse of entities is achieved.
Structural storage of modules constitutes a documentation itself. Pieces of code are not implemented just when they are required, but systematically stored according to a conceptual guideline and functionality. For example, the engineer expects to find a particular correlation for the evaluation of a physical property– say for example the molar volume of a pure liquid species– in a well structured branch of the knowledge domain defined in this library (see Fig. 3.3).

Problem: Experienced engineers do not have a systematic way to transmit their knowledge to non experienced generations.
Contribution: The development methodology proposed in this work points out the importance of knowledge representation and the collective character that reusing knowledge implies. The proposed development sequence (illustrated in Fig. 3.1) assists the experienced engineer in preparing his knowledge for reuse and is a guideline for new generations that are faced with continuously growing amounts of new knowledge (see section 3.3).

Problem: Lack or large amounts of documentation make it difficult knowledge reuse.
Contribution: The contribution of this work consists, on the one hand, of developing a library of modules for modeling chemical engineering processes. On the other hand, the suitability of the model structure and the modules comprising the library itself were tested through different applications. Software design patterns were developed and applied as a general strategy for structuring the library. In each application it was shown how the modeling process was performed systematically (see the applications of the library in chapter five and also Waschler et al. (2003), Mangold et al. (2004), and Waschler et al. (2006)).

1.4 Overview

In the next chapter, theoretical aspects about the modeling task are considered. Modeling is presented as a common activity in different scientific disciplines, with special emphasis on modeling in chemical engineering. This chapter concludes with the presentation of some aspects taken from software engineering used for the implementation of model entities of the proposed library.

In chapter three domain engineering activities were discussed, namely analysis domain and architecture development. These activities lead us to the software pattern proposed in this thesis and applied for the implementation of the library. The discussion in this chapter emphasizes that modeling chemical engineering systems by reusing modules of a library is different than modeling from scratch and that preparing modules for reuse is a process on its own. In this context it was shown how computer-aided modeling impacts the whole modeling task.

Chapter four describes the most important entities of the library. Theoretical aspects developed in chapter three are applied here for the development of concrete modules. This chapter is not a user manual of the library. Such a manual should not be required if the structural discussion of chapter three is well formulated. The modeler of a chemical process should find a conceptually well structured knowledge space in which entities can be systematically found, used, and further developed. For that, notational aspects for creating names are considered, a conceptual framework is developed to set phenomenological assumptions for the formulation of mass and energy balances, which find a very wide spectrum of application. This conceptual frame is proposed as a starting point for more detailed phenomenological developments of the library. Although the library is a final product itself, its development would be pointless without applications associated with it. Chapter five deals with applications of the library. In this chapter it is not only shown *what* was done, but also *how* it was done. Each application is a completely independent project with the library of models as a common root. It is shown that, once loaded, the library can be adapted to particular requirements of the modeler and it can be further developed by either creating new modules or by modifying some others. After a validation and reuse analysis, these new models can be added to the library to make them available for different users. In chapter six the conclusions and outlook are presented.

Chapter 2

A perspective of computer-aided modeling in chemical engineering

In this work we consider modeling and simulation as two closely related, but well distinguishable activities.

In the context of this thesis, simulation consists of numerical solving of sets of equations that describe a physical system. Hence simulation implies the application of computational algorithms to mathematical models. Setting up these models belongs to the realm of modeling. Nowadays it is possible to distinguish clearly between modeling and simulation since computer tools have evolved to allow the engineer to concentrate on each one of these activities.

This chapter starts with a discussion about general considerations of mathematical modeling that are required for focusing on mathematical modeling of chemical engineering systems.

The importance of structuring the implementation of mathematical models will be set in the the following section, in which different structuring approaches are considered and shown how the structuring is strongly influenced by the programming language in which the mathematical description of physical systems are described.

2.1 A common basis for the formulation of mathematical models

According to Aris, a ’mathematical model’, or just a ’model’, is made up of ”any complete and consistent set of mathematical equations which is thought to correspond to some other entity, its prototype.” The prototype belongs to any part of the reality under study, it is a physical or natural object; the prototype, then, should be a nonmathematical system (Aris (1999), p. 3). The mathematical model implies a change on the scale of abstraction. Therefore, some particularities will be removed and simplifications will be made in obtaining the model.
Even though modeling has been referred to as an art (Basmadjian, 1999), a general strategy for setting up models shall be mentioned. For that purpose, system theory, which was originally proposed as a ”common abstract basis and unified conceptual framework for studying the behavior of various types and forms of systems” (Zadeh and Polak (1969), p. vii), offers a comprehensive guideline, which is not proposed to be followed sequentially. In this sense, given a physical situation, the following basic problems are found while modeling a system:

- Identification of the relevant physical attributes of which the system comprises. Here it is determined which first principles are involved in the description of a system and the variables that they comprise. These quantities are called *fundamental dependent variables* and its specification at any time and position will contain all the information that might be required to study particular aspects of the system behavior.

- Characterization of the relation between these physical attributes. In general these relations are set with mathematical expressions.

- Determination of relations between the variables varied by the experimenter, that is, inputs, and the variables which are observed but not varied directly, that is, the outputs (Zadeh and Desoer, 1979).

- Determination of the mathematical consistency of the model. That is, the equation system should not be over specified (more equations than variables) or under specified (more variables than equations).

Experienced modelers know that modeling is an iterative process, they are aware that revision of ideas and development of models is a continuous process. In this regard, progress is not necessarily in the direction of complexity or an increasing number of parameters. Simplification and reduction of the number of adjustable parameters are always desired. The iterative character of the modeling task is emphasized by Rutherford Aris, an eminent and experienced modeler, as shown in figure (2.1). The figure suggests that the last version of a model is just the current one, since revision of ideas and discussion are always expected.

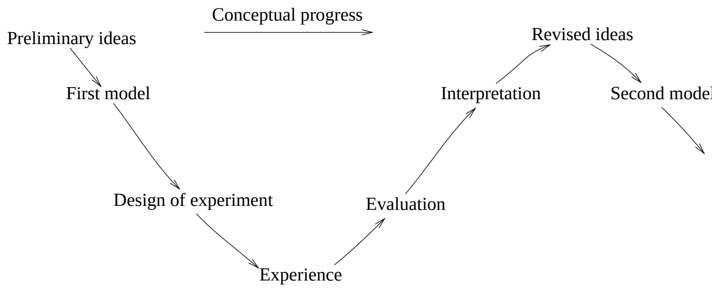


Figure 2.1: The conceptual progress envisioned by R. Aris. Adapted from (Aris 1994, p. 22.)

So far we have mentioned general concepts suitable for modeling systems, but the aim of the modeler deserves also to be considered. When modeling the behavior of a system, one of the following scenarios can be encountered, depending on the aims of the modeler (Cha et al. (2000), p.3):

a) Given the values of a set of independent variables, what are the values of the dependent variables?

b) In order to produce a desired set of dependent variables for a given set of independent variables, what changes must be made to the parameters of the system?

c) When some or all of the parameters of the system cannot be changed, what inputs must be applied in order to produce a desired set of dependent variables?

These three scenarios are also sketched in figure 2.2. In this thesis we are going to deal essentially with the first scenario. The later two items require us to solve the inverse problem, this type of problem is usually encountered in engineering design. However, it is desirable that models are flexible enough to be transformed from one of these scenarios into the others.

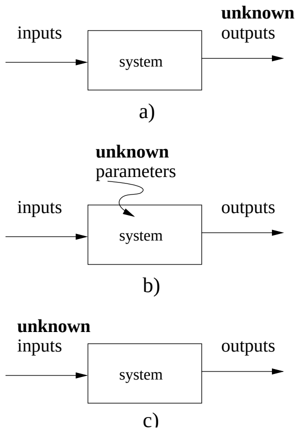


Figure 2.2: Three different scenarios according to the purposes of the modeler. Notice that a variable representing a quantity leaving the system, e.g. a flow rate of an outlet might be considered an input variable, that is, an independent variable. Scenarios a) and b) are usually considered to fall into the category of process analysis, while scenario c) is considered to be part of process synthesis.

2.2 Mathematical modeling of chemical engineering systems

The above discussion gathered ideas on modeling in general. Getting more into the point of this thesis, prototypes shall be concerned with chemical engineering systems. In this realm, models can be characterized by a well known structure of mathematical expressions, that can be grouped in three categories.

In this work we considered the so called *first principle models*. These models contain a universal law, which is usually expressed through a *conservation principle*, that is, equations that balance some quantity which enters or leaves the process being modeled. These balance equations are laws of nature and no exception of their applicability has been found. Mathematically they have the same form in most cases. For example, let F be the net flux of some property (say, mass or energy for example) into a spatial region known as *control volume*, G the total generation of the same property in this volume, then the total amount M, of this property contained in the control volume is

$$\frac{dM}{dt} = F + G. \tag{2.1}$$

In other words, balance equations comprise two contributions: the net rate of transport of the property into the control volume and the net generation of the property within the control volume.

The second kind of mathematical expressions found in models are the so called *constitutive relations*, they depend on the particular materials and circumstances in which they are applied. The validation of a model against the experiment is often largely a test of the adequacy of the constitutive equations, since the conservation principles are not in doubt (Denn (1986), p. 10). Typical examples of constitutive relations are correlations for the prediction of the saturation pressure of pure liquids; correlations for calculating the heat capacity of gases; chemical reaction rate expressions or equations of state coming from thermodynamics. Many constitutive relations are expressed usually as functions of temperature, since thermometry is a relative well known and accurate operation.

The third kind of expression required for the complete mathematical description of a prototype is the so called *operating mode* (Aris, 1993), which comprises operating constraints of the particular process. For example, the mass holdup of a system can be described by an empirical correlation which can include geometrical and hydrodynamic Al aspects.

Other types of equations like definitions or additional assumptions can fall into the categories of constitutive relations or operating constraints. For example, the summation condition for fractional expressions of the composition, like molar fractions $\sum_{i=1}^{nc} x_i = 1$ are result of the definition of these fractional variables. Another example are the so called mixing rules for calculating the value of a property in a mixture from the values of the pure components, like the expression for the heat capacity for a liquid mixture as the weight sum of the pure liquid property times its composition, $Cp_t = \sum_{i=1}^{nc} Cp_i x_i$.

In addition to the mathematical structuring of a model in the three categories discussed above, namely, conservation principles, constitutive relations and operating constrains; structural aspects from the software engineering point of view have to be considered in order to manage the complexity achieved on the side of computer tools and the required level of details in chemical processes.

In what follows structural considerations are taken into account when manipulating a chemical engineering model.

2.3 Knowledge representation and model structuring

Given its importance and difficulty, much effort has been done to improve the modeling process and it seems that research in this direction will continue in the future (Marquardt, 1996). Innovative design and operation of processes as well as their efficient operation require more detailed models. Consequently, computational tools used for that purpose evolve continously and also new challenges emerge as the diversity of operations performed by computer systems increases. Starting from a set of basic computer operations that include recording, storing, copying, moving, erasing and comparing symbols; computer tools have enhanced their capabilities for processing large quantities of information, that requires to be structured systematically if that information is going to be useful. The necessity to find a suitable structure for describing the portion of the reality under study becomes of fundamental importance, since the way in which the structure is described will critically determine how complex or simple the structure is. Herbert Simon aggregates in this regard the idea that follows:

> ”Most of the complex structures found in the world are enormously redundant, and we can use this redundancy to simplify their description. But to use it, to achieve the simplification, we must find the right representation.” (Simon (1996), p. 215.)

Identification of patterns in the composition of redundant structures leads to the simplification of its description. The set of conceptual entities defined for describing a particular scope of the reality constitutes a *knowledge space*[1], which should be structured in such

[1]Similar concepts regarding what we call here knowledge space can be found in the literature. For example, ”the universe of discourse” is used by Bayer et al. (2000) for referring to the knowledge

a way that the modeler can orient himself for performing operations with the entities that constitute the knowledge space without having much previous information about the allocation of the different information pieces. The creation of a knowledge space shall exhibit semantical relations and shall follow well defined practical conventions or rules, which facilitate surfing in the information without requiring a long previous experience. ”Surfing in the information” includes operations like retrieval of data, creation of new entities, storing them, etc. An example of the semantical representation of a distillation column in the knowledge space of separation processes is shown in Fig. 2.3. The entities

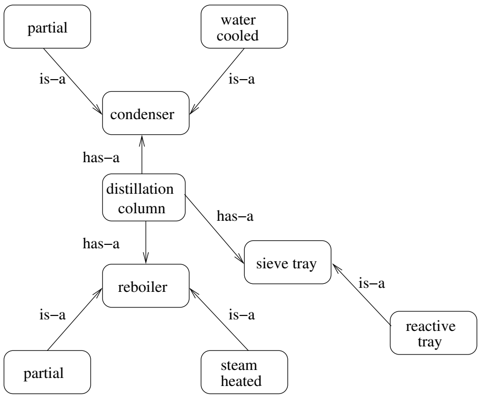


Figure 2.3: Graphical representation of a semantic network of a distillation column. Adapted from Quantrille and Liu (1991).

of this semantic network can be recognized immediately by a modeler in the chemical engineering field. In Fig. 2.3, relations between the entities are indicated through a text and arrows connecting the entities. It is also suggested how the knowledge structure of an specific domain can become more and more complicated if we consider for example

domain. The concept of *ontology* is also used in the literature for referring to the structuring and definition of the entities of a knowledge space, but the word ontology will not be used in this work though the concept is used implicitly. For our purposes, it is sufficient to mention that the concept of ontology is applied in computer sciences when dealing with the formal specification for representing objects, concepts and other entities that are assumed to exist in some area of interest (a knowledge space) and the relations that hold among them. The concept of ontology has become to be frequently used in the literature in the realm of chemical engineering. The reader is referred to Bogusch and Marquardt (1997) or Batres et al. (2002).

that a distillation column might consist of more than one side draw streams, liquid-liquid phase split, heterogeneously catalyzed chemical reactions, a complex control structure, etc. In general, knowledge is voluminous, it is hard to be characterized accurately, it is constantly changing and differs from data by being organized in a way that corresponds to the way it will be used (Quantrille and Liu, 1991).

The identification of structural patterns in a particular knowledge space should simplify its description, contribute to draw conclusions and solve problems.

All concerning the knowledge space that constitutes chemical engineering is result of a long development process that will be briefly surveyed in the following section.

2.3.1 A survey of structuring approaches

In the last century, chemical engineers based their practice on the concept of unit operation. First introduced in 1915, this concept was a cornerstone for representing chemical engineering processes during the last century (Li and Kraslawski, 2003). As usual, subdivision of a problem into subproblems was followed for defining an operation. Although the number of individual processes was great, each one was broken down into a series of subprocesses called operations, each of which appeared in different processes. It was noted that individual operations had common techniques and were based on the same scientific principles. Materials are carried from one phase to another and different forms of energy must be transferred from one substance to another, and tasks like drying, distillation, and evaporation must be performed (McCabe et al. (1985), p.3).

Knowledge representation as well as its implementation into programming languages have been evolved in a parallel and continuous fashion.[2] Progress on computer sciences allows and fosters dynamic research on more sophisticated representation approaches. Artificial intelligence emerges as a promise for mimic human knowledge processing. It is recognized that knowledge differs from data or information. Knowledge is information plus processing, and a language for representing knowledge is required, as well as a computer program that executes the processing of this knowledge (Polke (1994), p. 1-10). Much effort has been oriented for developing the so called *knowledge-based systems* with the intention of systematizing knowledge processing. These systems have been proposed in many fields and chemical engineering is not the exception.

In the early 1980s, *computer aided process engineering* emerged as a promise for all pro-

[2]In section 2.4, the evolution of computer implementation approaches is considered.

cess engineering activities (Sargent, 2004). First ideas introduced by theorists of artificial intelligence found a natural and promising application in process engineering. A fundamental and pioneering contribution, the environment DESIGN-KIT, was presented by Stephanopoulos et al. (1987). The proposed computer environment was supposed to cover a wide spectrum of activities: flowsheeting, equipment design, optimization, project engineering, planing, costing, drafting, and documentation. Knowledge-based systems were then introduced as a suitable solution for processing large amounts of knowledge required for the activities of the chemical engineer. But so far this is more an appealing intention than a concrete realization. Many authors claim to have proposed knowledge-based systems (see for example, Bogusch and Marquardt (1997), Kurzok et al. (2001), Hangos and Cameron (2001), or Shah and Kokossis (2001)). However, even though great achievements have been attained regarding knowledge representation, knowledge processing remains to be a promise. A knowledge-based system should provide both, independent reasoning tools and a sound representation of knowledge that allows such processing (Marquardt, 1996).

Knowledge processing leads to what has been called *expert systems*. Such computer systems comprise two basic parts. First, a data-base, also called *knowledge-data-base,* which represents the knowledge of a human specialist, called the expert. The second part of an expert system is a program that processes the information contained in the data-base in order to emulate operations originally performed by the human expert like, for example, responding to situations very flexibly; making sense out of ambiguous or contradictory information; recognizing the relative importance of different elements within a situation; finding similarities despite differences, and drawing distinctions despite similarities among various situations (Stephanopoulos and Mavrovouniotis, 1988). Given the complexity of the required data-bases and the difficulty to represent knowledge for its manipulation, the initial enthusiasm shown by the first proposals has evolved to more concrete goals, leaving the processing of information to be carried out by the engineer who will be assisted by well structured data-bases.

The well known difficulty for developing models seems to justify further efforts regarding representation and structuring of knowledge. However, new challenges emerge. The size of data bases increases rapidly for example, and it is desirable to reuse entities or models in later projects. Reusing, updating and documentation of models seem to be difficult and error prone tasks. New programming languages and notations have been developed in order to allow the modeler to represent chemical engineering knowledge.

In the middle of these necessities emerges a representation proposal that conceives the structuring of knowledge as an interconnected network of elementary entities. Indeed, Gilles in his network approach for modeling chemical processes points out the importance of reducing the number of elementary entities that describe a chemical process, aiming to systematize its structure. Elementary entities, namely, *components* and *couplings,* do not correspond to a particular physical entity, but their definitions encompass a wide diversity of physical situations. The simplicity of these ideas is where their strength is found. Gilles noted on the one hand that in general terms processes can be characterized by entities that take into account conservative properties (matter, energy, momentum or charge [3]), that is to say, that property balances are expected to be found in the so called storages, which play the role of containers of the properties to be balanced.

On the other hand, the state of a process is changed by driving forces that generate potential differences between the properties that define the state of two storages. Magnitudes of these driving forces are determined by the second type of entities proposed by Gilles, that is, the coupling entities that link storages. With only these two kinds of entities, interconnected networks can be built, with storages that comprise dynamic variables, related through differential equations and with couplings that comprise algebraic expressions for determining fluxes between components. It is proposed that phenomenological hierarchies should be built with only these two elementary entities. Hence, Gilles proposes a structuring pattern which consists of storages and couplings.

These ideas result to be appealing, for they comprehend two kinds of mathematical expressions discussed from page 10 on, which characterize chemical engineering models. The attractiveness of this simplicity is so strong, that many authors have proposed similar ideas. The equivalence between the different proposed terminologies is compared by Mangold et al. (2002) in their extension of the application of these structuring ideas to spatial distributed systems and particulate processes.

The network approach described in the above paragraphs was proposed independently of any programming language or computer environment. However, its appearance triggered the development of a language that exhibits a particular expressiveness for representing networks of storages and couplings. The proposed language was called Model Definition Language (MDL), and the associated modeling tool was called ProMoT (Tränkle, 2000). The characteristics that make this language and modeling tool eligible for modeling chemical processes are discussed in the next section. But before getting into details of

[3]Also the volume can be considered a conservative property, some authors consider, ”volume balances” for example, see Hangos and Cameron (2001).

the implementation of models using MDL and ProMoT, general aspects about computer-aided modeling are considered, as well as new problems and challenges derived by the use of computer tools are described in more detail.

2.4 Computer-aided modeling

Traditionally, modeling and simulation were conceived as coupled activities, since modeling was performed directly in the programming language used to solve the equations that comprised the model. The modeler was then impelled to think in terms of algorithms when setting up a model. Currently, computational tools allow to decouple modeling and simulation. The first task specializes in pure phenomenological considerations for describing a system, while simulation deals with algorithms for numerical treatment. One of the main theses of this work is to emphasize the importance of the computer implementation of mathematical models showing that this implementation is a valuable product itself. To set the conceptual frame of the most recent developments in computer aided modeling and to conceive the implementation of a mathematical model as an additional product of the modeling task, a brief description of object oriented terminology is given in what follows.

2.4.1 The object-oriented implementation approach

The arrival of *object-oriented programming* brings into the picture a new generation of computer languages, the so called *descriptive languages*, which can in principle be used as imperative languages, i.e. for performing algorithmic tasks, but their most valuable advantage consists of allowing the representation of different relations among groups of objects of the same kind. In this sense, a group of similar objects constitutes a *class* (Dattatri (1997), p. 5). Examples of these relations are illustrated in Fig. 2.3 in which two kinds of relations are shown. On one side, starting from a general definition to a more specific one, objects are related through the so called *"is-a"* relation.[4] On the other side, aggregation relations, designated as *"has-a"* relations, are naturally represented with object oriented languages. In Fig. 2.3, a distillation column is represented as an aggregate of objects of different nature: a condenser, a reboiler and distillation trays. The attributes

[4]The "is-a" relation is also known as an *inheritance* relation. In Fig. 2.3 a partial reboiler "is-a" special kind of reboiler, that is, the partial reboiler inherits the essential attributes of a reboiler and adds more detail.

– variables and equations – of objects that comprise a distillation column are then organized according to well-defined relations, which have a strong parallelism with objects in reality, as illustrated in figure 2.3 for the case of a distillation column. Moreover, it should be mentioned that attributes are encapsulated inside each object allowing the possibility to manipulate each entity independently from the others. *Encapsulation* of data is, hence, another valuable characteristic of object-oriented languages.

It is important to emphasize, that the above mentioned features of the object-oriented approach do not describe algorithmic procedures. That is why these languages are designated as descriptive (or non-imperative) languages. That is why chemical engineers, who used to implement models in algorithmic languages, find it difficult at the beginning to use of object-oriented languages.

2.4.2 Modeling and simulation tools

So far we have discussed different knowledge representation approaches, as well as the object-oriented programming approach. With these surveys we are very far from having a comprehensive overview of the efforts that have been dedicated to develop computer tools to support engineering activities. A very intense development process of such tools can be followed when knowledge representation techniques are put together with their corresponding implementation in computer languages. The development process gave rise to the different computer tools that have appeared in the recent years. In the literature is found copious documentation that describes and compares features of different computer tools appeared during the last decades.[5]

From the diversity of computer tools we need to highlight two features that can help to group them[6]:

- *Block-oriented systems* are characterized by being composed of standard modules which contain data and/or algorithms describing the particular behavior of each block. These blocks are linked by connections that represent flow of information, mass or energy. Once defined, the structure and the incorporated knowledge of a block are fixed and not accessible.

[5] See for example Stephanopoulos and Han (1996), Otter (1999), Tränkle (2000), Marquardt et al. (2000), Schneider and Marquardt (2002) or Sargent (2004).

[6] Bogusch (2001) (p.28), presents a comparison between different computer tools and their classification.

- *Equation-oriented systems* employ declarative languages for describing relations between modules defined through data and equations expressed symbolically. In a subsequent step, a compiler generates an overall system of equations for its numerical treatment.

In contrast, the equation-oriented approach, with the use of declarative languages, allows the separation of the description of the physical system from the mathematical solution method. General-purpose process modeling tools based on the equation oriented approach are now widely used in the industry.

The modeling/simulation environment ProMoT/DIVA

The modeling/simulation environment ProMoT/DIVA is an equation oriented system that shares the advantages of modularization of entities, this feature resembles the block oriented approach, but modules are not closed and fixed black boxes. On the contrary, modules can be adapted for very particular necessities.

Thought as a tool for dynamic simulation, DIVA allows the systematic application of a sound library of numerical methods. Since the appearance of the first publication (Holl et al., 1988), DIVA has been used by generations of researchers and graduated students for the simulation of chemical processes and it counts now more that fifteen years of non-interrupted development (Mangold et al., 2000). For numerical treatment, models must be formulated in DIVA as differential algebraic equation systems in a linear implicit form , consisting of a vector of variables and parameters, **x** and **p**, respectively:

$$\mathbf{B}(\mathbf{x},\mathbf{p})\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x},\mathbf{p}) \qquad \text{with} \qquad \mathbf{x} \in \mathbb{R}^n \qquad \text{and} \qquad \mathbf{p} \in \mathbb{R}^p. \tag{2.2}$$

From the very beginning, even though the available algorithms in DIVA proved to be robust for the simulation task, the implementation of models in FORTRAN code became to be ”cumbersome” for the user and a code generator was developed to lighten the implementation of models (Köhler et al., 1997). After the introduction of the code generator, implementation via FORTRAN code was abandoned. However, the code generator required a relatively long learning time for getting the best from it. Three facts, then, served as precursors for the development of ProMoT as an extension of DIVA and the code generator:

- the coding could be significantly lightened by the introduction of the code generator;

- the emergence of object oriented technologies and its declarative languages;
- the structuring ideas proposed by the network approach of Gilles.

Nowadays, the ProMoT/DIVA modeling and simulation environment has become a general-purpose package.

Thanks to the model definition language (MDL), ProMoT allows the definition of model entities in a declarative fashion making ProMoT an equation-oriented system. Moreover, ProMoT also permits the modularization of models that can be linked through properly defined terminals. Interconnected modules are subsequently compiled to generate a unique system of equations for its numerical solution in DIVA. This approach allows to build libraries of models which share the advantages of both, the equation and block oriented approaches.

After what was discussed in this chapter, the modeling task can be divided in two parts, namely the conceptual formulation, that corresponds to the physical description of the system in mathematical terms, and its implementation in a programming language. As illustrated in Fig. 2.4, even though both parts are interwoven, distinguishing between them is important. It allows us to develop strategies to tackle difficulties when dealing with the physical description of a system and its translation in computational terms. *Conceptual*

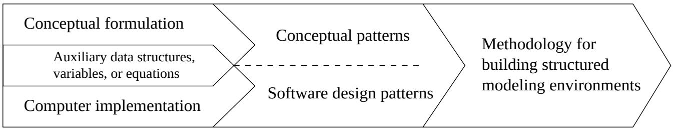

Conceptual formulation
Auxiliary data structures, variables, or equations
Computer implementation
Conceptual patterns
Software design patterns
Methodology for building structured modeling environments

Figure 2.4: Finding conceptual and software design patterns are interwoven when modeling with computer-aided tools. The dashed line in the middle emphasizes this fact.

structures can be found when describing a well delimited portion of the physical reality. In this sense, the conceptual description of a system can be arranged, as discussed previously (page 10) in first principles, constitutive relations, and operation constraints. The network approach of Gilles can also be considered as a conceptual structuring approach, since first principles, i.e., balance equations, are gathered together in storages, while constitutive relations and operation constraints define couplings. Since the conceptual structure captures the physical description of a system, it is independent of any computer language or tool.

When performing the implementation of a system into a programming language, architectural structures or recurrent data types can be identified and patterns can be defined. If recurrent implementation structures are found, *software design patterns* can be defined and software decisions are facilitated consequently. These recurrent structures may also be independent of the programming language, but it has to be admitted that software design is finally strongly influenced by the programming language.[7]

In both of the above mentioned kinds of tasks, patterns can be found, giving rise to *conceptual patterns* and *software design patterns* respectively. The identification of such patterns leads to enhance the scope of reuse. Not only pieces of source code of a programming language are reusable, but structural modeling patterns or architectures are repeatedly used in different circumstances.

Being aware of these two kinds of patterns constitutes a very useful guideline for accomplishing the formulation and implementation of new models as well as for manipulating existing modules in the library proposed in this work.

2.5 Summary

The mathematical solution and analysis of complex systems of equations constituted the main motivation for early computer tools. As computer power increased, new goals were conceived and the modeling task was considered to be also computer-aided. The current state of computer-aided process engineering can not be understood without having a perspective of this long way. Consequently, improvements will be fostered being aware of previous experiences and proposals. Despite the great efforts, knowledge-based systems remain a non-achievable goal if we consider that knowledge-based systems should perform independent reasoning operations. Undoubtedly, sound forms of knowledge representation will be required for developing such reasoning tools. Hence, the problem has to be defined more precisely. The implementation of knowledge processing tools will come in the future, but first, knowledge representation efforts should continue. New proposals in this direction should be tested by users of knowledge, i.e., modelers or chemical engineers, in our case. The contribution of this work emphasizes this point, proposing a structuring approach and its implementation in a computer language. The existence of a

[7]First ideas about pattern identification were brought into the software engineering literature for the contribution of Gamma et al. (2002).

modeling tool supports many operations that can be performed for manipulating knowledge in form of chemical process models. Such operations can consist of creating, modifying or reusing entities present in the knowledge space. The structure of a knowledge space depends on the one hand on the experience, knowledge background and aims of the modeler. On the other hand, the language used for representing knowledge plays also a fundamental role.

It is expected that contributions of computer-aided modeling allow the engineer to concentrate on those aspects that characterize the physical description of processes. Even though modeling tools support the work of the engineer, the modeler will not be substituted for any computer system.

ProMoT is the modeling tool used for our purposes and MDL the language used for representing knowledge. In the next chapter, more about ProMoT and the library built in this thesis will be considered.

Chapter 3

The construction of a library for modeling chemical engineering processes

With the continuously increasing importance of computational capabilities and with the complexity of mathematical models also growing, the implementation of process engineering models has consequently become much more important and also has become to be considered a valuable subproduct of the modeling task. Software engineering provides the chemical engineer with a systematic approach to the analysis, design, implementation and maintenance of large software projects. In this chapter software engineering considerations are brought into the realm of chemical engineering and are applied to build the library proposed in this thesis.

Arguments are presented regarding how modeling chemical processes from reusable entities differs from modeling from scratch and methodological considerations are discussed.

We take advantage of combining the object-oriented programming paradigm with the equation-based approach. As a result, a class comprises a set of equations which corresponds to real-world entities that can be manipulated as modules. The final result of coupling these modular entities is again a set of equations, but on its formulation we were assisted by graphical tools and text editors.

3.1 The process of creating reusable modules

Model reuse represents a further step in the working methodology within a technological community, say the engineering department of an enterprise or an academic research institution. In the industry, however, productivity criteria make it difficult to set up a project which solely consists of organizing previous developments for future reuse. This requires investment of time, and of human and technical resources. Hence, it is natural that first attempts for building libraries for supporting the modeling task have raised initially in academic environments. However, it seems that in the industry the tendency moves gradually towards developing libraries for supporting the modeling task, when models are recognized as a valuable asset of the enterprise (Eggersmann et al., 2002).

When attempting to build a library, some additional efforts will be required from the members of a modeling community, since operations that do not directly influence concrete and current problems have to be performed iteratively. For example, basic architectural decisions are to be taken or naming conventions should be established. In this context, the modeling task has been changing during recent years. Developing models of industrial chemical processes from reusable modules is different from developing models from scratch. Models have to be retrieved, evaluated, and adapted for reuse. In this sense, reusing models impacts the whole modeling process. Consequently, creating reusable modules should be considered as a process on its own. It is a long run project whose benefits are not going to be seen immediately.

Structural and methodological considerations are proposed in this thesis regarding the construction of the library presented in this work. Focus is not so much on technology, i.e., special computer tools, but on creating a culture to document and support sound engineering and design decisions that contribute to create a set of conventions that systematize and improve the modeling task of process engineers.

3.2 The basic software engineering conceptual frame for building a model library

In general computing terms, a *library* is a collection of programs or subroutines stored in one or more files and available for immediate use. In our case, the library does not consist of programs, but of pieces of code implemented in a declarative language (the MDL) that will be compiled in a subsequent step for its transformation into a procedural language.

The activities for identifying the relevant characteristics of entities present in the library with a strong emphasis on reuse are comprised in the so called *domain engineering*. These activities include, first, to define the application scope of the library, that is the *domain definition*. Secondly, the *domain analysis*, is the process of identifying, grouping, and representing the relevant information in a domain based on knowledge captured by domain experts or previous modelers. Domain analysis gathers information about software components, focusing on reusability. If a modeling entity corresponds to a concept of the physical reality, a very useful semantical parallelism between the physical system and its computer representation will be created. Models that have been prepared for reuse are usually more complicated than the original implementation used in a particular case. That is a normal situation, but a compromise between reusability and easy use of a module should always be pursued. Third, the *domain implementation* consists of codifying and documenting the components in a programming language. Although the library itself is a final product, its development would be useless without applications associated with it. *Application engineering*, that is, the use of modules of the library in particular cases is, then, supported by domain engineering activities. The new shape of the modeling task when transformed into computer-aided modeling is illustrated in Fig. 3.1, which shows the activities that characterize domain engineering and those related with application engineering.[1] The library that constitutes this work was built according to the domain engineering sequence shown in the upper part of Fig. 3.1.

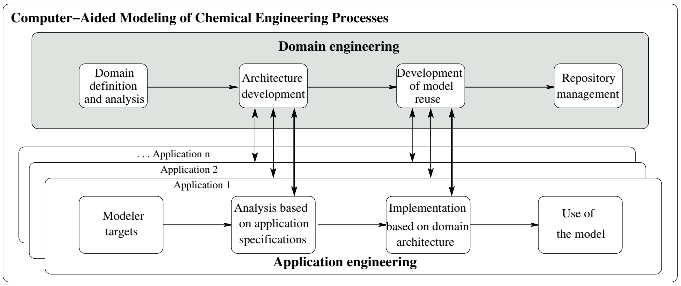
In what follows, results of domain engineering activities performed in this work will be described. The application engineering sequence of activities was followed for building particular applications of the library and is the subject of chapter five.

3.3 Domain engineering

An efficient usage and improvement of a library for supporting the modeling activity assumes a culture that embraces domain engineering, process engineering and human aspects between the members of the engineering community.

In the following discussion, the different activities of the domain engineering, shown in Fig. 3.1, performed for developing the library of the thesis are considered.

[1]Fig. 3.1 was adapted for modeling chemical engineering processes from Foreman (1996).



Computer–Aided Modeling of Chemical Engineering Processes
Domain engineering
Domain definition and analysis
Architecture development
Development of model reuse
Repository management
. . . Application n
Application 2
Application 1
Modeler targets
Analysis based on application specifications
Implementation based on domain architecture
Use of the model
Application engineering

Figure 3.1: Modeling of chemical engineering processes from the perspective of reusing modules in a library. Two interrelated engineering processes are illustrated: domain engineering and application engineering.

3.3.1 Domain definition

Modeling processes in which chemical transformation and/or separation of systems involving vapor/liquid phases should be covered by the proposed library. An important feature that characterizes modeling of such systems is, that it proceeds through a series of successively more detailed synthesis and evaluation stages. These successive refinements can vary from very simple and rapid, but not very accurate, estimates to very detailed calculations being as accurate as possible.[2] The library is focused on modeling processes in which vapor/liquid systems are involved.

Two main modeling approaches are to be implemented. On one side equilibrium models describe systems in which transfer resistances of mass, energy or momentum between phases are considered to be negligible. Hence, in equilibrium models state variables of phases are algebraically coupled by the thermodynamic equilibrium conditions. While, on the other side, transfer resistances are taken into account modeling non-equilibrium systems.

3.3.2 Domain analysis

An excellent guideline for performing the domain analysis of this library was provided by the network approach of Gilles introduced in the last chapter. Holdups of process entities

[2]Douglas (1988) (p. 7.) refers to this approximation approach as the ”engineering method.”

constitute components in which mass and energy are balanced. These process entities interact with the environment through the second kind of elements of the network approach, i.e. couplings that describe gradients between the properties balanced in components. This approach is particularly useful when modeling non-equilibrium systems, e.g. mass or energy transfer from a solid to a fluid phase. Fig. 3.2 captures the basic entities proposed in

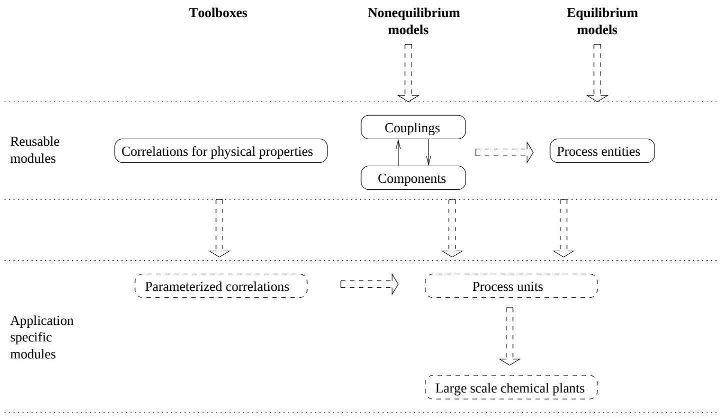

Toolboxes
Nonequilibrium models
Equilibrium models
Reusable modules
Correlations for physical properties
Couplings
Components
Process entities
Parameterized correlations
Process units
Application specific modules
Large scale chemical plants

Figure 3.2: Domain analysis diagram. Modules with solid frames are reusable entities that are stored in a more general library. Modules with a dashed frame are stored in a library with a specific application scope. Thick arrows represent implementation steps.

this work that result from the domain analysis for modeling chemical systems. Equilibrium models are widely used for different modeling purposes and are hence included as an important branch of application. Depending on the degree of the required detail of a model, process entities are described with components and couplings or as phenomenological entities in thermodynamic equilibrium.

Departure from the network approach

As shown in Fig. 3.2, process entities are reusable modules and process units are application specific modules. As mentioned above, a process unit is a parameterized module. An example of a process entity is a decanter in which two unmiscible liquid phases interact. If a finite mass transfer resistance is going to be taken into account, independent material balances should be set for each liquid phase and an interface between them is used for describing the magnitude of the mass transfer resistance. If no such detailed description is required, an equilibrium model captures the required description in a process entity that comprises balance and equilibrium equations. Clearly, the implementation of balance equations and equilibrium relations departs us from the network approach, since relations different from balance equations should not be implemented in a container or storage.

Proposed independently of any computational environment, the network approach remains in this work as a first proposal in the implementation of process entities. However, during the development of this library some flexibility in the application of the network approach was advisable when equilibrium models were implemented, favoring with that the reusability of modules. Many small classes, with just a few lines of code might result a very well structured implementation, but, according to our experience, difficult to use, to document and further to develop.

Going back to the domain analysis diagram of Fig. 3.2, a process entity is, however, not yet a self-contained model. This is because process entities are not application specific. Both, non equilibrium and equilibrium models are modularized and composed of reusable entities which require data of the particular system, it means that, these entities require to be parameterized. [3] A process entity becomes to be a process unit when the required data and parameters are provided. Even though a process unit can also be reused, given its modularized character, the reusability is constrained to the specific application. For example, in the same industrial application more than one condenser of the same type may be required, or very similar reactors with just a different residence time may be part of a chemical process.

Toolboxes

Models will require data and parameters for evaluating physical properties of materials under different conditions. Frames under the title ”Toolboxes” in Fig. 3.2 show that correlations can be also implemented to be reused and parameterized for a particular application. The evaluation of physical properties is carried out by theoretical models or empirical correlations, which have to be implemented and organized in a systematic fashion for facilitating its use. Fig. 3.3 shows the semantic organization of correlations as proposed in the library of this thesis. Calculation of physical properties is performed in-

[3]See pages 50 to 53 for the explanation of how to parameterize a process entity that represents a total condenser.

Thermodynamics

- Molar enthalpy
  - Gas
  - Liquid
  - Vaporization
- Vapor−liquid equilibrium
  - Coefficients
    - Activity
      - Ideal solution
      - UNIQUAC
      - Wilson
    - Fugacity
      - Redlich−Kwong

Volumetric

- Molar volume
  - Gas
    - Equations of state
      - Ideal gas
      - Redlich−Kwong
  - Liquid
    - Cavett
    - Rackett
- Saturation pressure
  - Antoine
  - Wagner

Phenomenological coefficients

- Mass transfer
  - Diffusivity
  - Mass transfer coefficient
- Heat transfer
  - Thermal conductivity
  - Heat transfer coefficient
- Momentum transfer
  - Viscosity

Reaction kinetics

- Elementary rate laws
  - First order
  - Second order
- Nonelementary rate laws
  - Hartig−Regner
  - HAc−reaction rate

Figure 3.3: Semantic structure of correlations for the evaluation of physical properties often used for modeling chemical processes.

tensively during the simulation and consumes important space in the working memory of simulation programs. It is therefore important to save computing resources devoted for evaluating properties. In the proposed library, modules for evaluating physical properties are completely reusable and have a limited scope, that means that a process entity requires to be provided with data and correlations that will be required for solving the equation set that defines that entity, hence modules for the evaluation of physical properties will have a local applicability. The same correlation can be used many times in other entities with the same or a different set of parameters. This resembles the implementation of parameterized routines in imperative programming languages, but takes advantage of object-oriented programming which is based on declarative languages. Modules for evaluating physical properties contain data that are available only in the context of that particular calculation. From the object oriented point of view this is a good example of *encapsulation*. If an encapsulated module is used repeatedly at different points in a model, changes are propagated only by modifying data definition modules. This means for example, that a chemical plant can be structurally reused as a whole, but with different chemical components, represented by data sets used for the calculation of the physical properties of the chemical species processed in the plant. Changing from one set of chemical compounds to another should have to be done only once and all modules would generalize the changes for a new chemical system. For example, the saturation pressure might be evaluated in the reaction system of a chemical plant, as well as in different parts of the separation system, i.e., distillation columns, condensers, reboilers, flash tanks. The plant can exist in reality and its functionality might be required to be tested in case a different chemical system is planned to be processed in the plant. The only required change to be done for a new set of chemicals is performed in those classes that define parameters for the former set. Obviously, there might have to be taken into account particular physicochemical conditions of the system and it should be ensured that the same assumptions will remain valid. For example, the assumed non-mixing effects of solutions or the ideal gas model assumption might not be valid for the new mixture.

Correlations and data modules are organized as described in Fig. 3.3. This allows to manipulate them in a polymorphic way. *Polymorphism* is an object-oriented feature that can be illustrated by the use of different modules for evaluating a physical property. Let's take again the case of the saturation pressure. The evaluation of this property is usually performed with a three parameter, empirical correlation – the Antoine equation for example. Let's suppose that for the purposes of the modeler a five-parameter correlation for the saturation pressure is required in one of the process units. Hence, only in the parameterized process unit the new correlation has to be substituted without having to change the equations that define the unit. The name of the physical property remains unchanged – saturation pressure in the example – but its value is calculated in a different form, that is why this feature is called polymorphism.

These ideas will be clearer, once the software architecture of the library is described in what follows.

3.3.3 Architecture development

In order to facilitate reuse, further development and systematic storage of modules in a library, it is strongly desirable that the library be structured according to a well defined methodology. The role of pattern identification was emphasized when revising different model representation approaches and Fig. 2.4 points out pattern identification during the conceptual formulation of a model and during its implementation in computer terms.

As emphasized in the previous chapter, a suitable representation allows us to identify structural patterns for the systematic description of our working domain. The network approach of Gilles proposes a guideline that classifies conceptual entities either as parts of components or as parts of couplings (see page 15). This guideline was followed during the conceptual formulation of the modules of this library and can be considered as the definition of a conceptual pattern. However, as discussed above, practical implementation decisions were taken to favor the reusability of modules in the scope of the applications of the proposed library. This pattern – as usually occurs when patterns are identified – was found after the formulation of different chemical processes and the associated representation language consisted of equations that were organized in different categories, grouped subsequently, and condensed in Gilles' proposal.

When we proceed with the computer implementation of a model, also a notation was used to identify software patterns for constructing, visualizing and documenting the implementation of chemical processes. This "language" consists of a set of notations and conventions used to record analysis and design decisions in object-oriented environments, the so called *Unified Modeling Language*, UML. The use of declarative languages for modeling chemical engineering systems is continuously increasing and consequently, the use of UML has become more frequent.[4]

In the context of object-oriented programming, *objects* represent special cases of more general ones, called classes. A *class* groups a set of entities that share a common structure and behavior (Bahrami, 1999). The representation of the class architecture of modules and different kinds of relations between them using UML, allowed us to find architectural patterns that emerged while setting up two important kinds of relationships between classes, namely, inheritance and aggregation.

Before describing software design patters found in the implementation of the library, some object-oriented concepts and its representation with UML are treated.

Inheritance, also called a general-specific relation (Kahlbrandt, 2001) is the property that allows objects to be built from other objects. Using UML, inheritance is represented as shown in Fig. 3.4(a). Classes are represented by rectangles with two sections, the class name – usually a subject in its singular form – appears in the top section, while attributes and methods that characterize the class are written in the bottom part. In the context of this work, classes usually represent mathematical models, therefore, equations instead of methods are represented in classes[5]. The inheritance relation is represented through a line with an empty small triangle pointing to the most general class. If a class has more than one superclass, we say to have *multiple inheritance*, a kind of relation that was widely used in the proposed library. In Fig. 3.4(b) the class 'application specific process unit' comprises attributes of two super classes: 'generic process entity' and 'toolbox collection'.

[4]See for example Batres et al. (2002), Bogusch et al. (2002) or Yang et al. (2003) as examples of the use of UML for representing architectural aspects of the implementation of chemical engineering processes.

[5]In the standard UML literature a class is represented with a rectangle with three sections: one for the name, a second for the attributes, and a third one for the methods of the class.

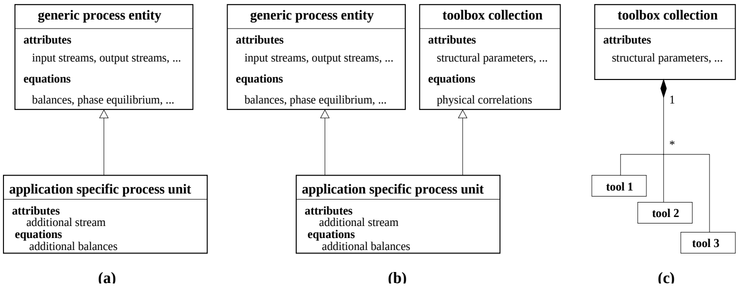


Figure 3.4: Representation of different relations between classes using UML: (a) simple inheritance, (b) multiple inheritance, and (c) aggregation.

Aggregation allows to describe cases in which an attribute is an object itself. Fig. 3.4(c) shows that classes named ’tool 1’, ’tool 2’ and ’tool 3’ are aggregated in the class ’toolbox collection’. This relation is represented with a line. In one of the extremes of the line a diamond is drawn indicating which class aggregates the class or classes that are in other extreme. The number besides the diamond and the asterisk on the other end of the line indicate *multiplicity*[6] of the aggregation relation. The multiplicity indicates how many objects participate in the relationship. The most common multiplicities are designated with 1, * (the asterisk means zero to infinity) and 0..1 (zero or one).

Visualizing the above described class relations with UML constitutes a very useful instrument for simplifying software design decisions and particularly for identifying patterns of relations. The implementation structure of chemical processes becomes complex enough after the first programming stages in such a way, that further development turns very difficult to follow without UML representations. We claim that the use of UML should be taken into account very seriously when chemical engineers work with object-oriented languages.

The progressive phenomenological implementation of entities: Simple-to-complex strategy

The proposed architecture of the library encourages extensive reuse of software and design. Inheritance and aggregation allow to extend and refine classes. Hence, reuse of sim-

[6]The multiplicity of aggregated classes is also known as cardinality.

ple classes promote a great degree of code reuse and avoids replication of code. What here we call a simple-to-complex implementation strategy encourages to abstract those attributes of classes that might be reused in different classes. Unfortunately, there are not rules about how simple a class should be. As a practical guideline for the implementation of classes, we may ask ”Will there be a true need for more than one instance of this class?” (Dattatri (1997), p. 11). The best example of the simple-to-complex implementation strategy is found when the class of a distillation tray was created. Basic features of the tray were abstracted and the most simple class was reused many times in the same model of a distillation column, in different columns of the same process and in different industrial processes.[7]

As discussed below (see page 38), the number of existing modules should be maintained as low as possible. The implementation of many simple classes, with just a few attributes implies that more documentation has to be managed, hence, it is very important that a practical compromise be achieved between many simple classes (easy to implement, but difficult to document) and large and complicated classes with many attributes.

Software design patters

Two branches of the library proposed in this work will be used to describe the identified structuring pattern: The first comprises elementary process entities and the second contains correlations for the evaluation of physical properties. Both of these branches are composed of reusable modules structured according to a semantic criterion in the chemical engineering domain. On one side, application specific classes inherit general phenomenological features, expressed through general purpose entities. On the other side, application specific classes inherit collections of data and correlations required for the particular set of chemical substances processed in each module. Collections of correlations are formed through aggregation of parameterized modules that are semantically organized as shown during the domain analysis. This constitutes the pattern definition illustrated with UML notation in Fig. 3.5 in which for the sake of clarity the implementation of a distillation tray was used as an example. Fig. 3.5 is divided in three parts. The first and the third constitute the whole standard library of ready-to-use modules and the middle part illustrates user applications. Code typing is reduced to providing data for the required correlations for the evaluation of physical properties. During the final stages

[7]More about the simple-to-complex implementation strategy is discussed in chapter 4, page 43.

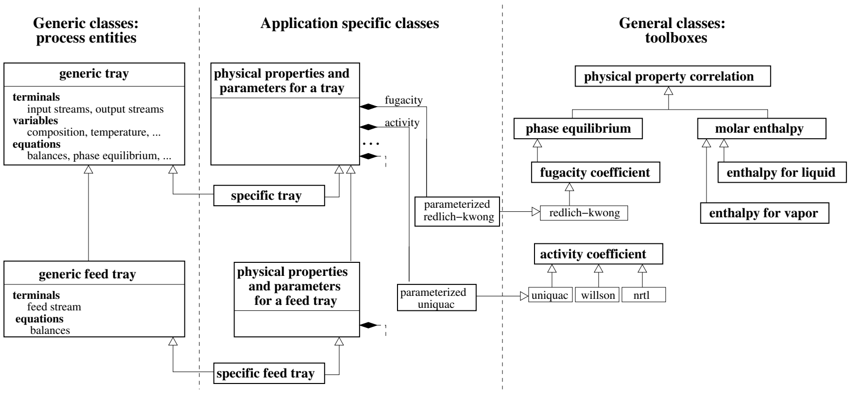

Generic classes: process entities
Application specific classes
General classes: toolboxes
generic tray
terminals
input streams, output streams
variables
composition, temperature, ...
equations
balances, phase equilibrium, ...
physical properties and parameters for a tray
fugacity
activity
specific tray
parameterized redlich−kwong
generic feed tray
terminals
feed stream
equations
balances
physical properties and parameters for a feed tray
parameterized uniquac
specific feed tray
physical property correlation
phase equilibrium
molar enthalpy
fugacity coefficient
enthalpy for liquid
redlich−kwong
enthalpy for vapor
activity coefficient
uniquac
willson
nrtl

Figure 3.5: Sketch of the design pattern that shows, through the implementation of a distillation tray, the organization and use of the proposed library.

of model implementation the user aggregates parameterized modules and links them together, usually using the graphical interface. Code typing is then constrained to the lowest levels of aggregation of the model implementation. Once the data requirements of the whole process are fulfilled, the user joints modules together, allowing him to concentrate on the modeling aspects of the process. Correlations for the evaluation of physical properties are parameterized through classes that are defined very low in the hierarchy, allowing to reuse classes that are located in higher levels. Correlations are organized as described in the domain analysis and illustrated in Fig. 3.3. These modules are aggregated in classes that will be used to parameterize process entities.

The parameterization of a process entity to form a process unit seems to be an arduous task, since many connections have to be established between the calculated values and variables defined in the process entity (see section C.3.6 in appendix C for more details about the implementation of these connections). Classes that aggregate collections of modules for calculating physical properties are however, structurally very similar from process unit to process unit, even though they are used to parameterize a chemical reactor or a heat exchanger. This is an evidence that not only lines of code are reusable, but also structural concepts can be applied repeatedly.

3.3.4 Development of model reuse

Reusing modules should not be more difficult than developing new ones. The fewer lines of code a modeler types in a programming language, the better. Indeed, minimizing the required programming effort is always one of the most important goals of developers of computer-aided tools. For this reason, reusing modules and manipulating them without having to type many lines of code and without having to read large quantities of documentation are goals that are permanent challenges.

The modeler follows the following sequence of steps when using the library organized as described above:

1. Create the subclasses that define the required parameters for each correlation for the evaluation of physical properties that the model needs.

2. Create ”toolboxes” for the evaluation of the required physical properties of each process entity through aggregation of modules created in the last step. It is possible to create a general toolbox for all process units, but this could be considered to be a misunderstanding of the encapsulation approach. It is then possible, but not advisable, since some process units would inherit modules that are not used at all.

3. Create a process unit, which is a self-contained module. This is done through multiple inheritance of, on one side, a process entity in which the equations that describe the process are found and on the other side, of all the required data and correlations which have been collected. This situation is illustrated in the central part of Fig. 3.5, designated as application specific classes in which the process unit ’specific tray’ inherits phenomenological features from the class ’generic tray’ and from the class ’physical-properties and parameters for a tray’ that provides a collection of all modules required for the evaluation of physical properties.

4. If a self-contained module is used for the first time, a preliminary test is recommendable for verifying that the process unit does accomplish the expected performance under the special circumstances of the application.

5. Aggregation of process units can then be performed by linking together self-contained modules.

As expected, modelers are released of taking software design decisions during the implementation of a model. Moreover, they find a sound orientation when having to work with models developed for different persons or implemented a long time ago. Where was a variable defined? What is its default value? Where should it be modified? How are physical properties calculated? Where are balance equations defined? Where can a new stream be aggregated to a process unit? With the proposed structure pattern, the modeler should easily find answers to these questions when manipulating models according to the described methodology.

A software design pattern should be described in a simple form, since its utility resides in its simplicity. Examples of the above described pattern are considered in applications coming in the following chapters.

In what follows four frequent scenarios are considered that describe the interaction of the modeler with the library:

- The modeler finds entities that fulfill his requirements. Usually the next step after finding the required module consists of implementing classes that contain data and parameters. For doing that, the modeler might follow the five-step sequence described above.

- The modeler requires to modify some entities in the library. In this case, advantage can be taken from inheritance and aggregation for performing the required modifications. If inheritance is applied, features of the superclass might be modified or some others can be added to the original set of attributes. It is not possible to eliminate attributes defined in a superclass. If one or more attributes of a class are not required, a process called *factorization* should be performed (see Fig. 3.6). Assume that the 'required class' must be defined without attribute 'c' defined in the 'original class' and perhaps a different one is required instead ('e'). Common attributes ('a' and 'b') are gathered together in a super class and then shared by classes in which non common parameters are defined. The factorized architecture shown in Fig. 3.6 seems to be the best one and the reader might wonder why should it be implemented after the non factorized version on the left side. This is a typical example of the iterative character of the implementation process. At the beginning the modeler might not have envisioned or required to implement the factorized structure on the right. It might also occur that the modeler requires a very similar model to some other previously parameterized, but for a distinct component mixture or distinct physical properties. In this case, it is convenient to make a copy of previous modules and to perform the required changes carefully.

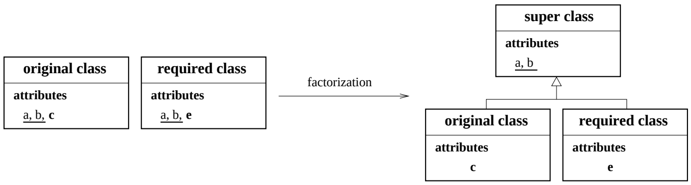


Figure 3.6: Factorization of classes.

- If the modeler does not find the required entity after searching in the library, a new entity might be developed. If this is the case, constraining to the semantic structure of the library is of high importance, since the new entity has to be allocated in some place. Hence, the previous structure can also guide the modeler when developing new entities.
- The modeler documents and stores a module preparing it for reuse. There are not well defined conventions about how large a module should be before being considered a reusable entity. There are two extremes that should be avoided. On one side, very small modules, with just a few lines of code, say a simple equation, augment the structural granularity of the library, increasing with that its complexity and the required documentation that the user has to manage. The other extreme consists of implementing very large modules that demand much time to be grasped. Very large modules tend to have a limited scope of applicability and are difficult to update or adapt for reuse. After an iterative process, a compromise between these two extremes should be reached.

As considered in what follows, reusing modules can be accomplished if the proper culture has been created among the members of a modeling community. Getting acquainted of contributions of previous modelers and developing modules taking into account future modelers are corner stones of the culture of an engineering community, if benefits of developing model libraries are to be achieved.

3.3.5 Repository management

Documentation. If reuse is going to take place or not, depends mainly on whether a potential modeler is aware of the existence of the available modules. Then, a catalog of the available modules in the library including a description of each reusable class is fundamental. The information should include *what* are the characteristics of modules as well as *how* to use them.

Writing documentation constitutes one of the most representative efforts when preparing modules for reuse. The production of documentation does not usually contribute directly to solve current problems of the modeler. Usually, who implements a model has the purpose to use it immediately and very often occurs that no documentation will be required. In this context, creating the culture to document modules constitutes a fundamental practice when building libraries is pursued and all members of the engineering community should be aware of its importance.

A very delicate compromise between the amount of documentation and its effectiveness should be kept. If the potential user of a module has to read large quantities of documentation before being able to make a model working, then the modeler tends to consider the use of entities as difficult and might prefer to implement his own modules for his particular use.

Test information in form of examples is consequently of great value. Some notes about the origin of modules (e.g. the author) in order to obtain support can also be important information. Indeed, a very suitable way for documenting about what can be expected from a module and how to use it is through application examples. Direct references to applications where a module was used are not very hard to include in the documentation and doing so saves much writing and reading efforts.

The number of existing modules in the library. Modules in a library should be comprehensive, but the number of them should be maintained as small as possible. The more general-purpose the library, the more complicated its use and maintenance. A functional criterion when developing a module should be taken into account: a trade-off between too specific (and less reusable) and too generic (and less valuable) should be considered. When a library becomes to be an entity of the reality which can only be understood through a long learning curve, then its functionality and original intentions for supporting the modeling task can just be canceled. However, maintaining the number of entities in the library reasonable low is, according to the development and application experiences of the library, not easy to accomplish. A well defined strategy when adding new modules into the library contributes to maintain the number of entities in manageable dimensions. In Waschler et al. (2003) and in chapter four more about this regard is discussed.

In the library proposed in this work, particular applications are composed, on the one hand, of reused modules already available in the library and, on the other hand, of modules that had to be tailored for the particular application. It is frequently encountered that one of those newly implemented modules becomes a candidate to be added into the library. Hence, before a model entity is a candidate to be included in the module hierarchy it has been tested in a particular application and shown that the module deserves to be included in the library. After its application, the module is isolated and reimplemented to fit general conventions of the library: suitable documentation and notational agreements are to be taken into account. The module is then prepared for its reuse in a more general scope and for different users. It is expected that the new aggregated entity be tested in the particular application it comes from. This process has proven to maintain the number of entities in the library in a reasonable number allowing that new users will not have to face a big challenge when getting acquainted with the class structure of the library.

Continuously evolving modules. Modules in the library are supposed to evolve and adapt during time. Entities in the library should not be considered as ”rooted” pieces of software that are not supposed to change. Two main cases of variability of a module can be identified (Ezran et al. (2002) p. 38):

- Variability through time: the module is maintained, corrected and enhanced to correspond to evolving requirements.
- Variability across applications: modules may be adapted according to the different applications that use it.

In the proposed library of this work modules are grouped according to their particular nature and are organized in single files. When an application is started, a local version of the library is used and the modeler works directly with his private copy. This allows to modify the available modules of the library at any time without affecting other applications. New members of the engineering community take advantage of modules varying through time, while experienced modelers can modify their copy and adapt it for different applications. Modelers tend to specialize the application scope of their models and identify entities with a reuse potential. When a module proves to be useful, it is incorporated to the repository that composes the library to be accessed by other users.

3.4 Summary

In this chapter software engineering considerations were discussed and applied for the construction of the library for modeling chemical processes. After finding conceptual patterns, described by the network approach of Gilles, software patterns were found and represented using UML notation. These patterns constitute an implementation guideline when a model has to be translated into a programming language. Sequences of engineering activities for the construction of the library using computer-aided tools were discussed and applied for the construction of the library. Domain engineering allows to define modules with a strong emphasis on reuse. This is demonstrated with the proposed domain analysis, in which the relevant concepts were identified (Fig. 3.2). With the architecture development software patterns were described (Fig. 3.5). The interaction of the modeler with the library was set with the discussion of model reuse. With that, different scenarios according to the requirements of the modeler are considered. The repository management describes the importance of documenting models and emphasizes the role of creating a development culture between the members of the engineering community to share ideas, notational conventions, and documentation of modules for allowing module reuse.

The discussion in this chapter emphasizes that modeling chemical engineering systems by reusing modules of a library is different than modeling from scratch and that preparing modules for reuse is a process on its own. In this context it was shown how computer-aided modeling impacts the whole modeling task.

Benefits of using a library for modeling means that code typing and debugging operations are reduced to a minimum. Consequently, time devoted for model development should also be significantly reduced.

In the next chapter concepts discussed here are applied for the implementation of the library.

Chapter 4

The model library

Software patterns and phenomenological structures result of domain engineering described in the previous chapter are applied for the implementation of the library. Most important details of classes of the library are presented in this chapter. Details of the phenomenological framework used as a theoretical basis for the implementation of processes entities of the library, as well as notational conventions for the implementation are described in appendixes A, B, and C. The reader will be refereed to them along the text.

4.1 General class structure of the library

Classes in the library are grouped together under different *abstract classes*[1] that are briefly described as follows:

- *Elementary process entities* are non parameterized modules in which a physical or chemical transformation of materials is performed. Typical examples of elementary process entities are reactors, condensers, or mixers.
- *Coupled process entities* consist, in general, of modules of different process entities that are linked together. For example, sequences of distillation trays.
- *Data and correlations* are classes that aggregate modules which calculate physical properties required for an elementary process entity. Classes that define correlations can be inherited by classes that define parameters for an application specific system.

[1]The role of an abstract class usually is to group classes that share some common conceptual features.

- *Signal transformers* usually calculate or provide values of physical properties without calculating these values through physical or chemical considerations. For example, a temperature controller gets temperature information from a process unit (the process variable) and transmits to the same or another process unit the value of the manipulated variable.
- *Terminals* are ProMoT defined abstract classes, that represent interphases between modules. For example, two process devices can be connected by a link that represents the flow of material from one device to the other.
- *Phase equilibrium entities* group modules for phase equilibrium calculations. For example, modules for calculating the boiling and dew point temperatures of a mixture.
- *Process units* are parameterized process entities which are created by inheriting both, process entities and correlations.
- *Physical property correlations* are classes that contain the implementation of correlations that can be parameterized and aggregated to complete the description of a process unit.

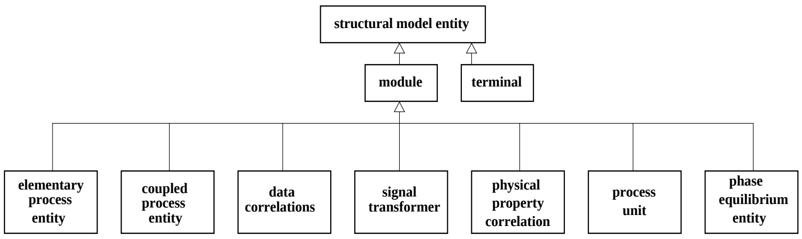


Figure 4.1: UML representation of the most important abstract classes of the library. Abstract classes called ”structural model entity”, ”module”, and ”terminal” are ProMoT defined abstract classes.

The following section describes the methodology followed in the implementation of modules in the library.

4.2 Simple-to-complex implementation strategy

The definition of *simple* and *complex* might depart us from our discussion. Here we will work with these concepts in the context of the implementation of classes of the library.

We argue that inheritance and aggregation of classes facilitate to follow a simple to complex implementation strategy. Classes with no attributes or just a few of them are considered to be the most simple classes. In other words, classes with more refinement, including more phenomenological details, are considered to be more complex classes. In this sense, Fig. 4.1 shows the simplest classes of the library.

Starting from the definition of the most important variables, subclasses *add* more attributes to superclasses. Figure 4.2 illustrates how the conceptual construction of process models in which a phase change occurs and no chemical reaction takes place is built following a simple-to-complex implementation strategy. The process shown in figure 4.2(a) represents a process entity in which a total phase transformation of the input takes place, for example, a total condenser or a total reboiler for which a similar set of variables and equations is required to determine the state of the outlet stream (in section 4.4.2 vapor-liquid separation devices are discussed).

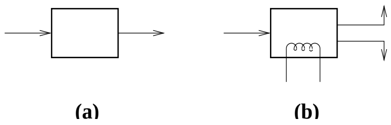


Figure 4.2: The simple-to-complex implementation strategy for process entities.

For the implementation of processes of Fig. 4.2, a class can define the main input and output variables, compositions, pressures, and temperatures as well as the molar holdup. Following the simple-to-complex path, in figure 4.2(b) a source or sink of heat is added, as well as an additional material stream. This flow diagram can represent a partial condenser or a partial reboiler. For cases represented in Fig. 4.2(b), the temperature of the output streams, their composition and their mass flow depend on the state of the input stream and of the value of the heat exchanged. Hence, flow diagrams of Fig. 4.2 may represent two families of models, one in which total condensation occurs and one in which total vaporization is involved.

The simple-to-complex strategy is followed as a natural consequence in the construction of model families, allowing the possibility of structural and conceptual reuse in the computer-aided modeling process.

The implementation of models following this strategy in MDL is facilitated when balance equations are treated systematically. A general statement of mass balances is not possible given the diversity of process entities, but a structural guideline can be applied most of the times. The phenomenological base used for the model implementation is a simple vapor-liquid model, which detailed discussion can be found in appendix A. Balance equations are derived and presented as a conceptual phenomenological framework, that is used as a basis of conceptual reuse. The final result are the vapor-liquid devices of the library. A set of assumptions is proposed and systematically organized in order to reuse these concepts in different process entities that are presented in what follows.

The simple-to-complex approach setting of the material balance allows us to formulate a family of process models that will fulfill a set of phenomenological assumptions. In case that one of the assumptions were not valid any more, the modeler can formulate the mathematical expressions that better describe the new physical situation. This methodology allows to create not only a conceptual frame on which models can be implemented reusing not only pieces of code, but also conceptual information.

In the following sections, the most important features of the modules of the library are discussed. Balance equations presented in the phenomenological framework of appendix A are going to be applied in modules grouped under the class `process-entity`. Since terminals are implemented in almost all types of modules, they are discussed in what follows.

4.3 Terminals

ProMoT allows the definition of data interfaces that couple variables between modules. Such interfaces are called *terminals*. The linkage between terminals represents the formulation of equalities between variables that are defined in each terminal. For example, if the calculated temperature of the bulk liquid phase of a continuous stirred tank reactor is the temperature of the stream flowing into a flash tank, the coupling with a terminal allows that when the value of the reactor temperature changes, the temperature of the input stream to the flash tank will also change. Fig. 4.3(c) on page 49 illustrates how terminals link together different modules.

The definition of terminals is one of the ProMoT features that makes very attractive the implementation of models in MDL, since a direct coupling between physical streams and ”flow” of information can be achieved. Terminals transmit data between modules that comprise compatible connections.

The definition of terminals allows to connect variables of different modules assisted by the graphical editor of ProMoT in a very simple way. When dealing with complex models, which generally comprise highly interconnected modules, the assistance of graphical tools allows to manage the complexity in the implementation phase and minimizes error prone programming operations. The reader is referred to appendix C (section C.3.6, page 114) for more details about the implementation of terminals. Table 4.1 lists the icons used to indicate type and ”expected” direction of the information flow of each kind of terminal. The word ”expected” was written in quotation marks since in ProMoT no information flow can be represented, since MDL is a descriptive language (see section 2.4.1) and flow of data can only be performed in procedural languages. However, when implementing models, physical streams are expected to flow in a given direction, but finally, the calculated direction of a stream will be determined by the solution of the equations describing the system. This allows to describe bidirectional flow of information that will be determined at simulation time instead of having to define the direction with algorithmic sentences. Hence, links between terminals represent bidirectional connections and the use of icons is just a visual guideline that suggests the expected direction of a stream or signal. For example, if the pressure difference of two modules is used for the calculation of the liquid flow, a change in the magnitude of the pressures between modules can change the direction of the liquid flow.

Terminals can be defined with more than one variable, each variable defined in a termi-

Table 4.1: Icons used for showing data flow in terminals.

Type of terminal	Icon	Color
Material flow inlet	•	Black
Material flow outlet	×	Black
Heat flow inlet	•	Red
Heat flow outlet	×	Red
Signal inlet	•	Blue
Signal outlet	×	Blue

nal is called a *slot*. For example terminals that define material flow streams contain five slots or variables: molar flow, temperature, pressure, molar composition and number of chemical species. In what follows, the different kinds of terminals defined in the library are discussed.

4.3.1 Material flow terminals

The definition of these terminals consists of attributes of material flows:

- molar flow, `jn`
- temperature, `te`
- pressure, `p`
- composition, `x`
- number of components, `nc`.

The number of components is part of material flow terminals, since this number can vary along a process when the disappearance of chemical species occurs thorough chemical reactions or the complete separation of any species. *nc* belongs to the so called *structural parameters* which determine the size of numerical arrays in the final algorithmic implementation of ProMoT models. Structural parameters cannot be varied during simulation time, since the size of arrays is fixed when a ProMoT model is compiled creating all the required data structures for the numerical simulation. Hence, the modeler determines when the number of components has to be changed and supplies the appropriate value when required.

4.3.2 Heat flow terminals

Heat flow terminals are in charge to manage heat flows. These terminals contain the variable,

- heat flow, `jq`.

Sources or sinks of heat are usually required when heat contributions are not the main purpose of a module and the value of heat flow is just introduced or transmitted from or to other modules. The heat supply of a reboiler and the cooling requirements of a condenser are examples of the use of heat flow terminals.

4.3.3 Signal transmission terminals

Signal transmission terminals are used as input/output interfaces of process units usually connected to controllers. Hence, signal transmission terminals are defined with just one variable,

- datum, `dt`.

A 'datum' is a generic designation of values of variables of different nature, which might represent either process variables, set points or manipulated variables. For example, the variable `te` can represent the value of the temperature of the reactor; `set_te` might be the set point of the temperature; and `jn_feed` might be the manipulated variable that represents the molar flow of one of the reactants into the reactor. All these values of different nature might be transmitted through signal transmission terminals.

4.4 Process entities

Process entities are not general-purpose models, since a set of assumptions behind their definition constraints their use. However, we have some flexibility, since they exhibit a wide scope of application. Adaptation or further development of process entities can also follow a systematic strategy, since notational conventions and structural approaches preserve their usefulness during the implementation of entities that fulfill different assumptions.

The basic parts of process entities are the following:

- *Variables*. Independent variables are defined as input variables, while dependent variables are defined as output or state variables.
- *Parameters*. Even though parameters are *declared* in the section of variables, they are *defined* as real parameters in MDL.

- *Equations*. Each equation is defined and characterized by a name. For ensuring the complete mathematical specification of a process entity, the number of equations and the number of state variables have to be equal.

- *Terminals*. Through terminals, input variables get their values and output variables determine the value of properties of the outlet streams.

Appendix B presents the degree of freedom analysis of selected process entities of the library. The most representative models of the library are included in this appendix, for that, it can be used as a reference guide.

In what follows, the most important process entities that constitute the library will be described.

4.4.1 Input/output modules

Input/output modules usually represent sources or sinks of mass or energy, but also signal sources – set points for example. For creating a consistent set of equations, a model requires that all its terminals be connected, no open terminals are allowed, since variables that are contained in all terminals require to be assigned. When a module is completely connected in ProMoT, we say that the model is *closed*. Fig. 4.3(a) shows icons for the three available input/output modules: a reservoir, a parameter source and a sink. An open model of a process unit, which does not contain any input/output module, is represented in figure 4.3(b). Part (c) of figure 4.3 shows the closed model with no open terminal.

Reservoirs

Input modules provide forcing functions for process entities. If forcing functions determine material streams or heat flows, they are called *reservoirs*. The most common case consists of providing a constant material flow or a constant heat flow. For these cases, parameterized equations are defined in reservoirs and these parameters can then be manipulated during the simulation. Changing the value of these parameters constitutes disturbances in the input variables of a model.

Table B.1 lists the attributes of the implementation of a reservoir that represents a liquid stream. The role of this reservoir is to assign numerical values to the variables that characterize a liquid stream i.e., molar flow, temperature, composition, and number of

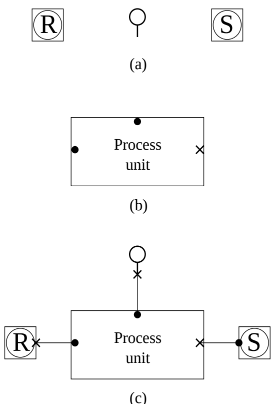


Figure 4.3: Input/output modules and closed models. (a) Icons for input/output modules; from left to right: a reservoir, a parameter source, and a sink. (b) The representation of a process unit that is already an open model. (c) A closed model with all terminals connected.

components. The values of these physical properties are given through forcing functions defined in the equations section of these modules.

Sinks

Output modules represent sinks of materials or energy. The most important role of these modules is to close models, allowing to catch the value of variables defined in the output terminals of process units. Output modules can also be sources of information, since connections in ProMoT are also bidirectional. An example of a sink of material that sends back information is a vapor sink or vapor output module. The pressure is a variable that in the proposed library, is assigned backwards or in counter direction with respect to the material flow. This reverse assignment of the pressure is justified if we observe that the state of vapor phases is strongly influenced by the total pressure. In general, in the applications of this library, the vapor flow was not calculated performing momentum balances for the calculation of the pressure around process entities, but considering the vapor holdup negligible all time and calculating the vapor flow using the energy balance. However, the value of the pressure is important considering the vapor phase and an empirical correlation or a fixed pressure gradient or pressure loss is used for calculating the pressure. Hence, the lowest pressure is fixed in the module in which the vapor is expected to flow (usually at atmospheric pressure) and the pressure of process units is calculated from this minimum value. More about the pressure value calculation is discussed in section 4.4.2 page 56. In summary, a vapor sink gets the molar flow, the temperature and the composition of the vapor stream, but the value of the pressure is sent out of the module. Table B.2 describes vapor sinks with more detail.

Parameter sources

This kind of input module is required when the value of a variable might be fixed outside of a process unit. These values might come from the output signal of a controller, a device that fixes a set point or just a value given by the modeler.

4.4.2 Vapor-liquid separation devices

Inside these kind of devices, thermodynamic equilibrium between the vapor and liquid phases is assumed. Even though the so called equilibrium models have been considered to be non accurate (Taylor et al., 2003), these kind of models are a starting point to build a conceptual structure that can be refined at any time when the requirements emerge and rate-governed models, which include much more data and parameters, can be formulated upon the conceptual basis of equilibrium models. The equation structure of devices grouped in this category is very similar. Reboilers, condensers or distillation trays belong to this category.

In what follows, a detailed discussion of the model of a condenser is presented. The detailed discussion allows us to illustrate implementation features of other modules in the library. Particularities of reboilers and distillation trays models will be mentioned in its turn.

Dynamic equilibrium model of condensers

The process entity.

The module that represents the process entity of a condenser is implemented in such a

way that it is an application independent model, developed of course, using the assumptions discussed in appendix A. Figure 4.4(a) sketches the flow diagram of this process entity. Focus is on the fluid to be condensed, the cooling fluid is not considered in this model, however, it may be taken into account when the temperature difference or the flow of the cooling fluid, as well as geometric features of the condenser, like the total cooling area, are important to be modeled.
In this work, all applications took into account only the side of the fluid to be condensed. Since the condensed liquid mixture is considered to be perfectly mixed, the model of a

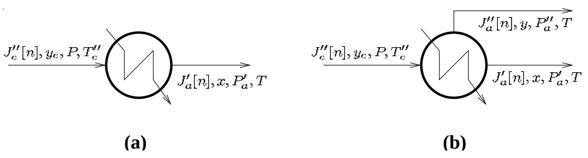


Figure 4.4: Flow diagrams for the process entities that represent condensers. (a) Total condenser. (b) Partial condenser.

total condenser has the following output variables:

- the saturation temperature, T, of the liquid mixture in the outlet stream
- the *nc* compositions of the liquid, x_i, which are in equilibrium with
- the *nc* compositions of the vapor, y_i
- the molar flow of the liquid outlet, $J'_a[n]$.

Hence, the model of a total condenser consists of $2nc+2$ variables. Let's start setting up the assumptions that describe the process and formulate the equations of the system.

Since the outlet liquid stream is assumed to be a saturated liquid in thermodynamic equilibrium with traces of vapor (no subcooling of the liquid takes place), the energy balance is not required in the model of a total condenser for the calculation of the saturation temperature. The condensed molar holdup collected in the accumulator remains constant all the time and the gas holdup is considered to be negligible with respect to the liquid content of the condenser. These assumptions, together with those set for the material balance, described in the phenomenological framework of appendix A, can be applied to set the model of the condenser (see table 4.2). The final form of the equations set on the conceptual set on the conceptual framework is discussed in what follows.

According to the constant molar holdup assumption for the liquid, the total material bal-

Table 4.2: Assumptions for the description of process entities with vapor-liquid interaction.

1. Assumptions for setting up the material balance of a liquid phase of table A.2 are still valid:
    - 1.1 The content of the control volume is perfectly mixed
    - 1.2 The fluid is considered to be incompressible
    - 1.3 Mixting effects are neglected
2. The molar liquid holdup is constant
3. The vapor molar holdup is negligible
4. The vapor phase behaves as an ideal gas
5. The feed stream is a saturated flow
6. The condensed phase is a saturated liquid in thermodynamic equilibrium with its saturated vapor

ance expressed by Eq. (A.1) for the case of the total condenser reads,

$$0 = rhs\_tmb. \tag{4.1}$$

Even though the liquid molar holdup remains constant, compositions of $nc-1$ chemical species are determined by the material balances for the components as expressed by Eq. (A.3):

$$\frac{V}{v'}\frac{dx_i}{dt} = rhs\_cmb_i - x_i rhs\_tmb \tag{4.2}$$

where V is a parameter of the system that represents the value of the total liquid volume. v' is the molar volume of the liquid mixture, which is calculated with a temperature dependent empirical correlation for the pure species. The last liquid composition is given by the definition of the liquid fraction mole:

$$\sum_{i=1}^{nc} x_i = 1 \tag{4.3}$$

The $nc-1$ vapor compositions in equilibrium with the liquid and the saturation temperature are calculated with nc vapor-liquid equilibrium relations:

$$y_i P = x_i \gamma_i P_i^o \tag{4.4}$$

The last vapor composition is calculated with the summation condition for the vapor phase:

$$\sum_{i=1}^{nc} y_i = 1 \tag{4.5}$$

Equations (4.1)-(4.5) provide $2nc+2$ relations needed for the complete description of our model for the process entity that represents a total condenser. This process entity can be reused in different applications, each one consisting of different chemical species.

The process unit.

As stated in section 3.3.3 and sketched in Fig. 3.5, a process entity is not an application specific module. For setting a closed model, i.e. a completely specified model, the so called process units have to be implemented and the parameterized correlations for the evaluation of physical properties have to be provided. Among the application specific properties, the evaluation of the following temperature dependent properties are required for each chemical species of the condenser model:

- the vapor pressure as a function of the temperature, P_i^o
- the molar volume, $v_i'^0$
- the activity coefficient, γ_i

In the proposed library, there exist different correlations that include the above set of correlations, which have to be parameterized for a specific application. Once created, these parameterized modules are aggregated in a class that collects all required data and correlations and then the process entity of the total condenser is put together with the data and correlations through multiple inheritance. At that stage the process unit can be used as an independent module that can be modeled as a stand alone process, if the condenser is connected to the appropriate reservoir and sink or if the condenser is connected to other process devices.

Table 4.3 summarizes the degree of freedom analysis of the process entity that represents a total condenser, and the additional correlations required for creating a process unit for this model. Parameters, input and output variables as well as the equations of the model are defined in the process entity class, while the specific application correlations and their parameters are defined in an class that aggregates all modules, which together with the process entity will constitute a process unit. Subtracting the total number of equations by the total number of output variables shown in table 4.3 shows that the process entity is

Table 4.3: Detailed model and degree of freedom analysis of a total condenser that fulfills assumptions of tables A.2 and 4.2.

Parameter name	**Notation in the text**	**Number of parameters**	**Units**
Pressure difference	$delta\_p$	1	bar
Total volume	V	1	m^3
Input variables		**Number of input variables**	
Vapor molar flow	$J''_e[n]$	1	kmol/s
Inlet vapor temperature	T''_e	1	K
Inlet vapor composition	$y_{e,i}$	nc	[-]
Pressure	P'_a	1	bar
Physical property name		**Number of physical properties**	
Vapor pressure	P_i^o	nc	bar
Molar volume	v_i^o	nc	$m^3/kmol$
Activity coefficient	γ_i	nc	[-]
Output variables		**Number of output variables**	
Liquid molar flow	$J'_a[n]$	1	kmol/s
Saturation temperature	T	1	K
Liquid composition	x_i	nc	[-]
Vapor composition	y_i	nc	[-]
Pressure	P	1	bar
Total number of output variables		2nc+3	
Equations		**Number of equations**	
Total material balance	$0 = rhs\_tmb$	1	
Component material balances	$\frac{V}{v'}\frac{dx_i}{dt} = rhs\_cmb_i - x_i rhs\_tmb$	nc-1	
Vapor-liquid equilibrium	$y_i P = x_i \gamma_i P_i^o$	nc	
Summation condition (a)	$\sum_{i=1}^{nc} x_i = 1$	1	
Summation condition (b)	$\sum_{i=1}^{nc} y_i = 1$	1	
Pressure loss	$delta\_p = P - P'_a$	1	
Total number of equations		2nc+3	

already a completely specified model. Hence, the degrees of freedom for the model are defined by the input variables of the model.

The information of table 4.3 is presented in an abbreviated form in table B.3. In appendix B, similar tables are presented for selected process entities of the library.

The partial condenser

As shown in Fig. 4.4(b), in a partial condenser an additional vapor stream has to be included in the model, since not all the vapor input condenses. The additional vapor output of the model will be represented with an additional terminal that may include the properties of the vapor stream. Also a terminal representing the cooling duty of the condenser may be required.

The vapor flow in the partial condenser will depend of the cooling duty. Hence, the energy balance is required as an additional piece of information. Following the simple-to-complex implementation strategy, equations implemented for the total condenser can be reused as well as data structures, variables, parameters, terminals, and modules for the evaluation of physical properties. The equations that require to be adapted for the partial condenser are those that take into account the additional vapor stream, namely, the material balances. Given the introduced abbreviations for expressing the algebraic terms of the material balances, only the right hand side of the material balances have to be redefined for including the additional vapor stream.

Assumption 6 of table 4.2 states that the vapor-liquid mixture inside the condenser is in thermodynamic equilibrium. This requires that the temperature of the condenser is the boiling point temperature of the mixture that is calculated with Eq. (4.4), which is an algebraic equation.

The energy balance of Eq. (A.16) can be used for the calculation of the vapor flow rate. However, since Eq. (A.16) includes a time derivative of the algebraic equilibrium temperature, using this equation would lead to a differential index of 2 (see Pantelides et al. (1988) or Lefkopoulos and Stadtherr (1993)). To overcome this problem, the time derivative in Eq. (A.16) will be neglected, corresponding this situation to a *quasi stationary* temperature dynamics. Therefore, the differential $\frac{dT}{dt}$ of Eq. (A.16) will be considered equal to zero. The energy balance for the partial condenser reads:

$$0 = rhs\_eb - \sum_{i=1}^{nc} h_i' rhs\_cmb_i. \tag{4.6}$$

Hence, the additional variable, the vapor flow $J''[n]$, is calculated with Eq. (4.6) and with that, we have again a complete system of equations. Of course, the cooling duty of the condenser has to be given as an input variable (or parameter) to the system. This can be done through an energy reservoir. If on the contrary, the cooling duty of the condenser were to be calculated, the vapor flow should be given as an input of the system and the cooling duty be considered as an output variable. Though the model of the partial condenser is very similar to the total condenser, the summary of the model is presented in table B.4 since the discussion about the formulation of these two models will be considered in subsequent developments. It is important to note that the pressure of the total condenser is determined by the pressure of the condensed liquid ($delta\_p = P - P'_a$), while the pressure of the partial reboiler is determined by the pressure of the non condensed vapor stream ($delta\_p = P - P''_a$). Here is a good point to discuss about the treatment of the pressure in the modules of the library.

The pressure calculation and assignment

The total condenser gets the value of the pressure of the outlet stream from the process device upstream. Hence, the outlet pressure is an *input* variable in the condenser. The operation pressure, P, of the condenser is an *output* variable calculated with an expression that belongs to the set of equations of the condenser (say for example, $P = P_a + delta\_p$). This value of P will be the the pressure of the outlet stream of the process device connected to the feed stream of the condenser, which on its turn will contain an equation for the evaluation of its operation pressure. Hence, the pressure of coupled devices is evaluated downstream.

However, there might be numerical conflicts if material flow terminals include the pressure in their slot, as discussed in section 4.3, especially, if a process includes recycling streams, like that sketched in Fig. 4.5. The pressure of the reactor (P_{r1} in the figure) is determined by the value ”sent back” from the condenser through its terminal, then, in the reactor P_1 is calculated and assigned to the purification system which calculates, P_{r2} that is again sent back to the reactor. Hence there would exist a conflict between P_{r1} and P_{r2}. To avoid this problem, terminals between the reactor and the purification system do not contain the pressure in their slots and the pressure of the purification system should be determined independently of the reactor. For example, the sink module designated as Products in Fig. 4.5 can fix the value of the pressure of the purification system.

The UML representation for the condenser models

Fig. 4.6 represents a particular application of the architectural pattern discussed in the last

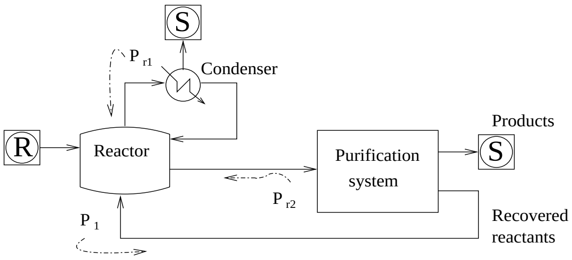


Figure 4.5: Example of the flow diagram of a process in which both, the recycling stream of recovered reactants and the reactor output to the purification system should not include the pressure.

chapter and it is used to sketch the structure of the implementation of the above discussed condensers. In the class designated as `condenser` the required parameters, variables, and equations are defined, while modules for the calculation of physical properties are aggregated in class `condenser-dc`. These two classes are then inherited by the application specific unit. The class `condenser` may be used many times while the phenomenological assumptions in setting up the model remain valid. Class `condenser-dc` might be structurally reused between different applications. Suppose for example that a condenser has already being implemented, consequently, modules of physical correlations have been aggregated for this condenser if a second condenser for a very completely different chemical system were to be required, the structure of the class which aggregates the all correlations for the first case can be copied and just the modules that contain data for the new mixture can be modified. Fig.4.6 represents a more detailed illustration of the structure of process entities and process units shown in Fig. 3.5.

So far, three different representations have been used to set up the model of a total condenser; namely, the process diagram of Fig. 4.4(a), the equation set of table 4.3 and finally, the implementation architecture using UML notation of Fig. 4.6. These three kinds of representations are three different forms to carry out the set up and implementation of models within the environment provided by a computer tool and a library of reusable entities. All of them complement to each other in the modeling process.

The proposed total and partial condensers are not fixed models. New model families can be developed if different accuracy is required with the evaluation of physical properties

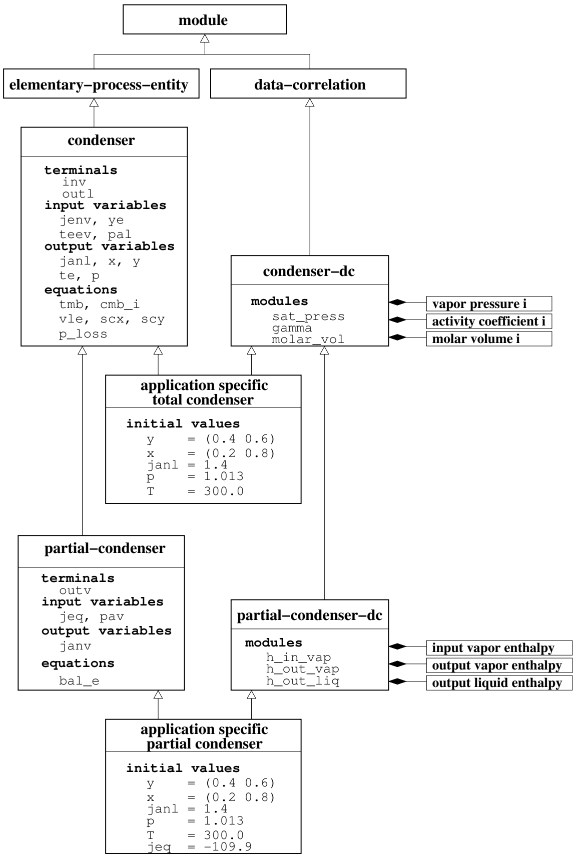


Figure 4.6: Representation of the implementation architecture for the process entities of total and partial condensers according to UML notation. Multiple inheritance is used for the implementation of the respective application specific process units.

or phenomenological coefficients, like the activity coefficient or if assumptions of tables A.2 and A.3 are not fulfilled any more. For example, if the pressure of the system does not allow to consider the vapor phase as an ideal gas, equations that describe the vapor liquid equilibrium (4.4), can be reimplemented to evaluate the non idealities of the gas phase. We have another interesting example when the molar liquid holdup is not considered constant anymore in the total condenser. In this case the differential term of equation, $\frac{dn'}{dt}$ of the mass balance is not equal to zero anymore and a new equation should be added to the system for the variable n'. A controller might provide the additional information, or an empirical correlation for the calculation of the liquid flow can represent the additional required information.

Similar model development and implementation procedures as described in detail for the condensers were followed for all process entities of the library and the relative detailed discussion illustrates the process followed in other entities. The family of models proposed for the condensers, are prototypes that the modeler might use as a conceptual and implementation base. In a more brief fashion, selected entities of the library are described in what follows.

Reboilers

The model development of what we call an elementary reboiler, shown in Fig. 4.7(a), is very similar to that of the total condenser. In this case a saturated liquid stream is completely vaporized and the same assumptions as in the previous case are considered. The only difference consists in that for the case of the condenser the calculated temperature corresponds to the bubble point of the mixture, while in the total reboiler the calculated temperature is the dew point of the mixture. Table B.5 shows the model of a total reboiler.

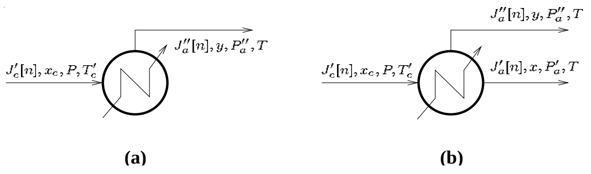


Figure 4.7: Flow diagrams of (a) a total reboiler and (b) a partial reboiler.

Similarly as in the case of the partial condenser, an energy balance is required for the calculation of the non vaporized liquid flow in a partial reboiler. Its flow diagram is shown in Fig. 4.7(b).

Flash tank

Assumptions and equations developed for a partial reboiler can be applied for describing a flash tank. The process is sketched in Fig. 4.8. A stream of saturated liquid flows into

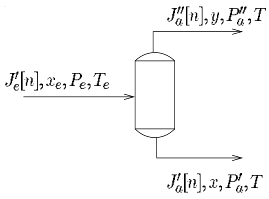


Figure 4.8: Flash tank flow diagram.

the tank in which an adiabatic expansion occurs due to the sudden pressure reduction, hence in the diagram of the figure the outlet pressure should be lower than the pressure at the feed stream ($P''_a < P_e$). The outlet streams consist of a saturated vapor and a saturated liquid in thermodynamic equilibrium. In the partial reboiler, an input variable for the heat duty was defined, but if this model is used as an adiabatic flash tank, the heat duty should be equal to zero. A single stage non adiabatic flash, however, is not often used in practice. Usually flash tanks are considered to operate adiabatically, hence, with the heat input equal to zero ($J_e[q] = 0$), the partial reboiler model can be used as a flash tank.

Distillation tray

The phenomenological base developed so far, can be reused to implement the model of distillation trays and to create a family of models with a wide application spectrum.

In an elementary distillation tray a separation process occurs taking advantage of the boiling point differences of the components of a mixture. Table 4.4 lists assumptions set for process entity that describes an elementary tray in this library. Each phenomenological statement of the table has been discussed in appendix A and in former sections of this chapter. Also for the model of an elementary tray, a detailed description is presented in table B.6. This elementary model is used as a starting point for the development of a family of models. Examples of some variations of the elementary version are shown in Fig. 4.9. The addition of new streams is taken into account by just redefining the right-hand sides

Table 4.4: Assumptions for setting up the model of an elementary distillation tray.

1. The content of the control volume is perfectly mixed
2. The liquid fluid is considered to be incompressible
3. The molar holdup of the liquid is constant
4. The molar holdup of the vapor is negligible
5. Mixing effects are neglected in the liquid
6. Kinetic energy is negligible
7. Potential energy is negligible
8. The differential $d(Pv')$ is negligible
9. The temperature has a quasi stationary dynamic behavior

of the balance equations and by aggregating modules to calculate the additionally required properties. For example, Fig. 4.9 (b) represents a feed tray in which the feed stream is a saturated liquid. The model of the feed tray will require the definition of all properties of the additional stream ($J'_f[n]$, T_f, x_{fi}) and the aggregation of modules for the evaluation of the liquid enthalpy of the components $h'_i(T_f)$. Hence, if additional features are required, a subclass of the elementary model is created and additional variables, parameters, and modules are defined and aggregated. That is, we follow a simple-to-complex approach in the implementation of model families allowing us to implement the additional features *only* when it is required in the practice. It could be argued that a very complex model would be suitable and the modeler just would have to simplify or eliminate what is not required in a particular case. If we consider Fig. 4.9 (f) as the most general case, four feed streams and the chemical reaction contribution of the model would have to be eliminated for getting an elementary tray. But Fig. 4.9 (f) does not represent what can be considered the most detailed case of a distillation tray, since always more streams can be added, or different phenomenological considerations than those listed in table 4.4 might be taken into account. Modules should remain the most simple as possible. That is why, the simple-to-complex approach was followed during the development and the applications of the library proposed in this thesis. In addition to examples of Fig. 4.9, the elementary model can be modified according to different kind of requirements, for example:

- Mixtures with high polarity. The election of a suitable calculation of the activity coefficient takes the polarity of the mixture into account.

- Low to moderate pressures. The ideal gas law can be applied within a wide range

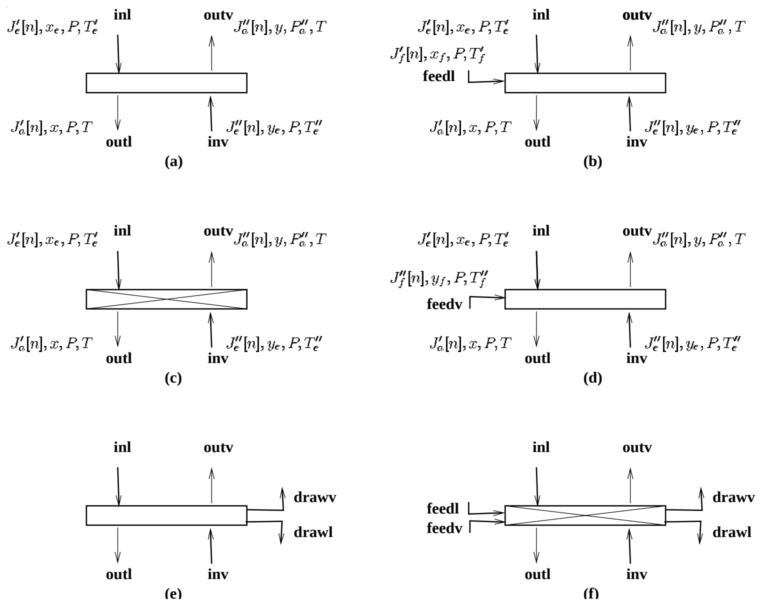


Figure 4.9: Examples of the modified versions of the process entity for an elementary distillation tray. Terminal names are printed with black characters. Names of variables were omitted in the last two examples. (a) Elementary tray process entity. (b) A saturated liquid feed stream is added. (c) The crossed out tray indicates that the model includes chemical reaction . (d) A saturated vapor feed stream is added. (e) Side-draw streams of saturated vapor and liquid are added. (f) All the former variations are included.

of pressure values for non polar mixtures. The evaluation of a fugacity coefficient can be considered aggregating the suitable modules in case of polar mixtures.

- High pressures. Equations of state can be implemented like modules for the calculation of physical properties and then aggregated when required.

- Chemical reactions. The reaction rate can be considered as a correlation that is aggregated together with the set of phenomenological tool boxes of the process entity.

- If rate based models are to be considered, transport resistances can be considered as additional phenomenological features. If the contribution of these resistances to the dynamics of the process becomes important, these contributions can be grouped in classes that represent interphases between well mixed phase containers.

4.4.3 Chemical reactors

In the present contribution an evaporatively cooled stirred tank reactor is considered (see section 5.2), which can be modeled in a similar way like a reactive distillation tray.

4.4.4 Other process entities

Decanter

At least two decanter models have been implemented for the applications of this library. The model described in this section was used in the model of a plant for the production of butyl acetate to be discussed in the next chapter and the flow diagram is shown in Fig. 4.10. It is assumed that in the decanter two liquid phases are in thermodynamic equilib-

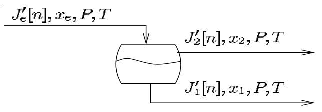


Figure 4.10: Flow diagram of a decanter.

rium. Such equilibrium is described for each component of the mixture according to the following equation:

$$\gamma_{1,i} x_{1,i} = \gamma_{2,i} x_{2,i} \tag{4.7}$$

where indexes 1 and 2 denote the liquid phases. The molar holdup of the decanter n is considered to be constant. The model calculates the overall composition, x_i, and for each phase: the composition, the molar holdups, and the molar flows. It is important to be careful when linking the module of a decanter with other process units, since there are two phases that in reality are physically well differentiated if we consider that the phase with

Table 4.5: Assumptions for setting up a decanter model.

1. Two liquid phase are in thermodynamic equilibrium
2. Each phase is perfectly mixed
3. The overall molar holdup is constant
4. The temperature is constant

higher density will be found at the bottom of the decanter. This fact is used to withdraw the heaviest phase at the bottom of the tank, while the lightest phase will be withdrawn at the top of the tank. The modeler needs to force that the most dense phase corresponds to one of the streams. For that, the initial conditions of the model should be such, that one of the phases be rich in the most dense component, while the other phase be rich in the lightest one.

Table B.7 lists more details of the decanter model. Note that the value of the pressure is assigned through a parameter, since the pressure was not considered to be important in this process entity model. The temperature has a similar treatment, but in this case the value comes from the input stream and this value is used for the equilibrium calculations of equation 4.7.

Reflux Drum

These kind of devices are especially useful when the reflux ratio of distillation columns has to be modeled once the condensed vapor stream at the top has to be divided. The reflux ratio is given as a parameter.

A liquid drum divides the flow of a liquid stream in two smaller ones. If the volume of the drum is small enough the drum behaves as a stream separator with an almost instantaneous dynamics of the composition. The situation is shown in Fig. 4.11. Since the holdup is considered to be perfectly well mixed, the properties of the outlet streams, with exception of the value of the molar flow, are identical. Table B.8 shows a survey of the drum model. Note that the definition of the stream ratio prevents to run into numerical inconsistencies if one of the output molar flows goes to zero.

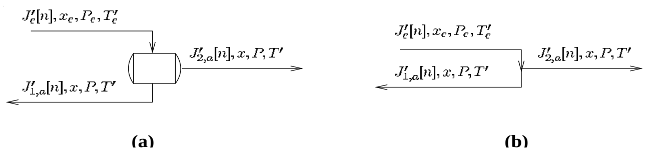


Figure 4.11: (a) Liquid drum. (b) Stream separator.

Stream mixer

An arbitrary number of streams with also an arbitrary number of chemical components can be connected to the module that represents a mixer and calculate the properties of a single outlet stream as sketched in Fig. 4.12(a). The model of a mixer requires two cou-

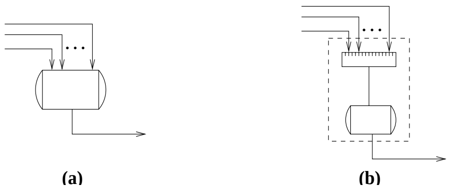


Figure 4.12: (a) Flow diagram of a stream mixer. (b) Representation of two coupled modules in the implementation of a mixer.

pled modules as shown in Fig. 4.12(b). The first one gets the information of the streams: the total number of streams, and for each stream: the number of components, the molar flow, the composition, and the temperature. All these data are allocated in a numerical array which is transmitted to the second module where material and energy balances are performed. The calculated variables of the mixture are the molar flow, composition, and temperature of the outlet stream.

4.5 Signal transformers

Fixing the set point of a temperature controller or a process unit can be achieved by an input module which provides the desired value. However, in reality, fixing a set point might be achieved by a more complex device that may consists of mechanical and electronic components, whose representation is not important for our purposes. Signal transformers do not represent process entities since no physical or chemical changes are described by them. The class called `parameter-source` groups modules whose purpose is to provide or receive data to or from process entities or other signal transformers, like controllers. Modules grouped under the class `parameter-source` might consist of just an isolated value, like the value of the temperature or pressure, or an array of data like the composition of a mixture. Also controllers are signal transformers since no physical or chemical

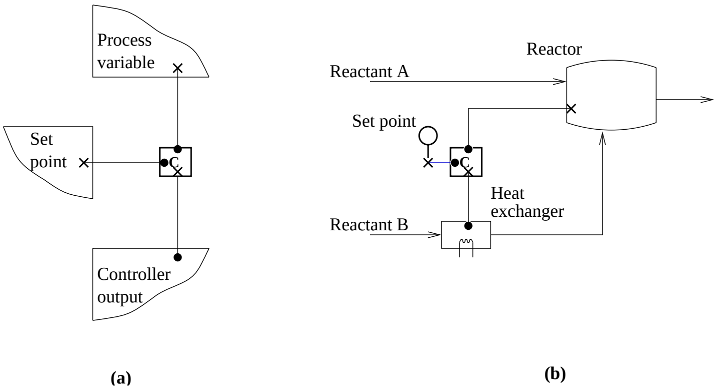


Figure 4.13: (a) Representation of modules that represent input/output modules connected to a controller. (b) Example of a temperature controller that regulates the value of the temperature of the reactor, controlling the temperature of the stream of reactant B.

changes occur in them. Modules of controllers implemented in the library receive the value of the process variable and set point as inputs and a control output is generated depending of the type of controller, either proportional or proportional-integral response controllers. As sketched in Fig. 4.13(a), in which a controller is represented with a framed letter **C**, values of the set point and the process variable might be data provided by modules of any kind. For example, suppose that reactants *A* and *B* are feed to the continuous stirred tank reactor of Fig. 4.13(b) whose temperature is to be controlled by the temperature of the feed stream of reactant *B*. The controller gets the value of the process variable from the reactor (the temperature) and the set point from a data input module (the required value of the reactor temperature). The control output is the value of the temperature of feed stream of reactant *B*, which is sent to a module that represents a heat exchanger. In this example the set point module illustrates the flexibility of the proposed controller module, since this set point entity might be substituted by a process unit that determines the value of the reactor temperature set point.

The calculated controller output, *CO*, might consists of a proportional response to the controller error $= SP - PV$:

$$CO = K_c(SP - PV) \tag{4.8}$$

where *SP* and *PV* stand for the set point and process variable, or a combination with an integral response:

$$CO = K_c[(SP - PV) + \frac{1}{\tau_I}\int(SP - PV)dt] \tag{4.9}$$

where the τ_I, the integral time or reset time, as well as the gain K_c have to be given parameters.

The controller output represents physical values in the modules proposed in the library, since no intermediate device (like valves, for example) was considered in our modeling purposes. For example the output of the controller of Fig. 4.13(b) represents directly the value of the stream temperature instead of some other indirect value like the fraction of wide open position of the stem of a control valve, which would determine the flow of a heating flow of the heat exchanger of Fig. 4.13(b). If required, the controller output expression (Eqs. 4.8 or 4.9) should be over written by expressions that include features of the desired control device like a control valve.

Table B.9 presents the detailed model of a general purpose proportional-integral controller. This table allows to show that conditional statements can be used with MDL. The value of the controller output is limited by the maximum and minimum values of the manipulated variable. These limits are given parameters that depend of the nature of the context in which the controller is applied.

4.6 Data and correlations

Modules used for the evaluation of physical properties are organized independently of any application context. These modules constitute tool boxes that, once parameterized, can be flexibly used repeatedly within an application. These modules are organized as shown in Fig. 3.3 and regarding the particularities of the implementation of correlations, the reader is referred to page 117 in appendix C.

It deserves to be mentioned here that each module for the evaluation of physical correlations has a local scope. This means that each process unit is provided with data and correlations required for the equation set defined in the correspondent process entity.

4.7 Summary

A general description of the main branches of the class architecture was presented at the beginning of this chapter. After that, the phenomenological framework of appendix A was presented to achieve one of the most important aspects of the implementation of models in the library: *their reusability*. Among the ideas of this conceptual frame are the conventions to create names of variables, terminals, and equations. These issues are described in the respective appendixes for the interested reader.

A more detailed description of the implementation of the process entity and process unit of a total and partial condensers was used as illustration for the model development methodology of other process entities and units.

Three different representations are used to support the development and implementation of models, namely: the process diagram; the equation set, represented in different tables of degree of freedom analysis; and the UML representation that visualizes aggregation and inheritance relationships between classes. These three representations complement each other and support the computer-aided-modeling task.

This chapter was mainly devoted to describe process entities of the library. In the next chapter, process units are coupled together to model complete chemical processes and numerical results are also discussed.

Chapter 5

Applications

The applications presented in this chapter consist of two different industrial chemical processes: the first one is a plant for the production of butyl acetate and the second one is a plant for the production of acetic acid. The highly non-linear phenomena that characterize these models is shown with the numerical simulation of the models.

The development of the library and the development of applications were related in Fig. 3.1. The two applications presented in this chapter illustrate that each modeling project could be carried out reusing the same modules present in the library and parameterized and adapted for particular cases.

Different aspects of the application development process are emphasized through these examples: in the plant for the production of butyl acetate the modularized assemble of process units is presented with more detail, while particularities of discrete event modeling were pointed out in the plant for the production of acetic acid.

For both industrial processes dynamic simulations are presented and briefly discussed.

5.1 Reactive-distillation process for the production of butyl acetate

The plant for the production of butyl acetate sketched in Fig. 5.1 is presented as a first example for the application of the library. The whole process is briefly described and then a more detailed description of each process unit and their internal structure are considered.

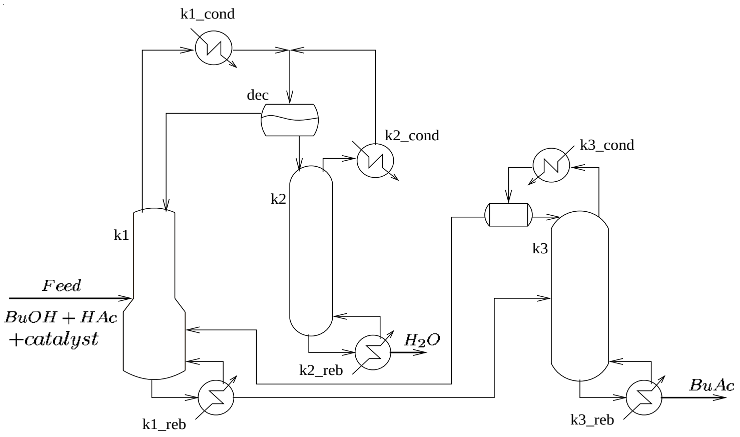


Figure 5.1: Flow sheet diagram of the plant for the production of butyl acetate.

In this continuously operated process (Hartig and Regner, 1971), liquid n-butanol and acetic acetic are fed to column `k1` and react in the stripping section of the column to the products butyl acetate and water according to the reversible esterification reaction

$$CH_3COOH + CH_3(CH_2)_3OH \rightleftharpoons CH_3COO(CH_2)_3CH_3 + H_2O, \tag{5.1}$$

which occurs in the liquid phase and is homogeneously catalyzed by means of a strong acid. The more volatile water is boiled up and removed from the reactive section, thus allowing almost total conversion of reactants. At the top of the rectifying section a vapor mixture rich in water is obtained, which is completely condensed in condenser `k1_cond`, and the condensate is collected in decanter `dec` where a liquid-liquid phase split occurs. The organic phase rich in the ester is recycled to column `k1`, while the aqueous phase is fed to the stripping column `k2` for further purification. Almost pure water can be withdrawn as the bottoms product, while recovered reactants are recycled to column `k1` via the decanter.

The bottoms stream of column `k1`, which is very rich in ester, is fed to the distillation column `k3` in which an almost pure bottom product is obtained. The second major recycle of the process from the top of column `k3` back to column `k1` serves again to recover reactants.

5.1.1 Modularized construction of the model of the plant

The mathematical model of the process of Fig. 5.1 consists of 2258 state variables, 570 of them are dynamic variables and 1688 are algebraic variables. The proposed modularized structure of the library will show its advantages. Each process device of the plant will be built in ProMoT from more simple modules and each device is then progressively coupled until the whole plant is assembled.

The reactive distillation column, `k1`

The modularized construction of the stripping section of column `k1` is shown in Fig. 5.2. A single reactive tray is composed by modules that evaluate physical properties, and are

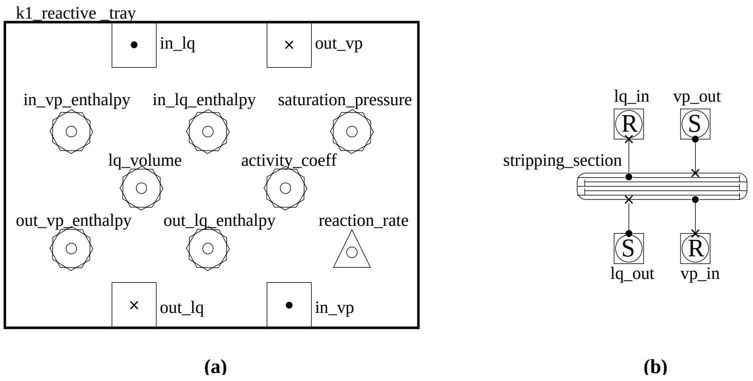


Figure 5.2: (a) A single reactive tray of the stripping section of column k1. The icons represent aggregated modules for the evaluation of physical properties. (b) Closed model for the stripping sequence of trays of column k1. The sequence is built coupling single reactive trays.

represented with the correspondent icons in part (a) of the figure. A single tray can subsequently be aggregated to create a complete sequence of reactive trays, as represented in Fig. 5.2(b), which shows a closed model of the stripping section. These coupled modules can now be compiled and be simulated in DIVA. Before the compilation, no data structures exist yet. All variables of each tray of the stripping section will be created after the compilation process. Values of the variables of the single tray are assigned to the numerical arrays created when the model is compiled. Reservoirs of Fig. 5.2(b) represent streams of liquid coming from the top and vapor coming from the bottom of the section, hence the state of these streams should correspond to saturated fluids. Modules for the evaluation of vapor-liquid equilibrium described in appendix C were used to calculate the state of the streams of the reservoirs of the intermediate models required for the construction of the plant. Simulating the stripping section of Fig. 5.2(b) allows to create a profile of the different variables along the section. These data will be loaded together with other parts of the

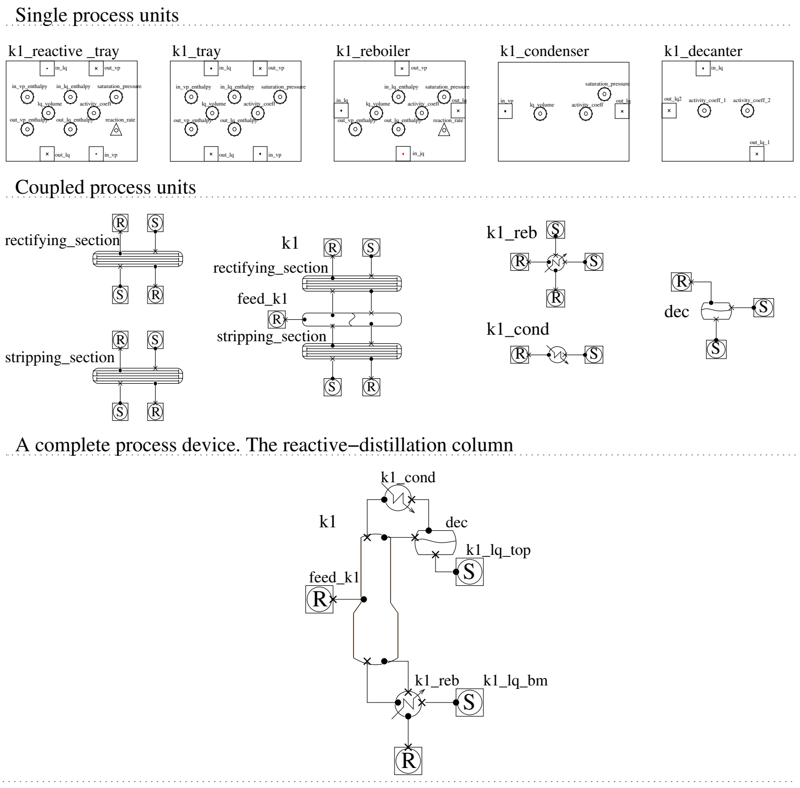


Figure 5.3: Modularized assemble of the reactive-distillation column.

column till the complete column is assembled. Fig. 5.3 illustrates the whole assembling process for column k1. In the first row of the figure, single process units are sketched. In the second row single modules are represented as closed modules in ProMoT. Consistent values are then created for each of these modules and subsequently be read progressively to built the whole column, which is shown in the last row of Fig. 5.3 as a closed model.

The complete plant

The same progressive procedure was followed to assemble columns k2 and k3.

Each column of the plant is shown as a closed model in the first row of Fig. 5.4, where the assembling process of the whole plant is sketched. In the sketch of the second row, columns k1 and k3 are assembled together. The state variables set of this intermediate plant was created reading the values created independently for each column. The next intermediate plant of this second row shows the aggregation of column k2. Note that recirculations between columns are not linked in this stage. Recirculations between columns have resulted to be problematic when trying to evaluate an initial state. Recirculations were finally linked at the end of the assembling process as show in the third row of Fig. 5.4.

Going back to the first levels of aggregation, changes can be performed in the most simple modules, for example, those for the evaluation of physical properties and these changes propagate to other modules. Other kinds of changes are performed with different flexibility along the modeling process. Note for example, that the liquid stream at the top of column k3, a stream reach in butanol, is sent back to the stripping section of column k1. To feed this stream into column k1, a feed tray has to be aggregated in the stripping section. But the assembled feed tray was not considered during the previous assembling process as shown in all the intermediate modules of column k1 of Fig. 5.4. This last change had to be done at the aggregation level of the plant. Of course, changes at this level imply that more variables are affected at once depending of the importance of the changes. Changes at any level of aggregation are possible, but it is advisable to perform them at low levels of aggregation.

5.1.2 Simulation scenario

As a simulation scenario, a 30% increase of the molar feed flow is considered. Figures 5.5 to 5.8 show the transient response of the most important devices of the plant after the

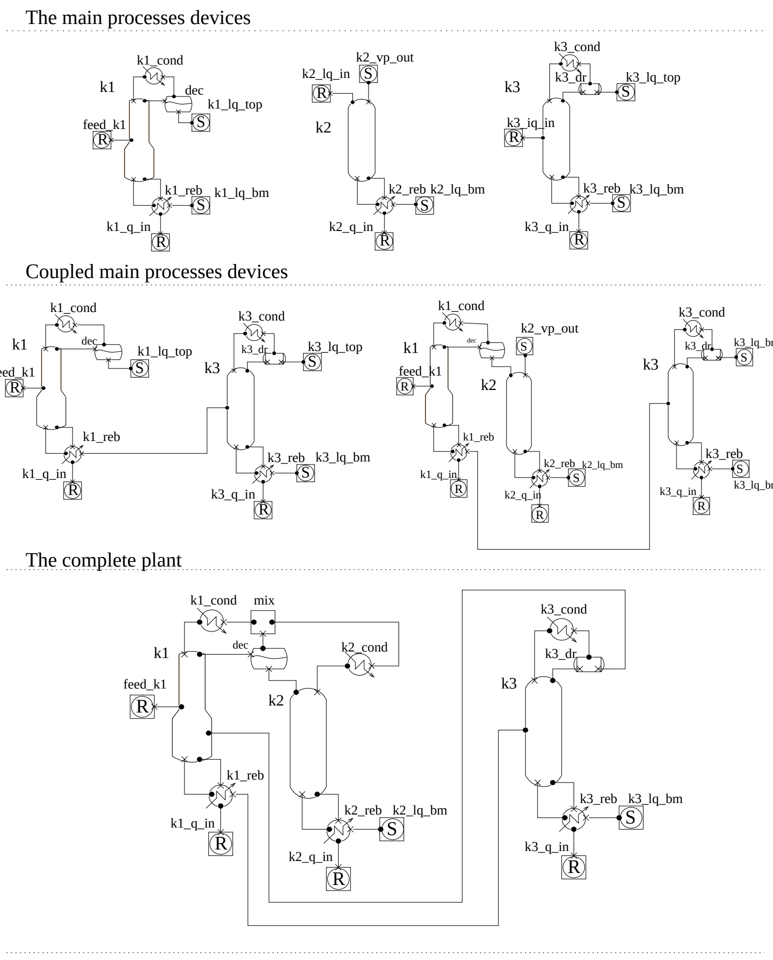


Figure 5.4: Modularized assemble of the plant.

stream of the equimolar mixture of acetic acid and n-butanol in the main feed of column `k1` is increased.

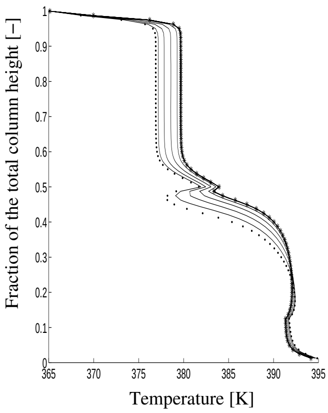


Figure 5.5: Transient profiles of temperature in the reactive column, `k1`, after the disturbance.

Black stars identify the initial steady state profile, while the final steady state is indicated with a thin doted curve. Fig. 5.5 shows that the temperature decreases from the initial to the final steady state in the rectifying section of `k1` and in the stripping section from the feed tray (located in the 50% of the total column height) to the 25% height of the column. Since the temperature in the trays with and without reaction is determined by the vapor liquid equilibrium relations, the temperature diminution can be explained by composition changes in the mixture along `k1`. In the rectifying section, water composition increases from the initial to the final state as shown in Fig.5.6(a) while n-butanol composition, the most abundant component in this section of the column, decreases, causing a diminution in the boiling point (Fig.5.6(b)). In the stripping section, the presence of butyl acetate becomes more important as we approximate towards the bottom, but also decreases after the perturbation, causing a temperature decreases from the feed tray to almost 25% height of the column. The butyl acetate composition augments till the bottom of the column and so does the temperature. Figures 5.6(c) and 5.6(d) show that the liquid and vapor molar flows also increase. The increase of the liquid flow in the stripping section of `k1` is more important than the increase in the vapor flow since the feed consists of a saturated liquid that flows down along the stripping section.

Figures 5.7(a) and (b) show that in the rectifying section of `k3` the temperature and com-

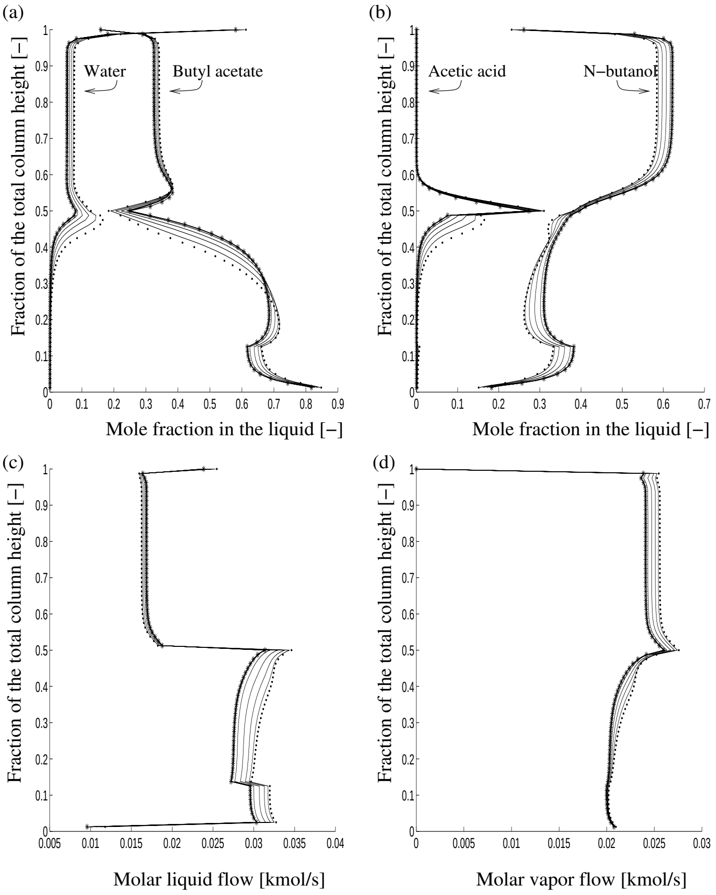


Figure 5.6: Composition and molar flow profiles of the reactive column, k1, after the disturbance.

position profiles remain almost constant after the disturbance with a light tendency to augment. However, the response of the stripping section shows significant temperature and butyl acetate composition increases starting from the feed tray to 30% height of the

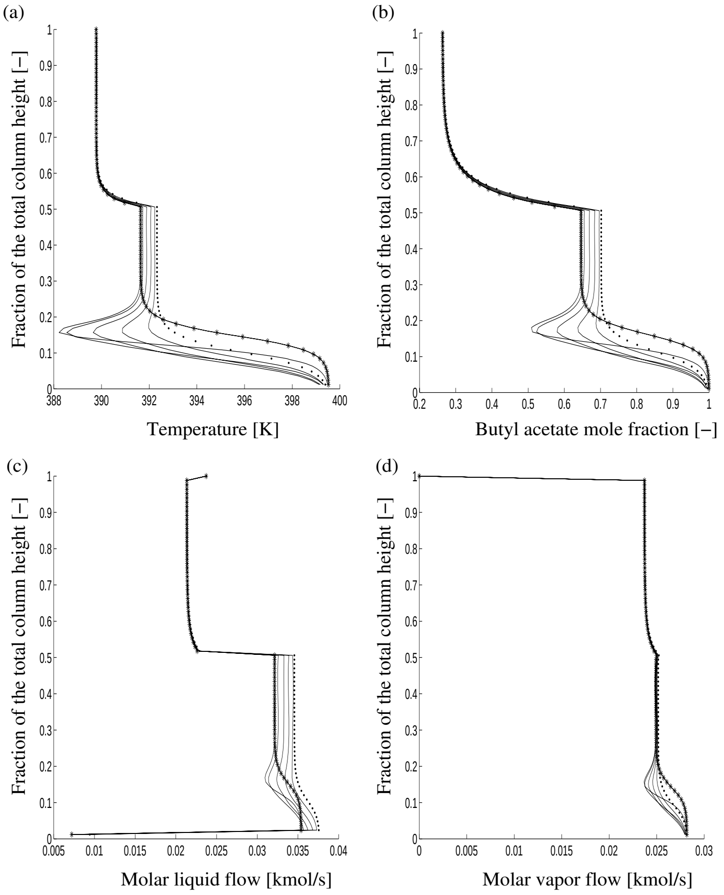


Figure 5.7: Transient profiles after the disturbance in the column for the purification of butyl acetate, `k3`.

column. Between 30 to 20% height an oscillating behavior occurs (the temperature and butyl acetate composition first decrease, but in the final state these variables are greater than those in the initial state). The butyl acetate composition and temperature profiles in the final state are lower till the end of the column. The final liquid flow in the stripping section, as shown in Fig. 5.7(c), increases after the disturbance. As a consequence, the production on butyl acetate also augments.

Fig. 5.7 (d) shows that the molar vapor flow does not exhibit important changes. Since the vapor flow is calculated with the energy balance, this result allows to conclude that the energy balance can be eliminated and set the vapor flow constant if some model simplification were required.

Fig 5.8 keeps track of the response of the decanter, `dec`, during intermediate time stages

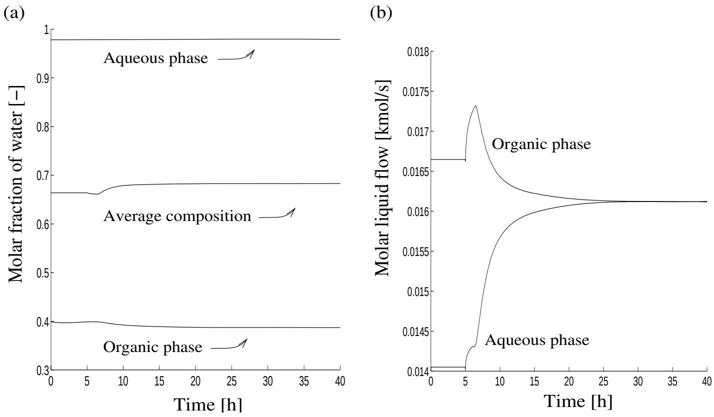


Figure 5.8: Composition and molar flow variations in the decanter, `dec`, in different time stages between the first and final states.

before reaching the final steady state. As expected, the azeotropic concentration of the water-n-butanol vapor stream at the top of column `k1`, ensures that the decanter operates in the liquid-liquid equilibrium zone. Since the composition of the aqueous phase in the decanter remains practically constant, the behavior in the water treatment column, `k2` shows no important changes. The increase of the molar feed in the plant also causes an increase of the production of water as shown in molar flow of the aqueous phase of Fig. 5.8(b).

The following section describes a second important application of the library proposed in this thesis.

5.2 The plant for the production of acetic acid

The flow sheet of the plant for the production of acetic acid depicted in Fig. 5.9 represents

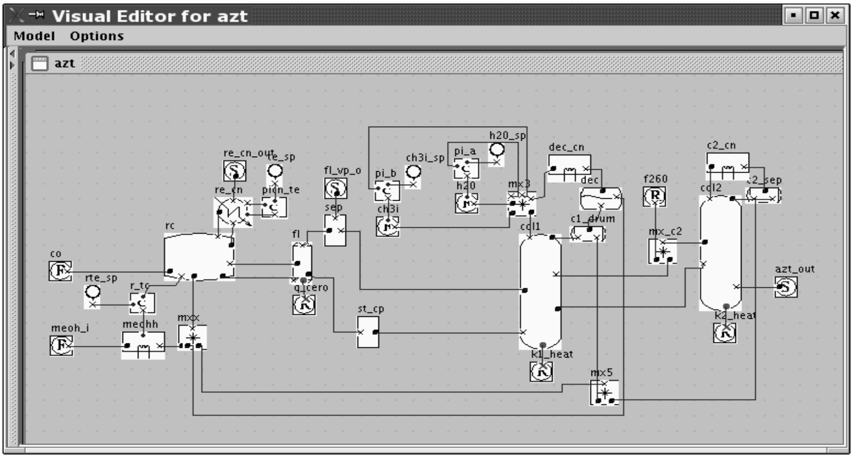


Figure 5.9: Representation of the flow diagram for the plant for the production of acetic acid using the graphical editor of ProMoT.

the model of the process, assembled with the graphic editor of ProMoT. In this continuously operated process (Waschler et al., 2002), acetic acid is produced through the carbonylation of methanol according to the exothermic reaction:

$$CH_3OH + CO \rightarrow CH_3COOH. \qquad (5.2)$$

Gaseous carbon monoxide and liquid methanol are fed to the, evaporatively cooling reactor `rc`. A saturated liquid stream from the reactor flows to the adiabatic flash tank, `fl` in which the non-volatile rhodium based catalyst is completely recycled to the reactor. The reactor and the flash tank constitute the reaction system. In this part of the plant, calculations are carried out with five chemical species (water, methanol, acetic acid, methyl iodide and carbon monoxide).

The liquid separation system consists of two distillation columns, `col1` and `col2`, where the purification of acetic acid takes place and where the catalyst promotor, methyl iodide, and water, used as solvent, are recovered. For simplicity, in the separation system only three chemical components are considered: water, acetic acid and methyl iodide.

The reaction and separation systems are coupled with recycle streams coming from the top of both columns and collected in mixer `mx5` and sent back to mixer `mxx` before being fed to the reactor.

The modularized construction of the model of the plant followed a similar methodology as described for the plant for the production of butyl acetate in section 5.1.1. Hence, no more about the individual assemble and progressive aggregation of the different modules and devices of this plant will be discussed. Instead, more detail will be devoted to the discussion of the reaction system of the plant, which is depicted in Fig. 5.10 as an isolated model.

5.2.1 The reaction system of the plant

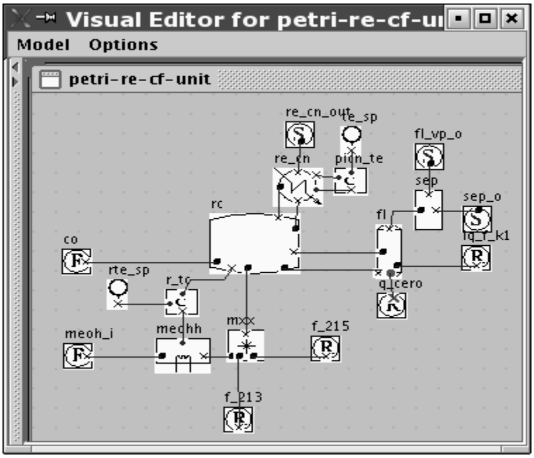


Figure 5.10: Representation of the reaction system of the plant as an isolated model using the graphical editor of ProMoT.

The temperature of the reactor is controlled with the temperature controller `r_tc`[1], which

[1]reactor temperature controller

gets the value of the current temperature of the reactor and determines the temperature of the methanol stream with the preheater `meohh`.

The data input module `rte_sp`[2], fixes the temperature set point of the reactor. Condenser `re_cn` is also temperature controlled with controller `picn_te`. Module `sep` serves to separate the non condensable carbon monoxide to the system. Reservoirs of streams coming from the separation system were implemented in such a way that each of them represents the most expected state of these streams.

During the problem statement, the importance of modeling the potential phase changes in the plant was set. Under standard operating conditions in the reactor, a liquid in thermodynamic equilibrium with the vapor should be found. However, temperature reductions

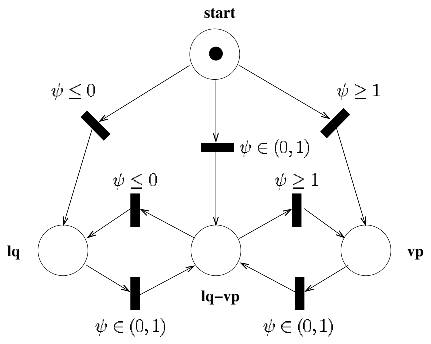


Figure 5.11: Petri network used to represent the different possible vapor-liquid operation regimes in the reactor.

caused by different disturbances, may bring about the system to switch to the pure liquid regime. This behavior is modeled as an implicit discrete event. The Petri network (Baumgarten, 1992) sketched in Fig. 5.11 was used to represent the different states of the reactor. In the Petri network, the states are represented with circles that indicate the current vapor-liquid regime of the reactor. The initial position is distinguished with a black labeled circle and represents the **start** position. The network represents transitions between one of three different possible states in the reactor:

1. **lq**: only liquid

[2]reactor temperature set point

2. **lq-vp**: vapor-liquid equilibrium

3. **vp**: only vapor

The first and second states were considered the most relevant, while the third one was not considered in the model. The state of the system is determined by the transition function defined with the vapor fraction $\psi = n''/(n' + n'')$ and represented in Fig. 5.11 with black blocks and the value of the function that triggers a transition.

5.2.2 Simulation scenario

A 10% increase of the methanol molar feed, followed by a decrement back to the same value serves as a simulation scenario. Fig. 5.12 shows simulation results of different devices of the the model of the reaction system of the plant. The simulation starts (time, $t = 0$) with a steady state and vapor-liquid equilibrium in the reactor (regime 1 in Fig. 5.12(b)). After an hour passed, the system is disturbed by a 10% increase in the methanol molar feed. After three hours the system reaches a new steady state and the molar feed of methanol is reestablished to the value after the first disturbance. The system reaches again the steady state after eight hours. As expected, because of the exothermic reaction ($\Delta H = -138.6 kJ/mol$), the temperature increases abruptly from the desired reactor temperature of 458 K to almost 466 K as shown in Fig. 5.12(a). The temperature controller reacts decreasing the methanol preheater temperature to the minimum allowed value (300 K) (Fig. 5.12(d)), causing that some minutes after the methanol flow increase, the vapor phase disappears in the reactor (regime 0 in Fig. 5.12(b)) and no vapor flow leaves the reactor as shown in Fig. 5.12(d). The one phase regime remains during the next three hours.The methanol feed flow is then reestablished to the original value and some minutes after this second perturbation, the vapor phase reappears in the reactor. Fig. 5.12(c) shows the dynamic response of the vapor flow leaving the reactor. As expected, the vapor flow goes to zero during the one-phase regime. In Fig. 5.13 results of the same disturbances considering now the whole plant are presented. In this case the dynamic response of the plant is significantly slower. The reestablishment of the original methanol molar flow had to be carried out in two small steps, otherwise, numerical instabilities were encountered and the integrator could not calculate the next step. Note that the reactor switches as expected for the two-phases to the one-phase regimes. As in the previous model with the reactor in the chemical system, a disturbance occurs after an hour. As shown in Fig. 5.13(a), the reactor requires more than five hours to reach a new steady state. Six hours

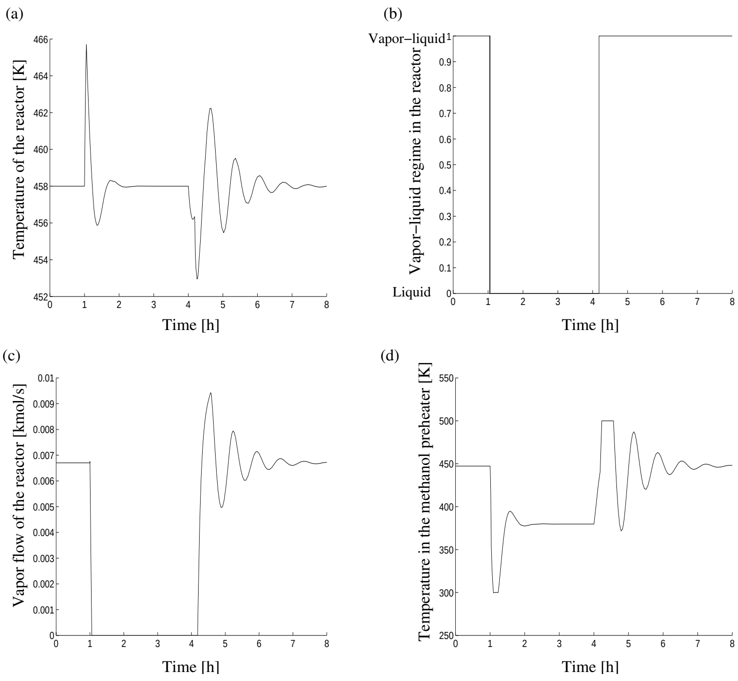


Figure 5.12: Response of the model of the reactor in the chemical system of Fig. 5.10.

after, the methanol feed is reduced 5% of the flow at $t = 0$ and passed hours the original flow is reestablished.

5.3 Summary

The library proposed in this work was applied for modeling two different industrial chemical processes using the same set of modules. Modeling details of both processes were omitted for reasons of space. The objective of this chapter is to show concrete features of the reusability of modules of the library.

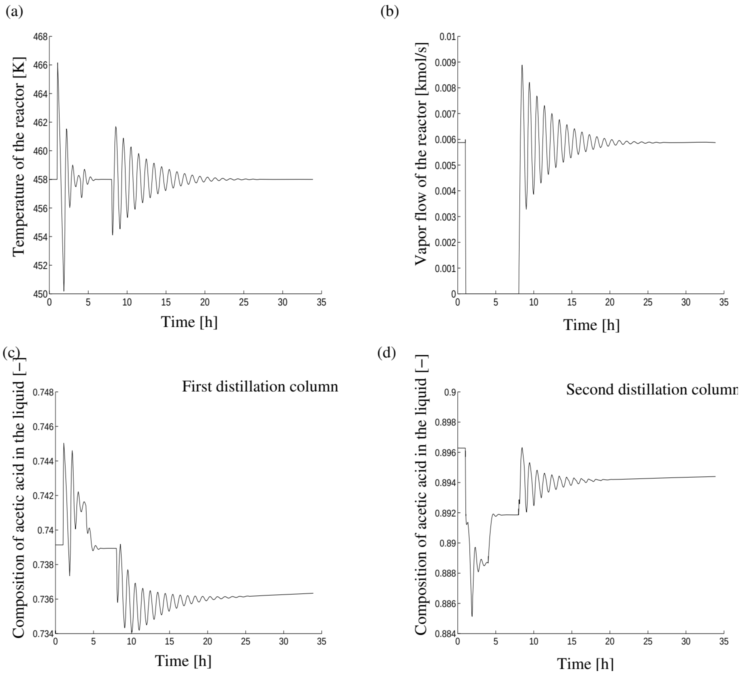


Figure 5.13: Response of the model considering the complete plant. Parts (c) and (d) show the composition of acetic acid in the feed trays of the first and second column respectively.

Chapter 6

Conclusions and outlook

A library of modules that supports computer-aided modeling of chemical processes is proposed in this work. Focus of this contribution is on lumped systems described by ordinary differential equations of first order and algebraic equations for vapor-liquid systems.

Modeling is presented as a common activity in different scientific disciplines, with special emphasis on modeling in chemical engineering. It is proposed that the computer-aided modeling task can be divided in two parts, namely the conceptual formulation, that corresponds to the physical description of the system in mathematical terms, and its implementation in a programming language. Even though both parts are interwoven, distinguishing between them is important. It allows to develop strategies to tackle difficulties when dealing with the physical description of a system and its translation in computational terms. The network approach of Gilles is considered a conceptual structuring approach, which can be used as a guideline for capturing the physical description of a system.

Regarding the problem that constitutes the computational implementation of mathematical models, domain engineering activities are discussed, namely analysis domain and architecture development. These activities lead to the software pattern proposed in this thesis and applied for the implementation of the library. The discussion emphasizes that modeling chemical engineering systems by reusing modules of a library is different than modeling from scratch and that preparing modules for reuse is a process on its own. In this context it was shown how computer-aided modeling impacts the whole modeling task.

We take advantage of combining the object-oriented programming paradigm with the equation-based approach. As a result, a class comprises a set of equations which corresponds to real-world entities that can be manipulated as modules. The final result of coupling these modular entities is again a set of equations, but on its formulation we were assisted by a computer tool.

The object-oriented modeling tool ProMoT and its Model Definition Language, MDL, were used for the implementation of the library. When we proceed with the computer implementation of a model, also a notation was used to identify software patterns for constructing, visualizing and documenting the implementation of chemical processes. This ”language” consists of a set of notations and conventions used to record analysis and design decisions in object-oriented environments, the so called *Unified Modeling Language*, UML. The representation of the class architecture of modules and different kinds of relations between them using UML, allowed us to find architectural patterns that emerged while setting up two important kinds of relationships between classes, namely, inheritance and aggregation. The proposed software pattern, a result of domain engineering is applied for the implementation of the library.

The modeler of chemical process should find a conceptually well structured knowledge space in which entities can be systematically found, used, and further developed. For that, notational aspects for creating names are considered, a conceptual framework is developed to set phenomenological assumptions for the formulation of mass and energy balances, which find a very wide spectrum of application. This conceptual frame is proposed as a starting point for more detailed phenomenological developments of the library.

Although the library is a final product itself, its development would be pointless without applications associated with it. The applications consist of two different industrial chemical processes: the first one is a plant for the production of butyl acetate and the second one is a plant for the production of acetic acid. Simulation scenarios serve to show the response of the plants. Each application is a completely independent project with the library of models as a common root. It is shown that the library can be adapted to particular requirements of the modeler and it can be further developed by either creating new modules or by modifying some others. After a validation and reuse analysis, these new models can be added to the library to make them available for different users.

From the modeling point of view, much more efforts are required regarding the automatic processing of knowledge. The user of knowledge, the modeler in our case, has to spend a long time systematizing the information processing for making it reusable. Hence, the development of automatic knowledge processing tools remains as a very promising challenge.

Extensions of the library to non-equilibrium vapor-liquid models seems to be the most natural future step. This kind of models require the definition of new entities for the evaluation of phenomenological coefficients and with that, the extension to modules that take into account hydrodynamic Al aspects will be prepared.

Appendix A

A simple standard mass-energy balance formulation as a phenomenological framework

A.1 Balance equations

If the possibility of setting up general balance expressions would exist, the model implementation would be considerably simplified, by just reusing the general expressions in different situations. However, given the diversity of processes, setting up general material, energy or momentum balances is not feasible.

Nonetheless, several specific balance equations can be formulated for various applications. In this section, a set of assumptions applicable to a wide spectrum of the chemical engineering domain is discussed. These assumptions find application on processes in which vapors and liquids are processed. If in different processes the same set of assumptions remains valid, we can use the balance equations deduced up to the same assumptions. If in a particular modeling project some of the assumptions are not valid any more, the modeler will be able to perform the pertinent modifications. Even in this case, the modeler does not have to set up the model of a new physical problem from scratch .

Balance equations usually contain the differential terms of dynamic models. Here are presented some conventions concerning the arrangement of the balance equations: It is customary, to write time derivatives of dependent variables on the left hand side of equations and write terms for the flows and sink/sources on the right. This fact helps to simplify the manipulation of differential equations. The set of abbreviations proposed by Waschler (1996) for facilitating the manipulation of the algebraic terms of differential equations was used. Right hand sides of dynamic equations are then expressed with the notation shown in table A.1. The use of this notation will be clearer in the following two sections

Table A.1: Notation for the right hand side (*rhs*) of equations of balance.

Description	Notation
Total material balance	`rhs_tmb`
Component material balance	`rhs_cmb`
Energy balance	`rhs_eb`

that deal with mass and energy balances.

A.1.1 Material balance

Rigorously, the molar holdup of the vapor phase has to be taken into account and a mass balance for the vapor should be performed. However, neglecting the vapor holdup allows the calculation of the vapor flow without having to consider the pressure of the system. For that, a momentum balance would be required or the introduction of an empirical correlation that gives the value of the pressure as a function of operational conditions like the open span of valve, for example.

From low to moderate pressures, neglecting the vapor holdup is, hence, a reasonable assumption. Table A.2 lists assumptions for setting families of models in which attention is paid to the liquid phase.

Assumption 1 of table A.2 implies that no temperature, concentration or pressure gradients will be found in the liquid control volume and the geometry of the volume in not

Table A.2: Assumptions for setting up the material balance in the liquid phase.

1. The content of the control volume is perfectly mixed
2. The fluid is considered to be incompressible
3. Mixing effecs are neglected

important, hence the total volume should be given as a parameter. Assumption 2 requires that $\left(\frac{\partial \rho'}{\partial P}\right)_{T'} = 0$. Assumption 3 implies that no mixing effects in the molar volume occur. This requires that the partial molar properties of the species in solution are equal to the properties of the pure species at the same temperature and pressure.

Following the simple-to-complex strategy, a very simple process (Fig. A.1) is used to introduce notational aspects and the development methodology of models in the library.

Consider the sketch of the isothermal liquid stream separator of Fig. A.1, in which the

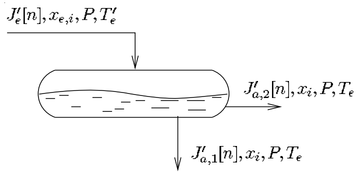

$J'_e[n], x_{e,i}, P, T'_e$
$J'_{a,2}[n], x_i, P, T_e$
$J'_{a,1}[n], x_i, P, T_e$

Figure A.1: Tank drum.

notation in the text is used to show the names of variables. The total material balance of the tank reads

$$\frac{dn'}{dt} = J'_e[n] - J'_{a,1}[n] - J'_{a,2}[n].$$

Substituting the algebraic terms of the right hand side with the abbreviation introduced in table A.1 for the right hand side of the total material balance we have,

$$\frac{dn'}{dt} = rhs\_tmb. \tag{A.1}$$

On the other side, balance of component i reads,

$$\frac{d(n'x_i)}{dt} = J'_e[n]x_{e,i} - (J'_{a,1}[n] + J'_{a,2}[n])x_i.$$

Again abbreviations of table A.1 are used for the right hand side of the component material balance:

$$\frac{d(n'x_i)}{dt} = rhs\_cmb_i. \tag{A.2}$$

Expanding the derivative on the left side of the last equation:

$$n'\frac{dx_i}{dt} + x_i\frac{dn'}{dt} = rhs\_cmb_i.$$

Then, expressing $\frac{dn'}{dt}$ of the last equation as the right hand side of Eq. (A.1) and rearranging we get,

$$n'\frac{dx_i}{dt} = rhs\_cmb_i - x_i rhs\_tmb.$$

According to the ideal solution assumption, the molar volume of the solution can be calculated from the molar volume of the pure components, $v_i'^0$ i.e. $v' = \sum_{i=1}^{nc} x_i v_i'^0$. For calculating the molar volume of the pure components an empirical correlation as a function of the temperature can be used.

Hence, substituting in the last equation n' by the relation $\frac{V}{v'}$ we get,

$$\frac{V}{v'}\frac{dx_i}{dt} = rhs\_cmb_i - x_i rhs\_tmb. \tag{A.3}$$

Eq. (A.3) expresses the component material balances as a function of the composition, the total volume as a parameter, and the molar volume of the ideal mixture, which is a function of temperature.

The form of Eq. (A.3) can be reused for processes in which assumptions of table A.2 remain valid. An example is a tank with more than one input and output streams like that shown in figure A.2. If more streams are added to the tank, the contribution of the

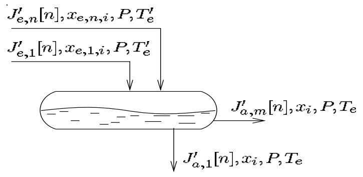

$J'_{e,n}[n], x_{e,n,i}, P, T'_e$
$J'_{e,1}[n], x_{e,1,i}, P, T'_e$
$J'_{a,m}[n], x_i, P, T_e$
$J'_{a,1}[n], x_i, P, T_e$

Figure A.2: Tank drum with multiple input/output streams.

new streams should be considered in the definition of the right hand sides of the total and component material balances. For a tank with *es* input streams and *as* output streams the right hand side of the total material balance reads,

$$rhs\_tmb = \sum_{k=1}^{es} J'_{e,k}[n] - \sum_{j=1}^{as} J'_{a,j}[n]$$

while for the right hand side of the component material balance of component *i* we have,

$$rhs\_cmb_i = \sum_{k=1}^{es} J'_{e,k}[n] x_{e,k,i} - x_i \sum_{j=1}^{as} J'_{a,j}[n].$$

Equation, names, and the defined variables remain the same, allowing us to take advantage of reusing all attributes previously defined for the case of the most simple case of figure A.1.

Reactive systems

In the above described processes also an isothermal chemical reaction can be included in the process description. If assumptions set by table A.2 remain valid, the generation of moles of species i by chemical reaction, $J'_g[n_i]$ should be considered and the right hand side of the total material balance including chemical reaction, is:

$$rhs\_tmb = \sum_{k=1}^{es} J'_{e,k}[n] - \sum_{j=1}^{as} J'_{a,j}[n] + \sum_{i=1}^{nc} J'_g[n_i] \tag{A.4}$$

and the right hand side of the component material balance for component i, including chemical reaction is

$$rhs\_cmb_i = \sum_{k=1}^{es} J'_{e,k}[n]x_{e,k,i} - x_i \sum_{j=1}^{as} J'_{a,j}[n] + J'_g[n_i]. \tag{A.5}$$

In summary, a simple-to-complex construction of structurally similar balance equations has been performed: We started from a very simple mixer tank, continued to a tank with multiple input/output streams and finally we included chemical reaction. This conceptual simple-to-complex building approach was also applied on the implementation of modules in our library, which has the following advantages:

- The engineer follows a logical sequence of phenomenological development that is, the structuring of information follows a conceptual progression. This implies that the engineer can get acquainted with the development process of modules and understand their physical meaning.
- Important decisions about the validity of the assumptions behind modules can be taken at any point of the development process and new branches of model families can be created.
- The development process is simplified, since conceptual, structural, and notational aspects are systematically reused.

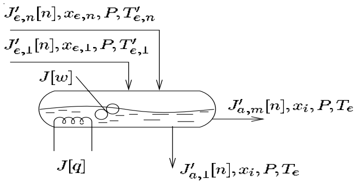
Along the phenomenological sequence of development, the consideration of the energy balance is the next step.

A.1.2 Energy balance

A simple-to complex strategy was also applied to set up a reusable conceptual frame in the case of the energy balance.

We start with a control volume that consists of a liquid phase in which the assumptions of table A.2 remain valid. The liquid container like that shown in figure A.3 which is very similar to that one shown in the last section, includes transport of heat, $J[q]$, and mechanical work, $J[w]$, into the control volume. Note also that the input streams are characterized by different temperatures.

Within the phenomena we are interested in, total energy of the system, U consists of at least three different kinds of contributions. In a molar basis: internal energy u, kinetic energy, $\frac{1}{2}c^2$, and potential energy, gz. Hence, the variation of the total energy with time in



$J'_{e,n}[n], x_{e,n}, P, T'_{e,n}$
$J'_{e,1}[n], x_{e,1}, P, T'_{e,1}$
$J[w]$
$J'_{a,m}[n], x_i, P, T_e$
$J[q]$
$J'_{a,1}[n], x_i, P, T_e$

Figure A.3: Tank drum with multiple input/output streams, heat and mechanical work transport.

the system is

$$\frac{dU'}{dt} = \frac{d}{dt}\left(n'(u' + \frac{1}{2}c'^2 + gz)\right). \tag{A.6}$$

The total energy balance of our liquid control volume reads,

$$\frac{dU'}{dt} = \sum_{j=1}^{es} J'_{e,j}[n](u'_e + \frac{1}{2}c'^2_e + gz_e)_j - \sum_{k=1}^{as} J'_{a,k}[n](u'_a + \frac{1}{2}c'^2_a + gz_a)_k + J[q] + J[w]. \tag{A.7}$$

Mechanical work, $J[w]$, consists of two contributions: the shaft work $J[ws]$, due to agitation for example, and the energy required to get the fluid into the system and the energy to force the fluid out. The later contribution is expressed by the product of the pressure and the molar volume of the fluid, Pv. Therefore, the total mechanical work is,

$$J[w] = J[ws] + \sum_{j=1}^{es} J'_{e,j}[n](Pv'_e)_j - \sum_{k=1}^{as} J'_{a,k}[n](Pv'_a)_k. \tag{A.8}$$

We require to simplify the very general energy balance of Eq. (A.6) and hence introduce first assumptions for the energy balance. For the systems considered in this work, contributions of the kinetic and potential energy can be neglected. Eliminating these two contributions from Eq. (A.6) and substituting the mechanical work contribution by Eq. (A.8) and arranging we get,

$$\frac{dU'}{dt} = \sum_{j=1}^{es} J'_{e,j}[n](u'_e + Pv'_e)_j - \sum_{k=1}^{as} J'_{a,k}[n](u'_a + Pv'_a)_k + J[q] + J[ws]. \tag{A.9}$$

Now we use the definition of enthalpy, $h = u + Pv$, in both sides of the last equation to get an expression of the energy balance as a function of the enthalpy,

$$\frac{d}{dt}\left(n'(h' - Pv')\right) = \sum_{j=1}^{es} J'_{e,j}[n]h'_{e,j} - \sum_{k=1}^{as} J'_{a,k}[n]h'_{a,k} + J[q] + J[ws]. \tag{A.10}$$

Eq. (A.10) was obtained considering a liquid phase, but the assumptions taken so far make that this expression can also be applied to a gas phase. For the case of liquids, the

Table A.3: Set of assumptions for the formulation of the energy balance

1. The fluid in the control volume is perfectly mixed
2. Mixing effects are neglected
3. Kinetic energy is negligible
4. Potential energy is negligible
5. The differential $d(Pv')$ is negligible

differential $d(Pv')$ is usually neglected. Hence, the left hand side of Eq. (A.10) can be expressed as a function of the total enthalpy H. Using the abbreviation for the energy balance for the algebraic terms of the right hand side we get:

$$\frac{dH'}{dt} = rhs\_eb. \tag{A.11}$$

Table A.3 summarizes the assumptions taken so far for the formulation of the energy balance.

The energy balance expressed as Eq. (A.11) adds a new unknown variable, the total enthalpy, H'. More transformations of this equation are required for expressing the energy balance as a function of temperature, which is an intensive variable for which initial values are easier to set than initial values for the total enthalpy of the system. For that purpose, we will express the total enthalpy as a function of the temperature, the pressure and the number of moles, i.e. $H' = H'(T,P,n'_i)$. The total differential of H' as a function of these variables reads

$$\frac{dH'}{dt} = \left(\frac{\partial H'}{\partial T}\right)_{P,n'_i}\frac{dT}{dt} + \left(\frac{\partial H'}{\partial P}\right)_{T,n'_i}\frac{dP}{dt} + \sum_{i=1}^{nc}\left(\frac{\partial H'}{\partial n'_i}\right)_{T,P,n'_{j\neq i}}\frac{dn'_i}{dt}. \tag{A.12}$$

Starting from left to right of Eq. (A.12), we find:

$\left(\frac{\partial H'}{\partial T}\right)_{P,n'_i}$ is the definition of the heat capacity, C_p (written with capital "C" indicates, that it is heat capacity of the bulk liquid phase i.e. $n'c_p$).

The partial derivative $\left(\frac{\partial H'}{\partial P}\right)_{T,n'_i}$ can be expressed as a function of the total molar volume, $n'v$ and the temperature, according to the derivation that can be found in a standard textbooks (e.g. Stephan and Mayinger (1998), p. 278),

$$\left(\frac{\partial H'}{\partial P}\right)_{T,n'_i} = n'v' - T\left(\frac{\partial(n'v')}{\partial T}\right)_P. \tag{A.13}$$

This expression consists of variables that can be measured or calculated by empirical correlations. However, we do not consider more details in this case because the second partial derivative of Eq. (A.12) will be eliminated if we consider that for our purposes, the pressure dynamics will not be important in the liquid phase. Therefore we set $\frac{dP}{dt} \approx 0$.

The remaining term of Eq. (A.12) contains the definition of the partial molar enthalpy, which should be taken into account if mixing thermal effects of the components are important. Assuming an ideal solution, the partial molar enthalpy of component i in the mixture is equal to the molar enthalpy of the pure component, $h_i'^0$.

After the above considerations about the partial derivatives of Eq. (A.12), we get,

$$\frac{\partial H'}{\partial t} = n'c_p\frac{dT}{dt} + \sum_{i=1}^{nc} h_i'^0\frac{dn'_i}{dt}. \tag{A.14}$$

Eq. (A.14) expresses the total molar enthalpy of the liquid phase as a function of intensive variables. The dynamic change of the number of moles $\frac{dn'_i}{dt}$ can be substituted by the abbreviation of the right hand side of the component material balance:

$$\frac{\partial H'}{\partial t} = n'c_p\frac{dT}{dt} + \sum_{i=1}^{nc} h_i'^0 rhs\_cmb_i. \tag{A.15}$$

It is important to notice that we have not performed an "enthalpy balance", an idea that is frequently referred erroneously in the literature. For example, Mäkilä and Waller (1981) explain how the ”enthalpy balance” idea can lead to errors modeling the gas phase reactor dynamics.

Eq.(A.15) can be used for the left hand side of Eq. (A.10). Using the equivalent form Eq. (A.11) and rearranging we have:

$$n' c_p \frac{dT}{dt} = rhs\_eb - \sum_{i=1}^{nc} h_i'^0 rhs\_cmb_i. \tag{A.16}$$

In summary, Eq. (A.16) is the energy balance as a function of the liquid phase temperature in which the assumptions of table A.3 are valid.

Balance equations are corner stones for the conceptual constructions of models. With equations (A.3) and (A.16) we have developed a phenomenological framework that can be reused in processes in which the physical conditions listed in tables A.2 and A.3 are valid. Even when some of these assumptions are not valid anymore, the modeler can perform the appropriate transformations and take advantage of the implemented entities when possible, perform refactorization as described in page 37, or create completely new branches of models that can be described with new assumptions. For example, if we consider that mixing effects become important for an application, the contribution of these non-idealities can be added to the right hand side of the material and energy balances. Subclasses derived for this purpose would include only the additional variables and correlations for taking into account mixing effects of the solution. However, regarding this example, the class structure would essentially remain the same and the modeler would not have to develop a model from scratch.

Appendix B

Degree of freedom analysis for selected process entities and modules

Table B.1: Input liquid reservoir.

Parameters	**Notation in the text**	**Number of parameters**	**Units**
Number of components	nc	1	[-]
Molar flow	pjn	1	kmol/s
Temperature	pte	1	K
Composition	px_i	nc	[-]
Input variables		**Number of input variables**	
Pressure	P	1	bar
Output variables		**Number of output variables**	
Liquid molar flow	$J'[n]$	1	kmol/s
Temperature	T	1	K
Composition	x_i	nc	[-]
Total number of output variables		nc+2	
Equations		**Number of equations**	
Molar	$J'[n] = pjn$	1	
Temperature	$T = pte$	1	
Pressure	$x_i = px_i$	nc	
Total number of equations		nc+2	

Table B.2: Vapor sink.

Parameters	**Notation in the text**	**Number of parameters**	**Units**
Pressure	p_p	1	bar
Input variables		**Number of input variables**	
Molar flow	$J''[n]$	1	kmol/s
Temperature	T	1	K
Composition	y_i	nc	[-]
Output variables		**Number of output variables**	
Pressure	P	1	bar
Equation		**Number of equations**	
Pressure	$P = p\_p$	1	

Table B.3: Total condenser.

Input variables	**Notation in the text**	**Number of input variables**	**Units**
Vapor molar flow	$J''_e[n]$	1	kmol/s
Inlet vapor temperature	T''_e	1	K
Inlet vapor composition	$y_{e,i}$	nc	[-]
Outlet liquid pressure	P'_a	1	bar
Output variables		**Number of output variables**	
Liquid molar flow	$J'_a[n]$	1	kmol/s
Saturation temperature	T	1	K
Liquid composition	x_i	nc	[-]
Vapor composition	y_i	nc	[-]
Pressure	P	1	bar
Total number of output variables		2nc+3	
Equations		**Number of equations**	
Total material balance	$0 = rhs\_tmb$	1	
Component material balances	$\frac{V}{v'}\frac{dx_i}{dt} = rhs\_cmb_i - x_i rhs\_tmb$	nc-1	
Vapor-liquid equilibrium	$y_i P = x_i \gamma_i P_i^o$	nc	
Summation condition (a)	$\sum_{i=1}^{nc} x_i = 1$	1	
Summation condition (b)	$\sum_{i=1}^{nc} y_i = 1$	1	
Pressure loss	$delta\_p = P - P'_a$	1	
Total number of equations		2nc+3	

Table B.4: Partial condenser.

Input variables	**Notation in the text**	**Number of input variables**	**Units**
Vapor molar flow	$J''_e[n]$	1	kmol/s
Inlet vapor temperature	T''_e	1	K
Inlet vapor composition	$y_{e,i}$	nc	[-]
Cooling duty	$J_e[q]$	1	kJ/s
Outlet liquid pressure	P'_a	1	bar
Outlet vapor pressure	P''_a	1	bar
Output variables		**Number of output variables**	
Vapor molar flow	$J''_a[n]$	1	kmol/s
Liquid molar flow	$J'_a[n]$	1	kmol/s
Saturation temperature	T	1	K
Liquid composition	x_i	nc	[-]
Vapor composition	y_i	nc	[-]
Pressure	P	1	bar
Total number of output variables		2nc+4	
Equations		**Number of equations**	
Total material balance	$0 = rhs\_tmb$	1	
Component material balances	$\frac{V}{v'}\frac{dx_i}{dt} = rhs\_cmb_i - x_i rhs\_tmb$	nc-1	
Energy balance	$0 = rhs\_eb - \sum_{i=1}^{nc} h'_i rhs\_cmb_i$	1	
Vapor-liquid equilibrium	$y_i P = x_i \gamma_i P_i^o$	nc	
Summation condition (a)	$\sum_{i=1}^{nc} x_i = 1$	1	
Summation condition (b)	$\sum_{i=1}^{nc} y_i = 1$	1	
Pressure loss	$delta\_p = P - P''_a$	1	
Total number of equations		2nc+4	

Table B.5: Total reboiler.

Input variables	**Notation in the text**	**Number of input variables**	**Units**
Liquid molar flow	$J_e[n']$	1	kmol/s
Inlet liquid temperature	T'_e	1	K
Inlet liquid composition	$x_{e,i}$	nc	[-]
Pressure	P''_a	1	bar
Output variables		**Number of output variables**	
Vapor molar flow	$J_a[n'']$	1	kmol/s
Saturation temperature	T	1	K
Liquid composition	x_i	nc	[-]
Vapor composition	y_i	nc	[-]
Pressure	P	1	bar
Total number of output variables		2nc+3	
Equations		**Number of equations**	
Total material balance	$0 = rhs\_tmb$	1	
Component material balances	$\frac{V}{v'}\frac{dx_i}{dt} = rhs\_cmb_i - x_i rhs\_tmb$	nc-1	
Vapor-liquid equilibrium	$y_i P = x_i \gamma_i P_i^o$	nc	
Summation condition (a)	$\sum_{i=1}^{nc} x_i = 1$	1	
Summation condition (b)	$\sum_{i=1}^{nc} y_i = 1$	1	
Pressure loss	$delta\_p = P - P''_a$	1	
Total number of equations		2nc+3	

Table B.6: Detailed model and degree of freedom analysis of an elementary distillation tray.

Parameter name	**Notation in the text**	**Number of parameters**	**Units**
Pressure difference	$delta\_p$	1	bar
Total volume	V	1	m^3
Input variables		**Number of input variables**	
Molar flows	$J_e[n'], J_e[n'']$	2	kmol/s
Temperatures	T'_e, T''_e	2	K
Composition	$x_{e,i}, y_{e,i}$	2nc	[-]
Pressure	P'_a, P''_a	2	bar
Physical property name		**Number of physical properties**	
Vapor pressure	P_i^o	nc	bar
Molar volume	v_i^o	nc	$m^3/kmol$
Activity coefficient	γ_i	nc	[-]
Molar enthalpy for the vapors	$h'_{ei}, h'_i,$	2nc	kJ/kmol
Molar enthalpy for the liquids	$h''_{ei}, h''_i,$	2nc	kJ/kmol
Output variables		**Number of output variables**	
Molar flows	$J_a[n'], J_a[n'']$	2	kmol/s
Saturation temperature	T	1	K
Liquid composition	x_i	nc	[-]
Vapor composition	y_i	nc	[-]
Pressure	P	1	bar
Total number of output variables		2nc+4	
Equations		**Number of equations**	
Total material balance	$0 = rhs\_tmb$	1	
Component material balances	$\frac{V}{v'}\frac{dx_i}{dt} = rhs\_cmb_i - x_i rhs\_tmb$	nc-1	
Energy balance	$0 = rhs\_eb - \sum_{i=1}^{nc} h'_i rhs\_cmb_i$	1	
Vapor-liquid equilibrium	$y_i P = x_i \gamma_i P_i^o$	nc	
Summation condition (a)	$\sum_{i=1}^{nc} x_i = 1$	1	
Summation condition (b)	$\sum_{i=1}^{nc} y_i = 1$	1	
Pressure loss	$delta\_p = P - P'_a$	1	
Total number of equations		2nc+4	

Table B.7: Liquid-liquid decanter.

Parameters	**Notation in the text**	**Number of parameters**	**Units**[1]
Total molar holdup	n	1	kmol
Pressure	p_p	1	bar
Input variables		**Number of input variables**	
Molar flow	$J'_e[n]$	1	kmol/s
Temperature	T_e	1	K
Composition	$x_{e,i}$	nc	[-]
Output variables		**Number of output variables**	
First molar flow	$J'_{a,1}[n]$	1	kmol/s
Second molar flow	$J'_{a,2}[n]$	1	kmol/s
First molar holdup	$n1$	1	kmol
Second molar holdup	$n2$	1	kmol
Overall composition	x_i	nc	[-]
First liquid composition	$x_{1,i}$	nc	[-]
Second liquid composition	$x_{2,i}$	nc	[-]
Temperature	T	1	K
Pressure	P	1	bar
Total number of output variables		3nc+6	
Equation		**Number of equations**	
Holdups of components	$nx_i = n_1x_{1,i} + n_2x_{2,i}$	nc	
Total holdup	$n = n_1 + n2$	1	
Component material balances	$n\frac{dx_i}{dt} = rhs\_cmb_i - x_i rhs\_tmb$	nc-1	
Total material balance	$0 = rhs\_tmb$	1	
Constant pressure	$P = p\_p$	1	
Constant temperature	$T = T_e$	1	
Liquid-liquid equilibrium	$\gamma_{1,i}x_{1,i} = \gamma_{2,i}x_{2,i}$	nc	
Leverage for output flows	$J'_1[n]n_2 = J'_1[n]n_1$	1	
Summation condition (a)	$\sum_{i=1}^{nc} x_{1,i} = 1$	1	
Summation condition (b)	$\sum_{i=1}^{nc} x_{2,i} = 1$	1	
Total number of equations		3nc+6	

Table B.8: Liquid drum.

Parameters	**Notation in the text**	**Number of parameters**	**Units**
Total volume	V	1	m^3
Pressure	p_p	1	bar
Stream ratio	$stream\_ratio$	1	[-]
Input variables		**Number of input variables**	
Molar flow	$J_e[n']$	1	kmol/s
Temperature	T_e	1	K
Composition	$z_{e,i}$	nc	[-]
Output variables		**Number of output variables**	
First molar flow	$J_1[n']$	1	kmol/s
Second molar flow	$J_2[n']$	1	kmol/s
Outlet composition	z_i	nc	[-]
Pressure	P	1	bar
Temperature	T	1	K
Total number of output variables		nc+4	
Equations		**Number of equations**	
Component material balances	$\frac{v}{vm}\frac{z_i}{dt} = rhs\_cmb_i - z_i rhs\_tmb$	nc-1	
Total material balance	$0 = rhs\_tmb$	1	
Summation condition	$\sum_{i=1}^{nc} z_i = 1$	1	
Stream ratio definition	$stream\_ratio = \frac{J_1[n']}{J_1[n']+J_2[n']}$	1	
Constant temperature	$T = T_e$	1	
Pressure	$P = p\_p$	1	
Total number of equations		nc+4	

Table B.9: Proportional-Integral controller.

Parameters	**Notation in the text**	**Number of parameters**	**Units**[2]
Controller gain	K_c	1	
Integral time	τ_I	1	
Maximum manipulated variable value	max_man_var	1	
Minimum manipulated variable value	min_man_var	1	
Input variables		**Number of input variables**	
Process variable	PV	1	
Set point	SP	1	
Output variables		**Number of output variables**	
Controller output	CO	1	
Equation		**Number of equations**	
Manipulated variable	If $CO_{aux} > max\_man\_var$ $CO = max\_man\_var,$ If $CO_{aux} < mim\_man\_var$ $CO = min\_man\_var,$ else $CO = K_c[(SP-PV)+\frac{1}{\tau_I}\int(SP-PV)dt]$	1	
Observations			
Auxiliary variable	$CO_{aux} = K_c[(SP-PV)+\frac{1}{\tau_I}\int(SP-PV)dt]$		
2	Units of parameters or variables depend of the application context		

Appendix C

Implementation example: Vapor-liquid equilibrium calculations

Processes including vapor-liquid equilibria (VLE) are widely described and applied all along the chemical engineering literature. For this reason and for its importance in almost all chemical processes, this topic is taken as an illustrative example. Although this library comprehends an increasing number of model entities, we are going to concentrate our attention only on those modules regarding with VLE calculations, which include dew and bubble point calculations. The implementation of the required modules allows us to build a class structure and emphasize some interesting programming features of the Model Definition Language (MDL), the programming language of ProMoT. While only some modules are going to be taken from the current general library, the class structure used and described in this example is the same that can be found in the complete library.

In this appendix it will be shown that traditionally well described algorithmic tasks like VLE calculations can be solved or set up by creating the proper modules in MDL, which is not an algorithmic language. When using ProMoT for the first time it might be surprising that the language does not allow the implementation of functions as is done in traditional imperative languages like, for example, the sentence, $p = f(x)$ which assigns to the variable p the value of a function f with an argument x. Furthermore, it is not allowed to express assignations as it is done in traditional programming languages. We will see here, why it is not possible and the way MDL does it instead.

MDL discharges the modeler of setting algorithmic solutions instead of thinking about what is important for modeling; i.e., the set of equations; another tool (DIVA in this case) will take charge of the numerics, an intensive algorithmic process, while finding a solution for the system of equations.

Structuring entities in a suitable way is rather important. It can be said that the structure of a library is a ”suitable one” not only if it fulfills the requirements of an original modeler, but also because the entities of the library are extensively reused by subsequent modelers. Hence, the reuse frequency of an structured set of model entities is one its most valuable attributes.

C.1 The problem statement

We need VLE calculations for the following reasons: First, while designing a new process it is very important to perform phase equilibria calculations for sizing separation equipment like, for example, distillation columns, flash tanks and also equipment where chemical reactions are carried out. Second, while providing mathematical models with initial conditions, we need consistent data for the first iterations. It occurs that we need consistent VLE data as initial conditions but very often such data are not available in the literature. Finally, it is also useful to have VLE routines to test our models against experimental data, i.e., if we have experimental data we can estimate parameters, given a correlation or just test how well our model behaves under different P, T and composition conditions.

Let’s suppose that models described here were implemented according to the following problem description:

Given a mixture, calculate its dew and bubble point conditions, if the pressure or the temperature is fixed.

C.2 The model

According to the problem description, figure (C.1) shows a graphical representation for a particular case using the ProMoT graphical editor. This case consists of a methanol-water mixture for which phase equilibrium diagrams can be calculated at different conditions. Parts (a) and (b) of figure (C.1) seem to be rather similar; the icon under the text `p_mix` of part (a) represents the module that fixes the pressure and the overall methanol-water composition. In part (b), the module assigned to the text `t_mix` fixes also the composition, but instead of the pressure, the temperature is fixed in this case. `mw_vle` is the abbreviation of ”methanol-water vapor-liquid equilibria”, which is the name of the central module in cases (a) and (b) of figure (C.1), where the equations for solving the calculations are defined. Finally, `dew_point` and `bubble_point` are the names of the modules getting the corresponding calculated data. Details of the implementation are going to be considered, as soon as the modeling equations have been set in what follows.

When vapor and liquid phases are in thermodynamic equilibrium, the temperature, pres-

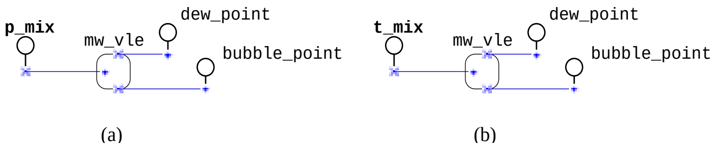


Figure C.1: VLE aggregated modules represented with icons using the graphic editor of ProMoT. (a) Is a model in which the pressure and composition of the mixture are given. (b) Is a model in which the temperature and composition of the mixture are given.

sure and chemical potential of each component are uniform throughout the system. Let’s start with the equality of the chemical potential of the vapor and the liquid phases expressed as a function of the temperature, T, the pressure, P and the composition of the vapor and liquid, y_i and x_i,

$$\phi_i y_i P = x_i \gamma_i f_i^o. \tag{C.1}$$

At given T and P, the fugacity coefficient, ϕ_i, and the activity coefficient, γ_i, are functions of the composition of the vapor and liquid respectively.

A very important application of equation (C.1) is found while working with systems under low to moderate pressure conditions, this means that the vapor phase can be considered to follow the ideal gas behavior. It implies that $\phi_i = 1$ and that the fugacity of the liquid is given by the saturation pressure of pure liquid i at the temperature of interest, P_i^o. Thus, our final working equation is

$$y_i P = x_i \gamma_i P_i^o. \tag{C.2}$$

For a two phase system in equilibrium, the Gibbs phase rule (Number of independent intensive properties = Number of components - Number of phases + 2) foresees a number of independent intensive properties equal to the number of components, nc. Only $nc-1$

compositions are independent, the last composition is calculated with one of the summation conditions, either $\sum_i^{nc} y_i = 1$ or $\sum_i^{nc} x_i = 1$. Thus, if we set the value of $nc-1$ compositions of a phase, it remains a degree of freedom, which can be either P or T, but not both.

Figure (C.2) illustrates, for the special case of a binary system, which calculations are

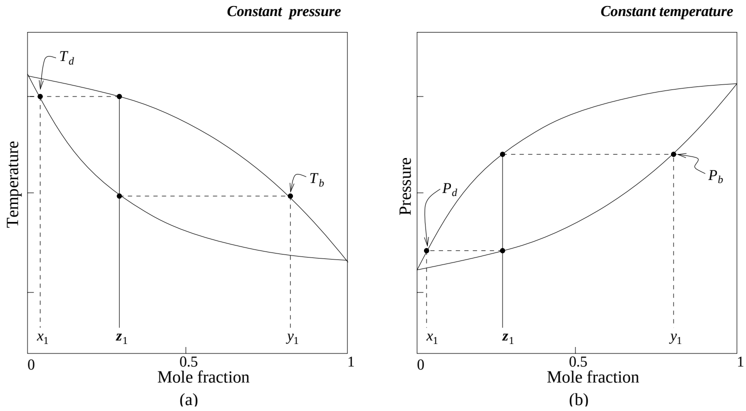


Figure C.2: Phase diagrams for a binary system. z_i stands for the composition of the mixture; x_i and y_i are calculated variables. It is important to note that z_1 and x_1 are compositions in equilibrium and also are y_1 and z_1.

possible with equation (C.2). Parts (a) and (b) of figures (C.1) and (C.2) are correspondingly related. Figure (C.2) shows a schematic representation of calculations performed by modules in figure (C.1). In both diagrams of figure (C.2) the composition of the mixture is denoted by z_i. If the pressure is set as parameter (see figure C.2a), the unknown variable will be the temperature. The dew point temperature corresponding to the given pressure is designated by T_d and the equilibrium composition is x_i. In this case z_i represents the composition of vapor in equilibrium with a liquid of composition x_i. T_b is the boiling point temperature of the liquid with composition z_i in equilibrium with a vapor of composition y_i.

Correspondingly, figure (C.2b) represents the case in which the temperature is given as a parameter and the dew point and bubble point pressures are calculated. Table C.1 presents the four types of calculations we will perform using equation (C.2). Given the possibility of four calculations, we shall determine, which of them are going to be performed and how these calculations will be translated in a ProMoT implementation.

Now that we have a physical model, we can proceed with the implementation using a programming language.

Table C.1: Vapor-liquid equilibrium calculations.

Calculation	Degrees of freedom	Unknown variables	Point in Fig. (C.2)
Dew point	P, y_i	T, x_i	T_d
	T, y_i	P, x_i	P_d
Bubble point	P, x_i	T, y_i	T_b
	T, x_i	P, y_i	P_b

C.3 Implementation

C.3.1 General features of the class architecture of modules involved in this application.

Our working model has been briefly set through equation (C.2). Now this model has to be translated into a programming language taking into account the problem description. For that the UML was used for representing the class architecture of the classes involved.

An abbreviated view of the class architecture for the VLE modules is shown in figure (C.3). The abstract class `vle-pt`[1] joints together the subclasses that according to the problem formulation will define the required calculations: one that represents the dew point and the other one for the bubble point calculations.

C.3.2 Dew point calculation.

In this case, the composition of the mixture, let's call it z_i, represents the composition of a vapor mixture and it will be initially known; also either the pressure or the temperature

[1]The name of this class is an abbreviation of: *vapor-liquid-equilibrium pressure-temperature*. With this name is indicated that the class is not limited to a fixed pressure or a fixed temperature.

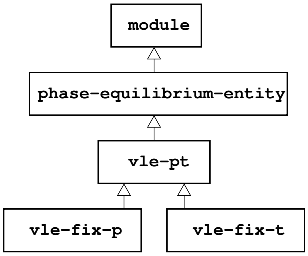


Figure C.3: Class diagram for the VLE model. First version

should be given. The following equation is the implemented version of equation (C.1) for dew point calculations,

$$z_i P_d = x_i \gamma_i(T_d) P_i^o(T_d). \tag{C.3}$$

Here, y_i has been substituted by z_i indicating that this value represents the composition of the initial mixture. P_d stands for the total pressure, but in this particular case P_d represents the dew point pressure of the mixture. In equation (C.3) it is also indicated that the temperature dependent functions are evaluated at T_d, the dew point temperature, but up to now it has not been set which variable, the pressure or the temperature, is preset.

C.3.3 Bubble point calculation.

Again, the known composition is denoted by z_i but now represents the composition of the liquid, the composition of the vapor y_i, is now to be calculated by the equation,

$$y_i P_b = z_i \gamma_i(T_b) P_i^o(T_b). \tag{C.4}$$

Where P_b is the total pressure and represents the bubble point pressure. The activity coefficient and the saturation pressure depend now on T_b, the bubble point temperature.

C.3.4 The variables.

Variables comprised in the implemented equations of the model have to be defined. The `vle-pt` class requires only three different types of variables, namely, parameter, input and output variables. Table (C.2) lists the defined variables and their respective type. It deserves here to mention that *input* are, in general, independent variables and, therefore their values are given by the modeler, that is how the word ”input” should be understood: as user determined variables. It occurs very often that input variables are data describing properties of streams ”getting into” a module, but also such properties can be calculated, i.e, they can be dependent variables. [2] In MDL, dependent variables are known as *output* variables. As expected in a complete model, there should be one and only one equation for each state variable. From table (C.2) we count $2nc+4$ output variables , but only $2nc+2$ equations have been defined—*nc* from equation (C.3), *nc* from equation (C.4), and two summation conditions—, thus the class `vle-pt` is an underspecified model, we need two more equations, which are going to be defined in the following two classes.

Table C.2: Variables defined in the class `vle-pt`

Notation	Type of variable	Implemented name	Number of variables	Meaning
y_i	output	`y[i]`	*nc*	Fraction mole of the vapor
x_i	output	`x[i]`	*nc*	Fraction mole of the liquid
P_d	output	`pd`	1	Dew point pressure
P_b	output	`pb`	1	Bubble point pressure
T_d	output	`ted`	1	Dew point temperature
T_b	output	`teb`	1	Bubble point temperature
z_i	input	`z[i]`	*nc*	Fraction mole of the mixture
- -	input	`pt`	1	Parameter for the pressure or the temperature
nc	structure parameter	`nc`	1	Number of chemical components

C.3.5 Using inheritance for fixing the last degrees of freedom.

The pressure or the temperature has to be fixed for the remaining degrees of freedom. Figure (C.4) shows schematically, the implementation of the equations, which complete

[2]For example, the heat charge of a heater flows *into* this device, but can be a dependent variable, which has to be calculated and should defined as output variable.

the definition of our model. Classes vle-fix-p and vle-fix-t[3] provide the equations required. These two classes illustrate the very interesting feature of inheritance in MDL, both of them share all attributes defined in the superclass vle-pt but each of them adds two equations depending on which variable is preset, either the pressure or the temperature, i.e., the parameterization is performed through equations that can be included in the model just selecting one of the two branches of the class structure. Though we started

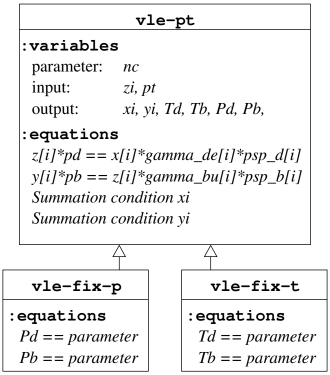


Figure C.4: Class diagram for the VLE showing more details of the implementation.

mentioning a particular mixture (methanol-water), the classes implemented so far are general classes and are completely reusable, they do not specify mixture dependent values. Before considering particular cases some more has to be said about, how these classes get data from outside?, how data are send out?, how temperature dependent functions are evaluated?, and how these functions get their mixture dependent parameters? Let's continue with the interphases with the outside world.

C.3.6 Terminals as interphases of modules.

The use of terminals is one of the most appealing features of ProMoT, since they provide the user with flexibility and security while connecting modules. Flexibility is achieved while using the visual editor, which allows drag and drop manipulation of modules and

[3]vle-fix-p is the abbreviation of *vapor-liquid equilibrium with fixed pressure* and vle-fix-t corresponds to the case of *fixed temperature*

their interconnection between terminals. The syntax analyzer of ProMoT ensures that only compatible connections be established between components.

Let's continue implementing the terminal for the information from the outside world required for class `vle-pt`, namely, the composition of a mixture and the fixed parameter (either pressure or temperature). Additionally, the number of components *nc* has not been fixed yet and has to be provided also from the outside. What makes the use of terminals a very interesting feature of ProMoT is that, once defined, they can be linked with just one coupling operation. For doing so, it is necessary to define compatible interphases in both sides of the connected terminals. Our class requires a terminal, which we are going to name, `mx`, with the following input data:

- the number of components, *nc*,
- the value of the fixed parameter, either T or P, and
- the composition of the mixture, z_i.

The identifier of the terminal, `mx`, aims at describing the properties of the mixture. While defining a terminal it is very important to choose standard names for the data types inside it, since the same names have to be found in classes whose terminals are to be connected. Names of variables defining terminals have an internal linking with names of corresponding variables in the class. Therefore, variables can have internally different names as those defined in the terminal. For example, z_i and x_i are both different names for compositions, these data have to"flow" through terminals, notwithstanding they have different names. Let's propose the following names for the variables of the terminal in class `vle-pt`:

- *nc*
- *dt*
- *composition*.

It is said that each variable inside a terminal is a *slot* and terminals require to have compatible slots to be connected. *nc* is the name of the slot where the integer data representing the number of components will be "transmitted" from or to the class; *dt* stands for data, which can be either a temperature or a pressure value; and the last slot, called *composition*, shall convey values of mole fractions, either z_i, y_i, x_i or whatever.

Using the same names as general identifiers inside terminals, ensures us to establish connections between terminals containing similar data types. The situation is illustrated in figure (C.5). In order to establish a relation between the name of a variable in a class and a slot in the terminal, it is used the reserved MDL word `parent`, as illustrated in figure (C.5). Now, terminals considering the output information have to be defined.

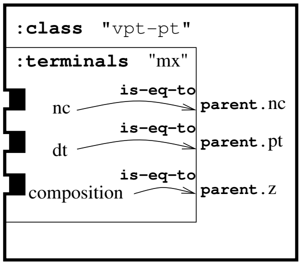


Figure C.5: Representation of the definition of a terminal. The three black filled squares represent slots, which should get information of the same type from another terminal. The arrows represent the internal connection between the names of the slots in the terminal and the names of the variables inside the class.

Variables defined as `output` are calculated by class `vle-pt`. But, as set by our class structure, the pressure or the temperature, can be fixed by parameterized equations defined in subclasses `vle-fix-p` and `vle-fix-t`. Then, it was suitable to define the terminals carrying the corresponding calculated data, in the class branch where the variables were really calculated.

The class `vle-fix-p` fixes the pressure, thus dew and bubble point temperatures, as well as their respective equilibrium compositions x_i and y_i, are to be calculated. Two terminals for the calculated data are required: The first one when the mixture is considered to be vaporized, whose name will be `vp` and the second when the mixture is considered to be condensed, called in this case `lq`. Before referring to the graphical illustration of this class, recall that `vle-fix-p` is a subclass of `vle-pt`, hence, the terminal `mx` will be inherited in this class-subclass entity. Figure (C.6a) shows graphically the class `vle-fix-p` and its terminals. Part (b) of this figure illustrates how are implemented the terminals in the class `vle-fix-t`, in which the temperature is fixed as parameter.

It is necessary to define some more classes and their respective terminals that can be cou-

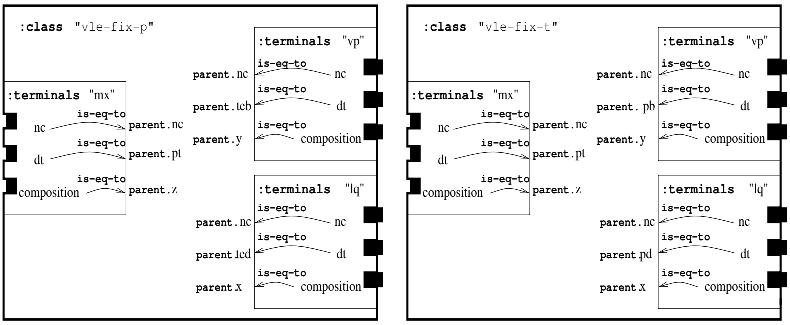


Figure C.6: Classes and their terminals. Both classes inherit terminal `mx`. Part (a) represents the class where dew and bubble point *temperatures* are calculated. Part (b) represents the case where dew and bubble point *pressures* are calculated. The arrows represent internal connections between slots in the terminals and names in the class. The terminals whose data are sent out of the class are represented with black pins.

pled with all the above described classes, in order to provide our classes with input data and a mean to get the calculated information. But now, we interrupt for a while what has to do with terminals, since nothing new will be added. Very near to the end, these classes working as sources or sinks of information will be considered.

In what follows, the definition of constitutive relations for our problem shall be tackled, namely, the activity coefficient and the saturation pressure.

C.3.7 Physical property correlations.

Even though our problem requires only two temperature dependent functions, in a library for chemical engineering processes, many other constitutive relations should be available. However, only the required relations will be described in detail, since the class structure for this few correlations refers also to other correlations available in the library.

These correlations are structured, in such a way, that the modeler should find a particular class just following the structure with no more resources than the background provided by an education in any engineering field. If we succeed providing a class structure, which can be followed without many other resources, we claim that our proposal was ”suitable”. The class structure will be briefly described, then we will continue using these classes, which is a more interesting issue.

The calculation of activity coefficients is taken as a start point. Some correlations for the activity coefficient are grouped in an abstract class. Suppose that our library contains implemented correlations for evaluating activity coefficients using three methods called here UNIFAC, UNIQUAC and Wilson (Smith et al., 1996). The class structure, where these classes are located, is shown in figure (C.7). Boxes with the respective names, represent classes where the correspondent correlation is defined. Note that these classes are general, since, no substance dependent parameter has been defined. Now, let’s suppose that we have decided to evaluate the activity coefficients for our illustrative example—the mixture methanol-water—using the UNIQUAC method. For this special case, the required parameters are found in the class `mw-uniquac`[4], which is a subclass of `uniquac`, where the correlation itself is implemented. Classes containing very particular information are defined at a very low level of the class hierarchy, allowing us to take advantage of reusing classes defined upward in the structure.

Now, assume that the Antoine equation is implemented in the library for calculating the saturation pressure in the class named `antoine` and we have the suitable parameters for our methanol-water mixture, which are defined in the class `mw-antoine`.

The classes `mw-uniquac` and `mw-antoine` can belong to an independent class hierarchy of any other branch in the library, say for example an abstract class `mw-data` but for the sake of simplicity, the structure remains as shown in figure (C.7).

A class containing the required correlations is to be defined. The class shall comprise modules of data and correlations, let’s call this class `mw-vle-dc`[5]. This class will aggregate all the correlations we need for our particular problem description. It is relative unlikely that this class can be reused for different problem descriptions, since the class consists of a set of correlations and parameters for a very specific case, i.e., VLE calculations for the methanol-water system. However, the *structure* of this class can be reused for a different mixture, if VLE calculations with the same the problem description statement, were required. It deserves to be mentioned that not only source code and a given set of classes can be reused, but also the structure of classes, that is to say, that the form can be

[4]Here we establish a naming convention for classes: when found `mw` in the name of a class, it refers to *methanol-water*.

[5]Another convention for naming classes: `dc` refers to *data and correlations*.

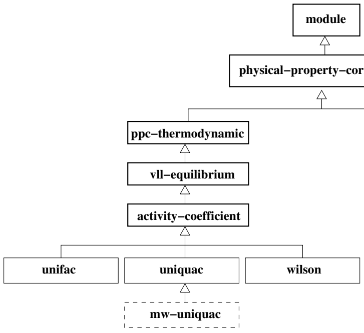


Figure C.7: An excerpt of the class hierarchy for physical property correlations. Thick frames represent abstract classes, thin frames represent classes where the correlations are defined, and dashed frames correspond to classes with parameters for an application specific case.

reused if the the same relationship between classes is maintained, even though the implemented classes are completely different. Further on, this type of reuse will be illustrated changing the mixture for which VLE calculations will be performed (see the section of applications).

According to equation (C.3) the evaluation of the activity coefficient and the saturation pressure as functions of the dew point temperature, T_d, are required. The MDL reserved word `:modules` is used to aggregate the required classes in `mw-vle-dc`. But the aggregation only ensures that some classes will be included into another one, in this case, it is also required an internal connection between variables of the different aggregated modules and the class that aggregates all these modules. In terms of our example, the classes `mw-uniquac` and `mw-antoine`, both aggregated in class `mw-vle-dc`, have their own variables and they have to be related with the variables defined in `mw-vle-dc`. It has to be recall that the correlations defined in these two aggregated classes are implemented with general names. How do we relate the names of our particular case with the names of the variables implemented in a general fashion, which define the correlations? That is also done through internal connections as was similarly done with the terminals.

For a complete aggregation of modules two main steps are required. First, the required modules are aggregated using the reserved word `:modules`. Each aggregated module will have its own name when included in a class. It allows us to include the same module more than once, giving a different name each time the module is aggregated; the reserved MDL word `:is-a` is used to match the given name with the name of the aggregated module. The way aggregation works is illustrated in figures (C.8) and (C.9). The class `mw-vle-dc` contains two exemplars of the class `mw-uniquac` for evaluating the activity coefficient, one at the dew point temperature, T_d, and the second one at T_b. Similarly, two modules are required for the saturation pressure at the respective temperatures.

Putting together the general and the particular scopes. So far two general classes were

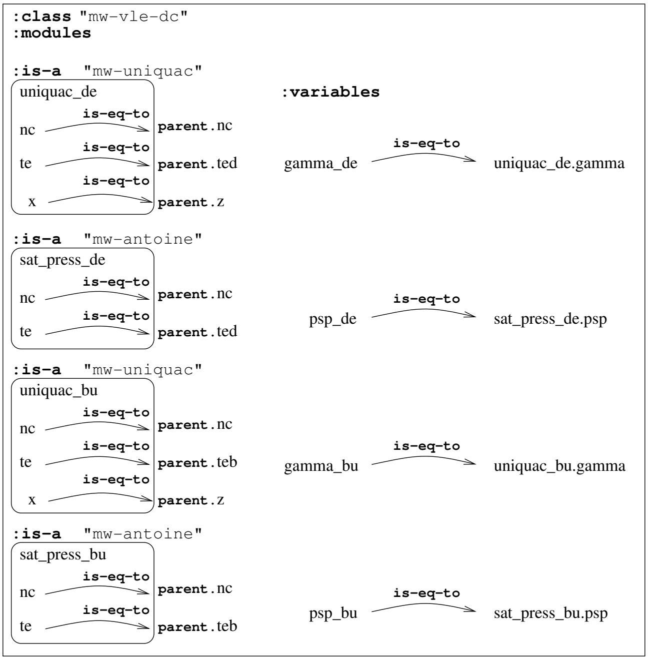


Figure C.8: The class `mw-vle-dc` aggregates four modules for calculating physical properties. Internal connections between variables of the modules and the variables in the class are illustrated. Also connections between the variables defined in the class and the variables calculated in the aggregated modules are sketched.

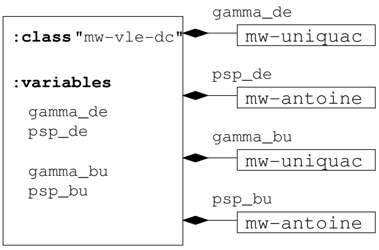


Figure C.9: UML representation of the implementation of the class shown in Fig. C.8. Black diamonds represent the aggregation relationship.

implemented: `vle-fix-p` and `vle-fix-t` according to the problem statement, these classes contain equations and variables covering the case at constant pressure and the case considering the temperature as constant. Additionally, the class `mw-vle-dc` comprises the required correlations for both cases, for the binary mixture methanol-water. In what follows, a class will be created inheriting both, the so called general class and the one containing data and correlations for a particular case. Two specialized classes result, the first one was named `mw-vle-p-fix` and the second one `mw-vle-t-fix`. Figure (C.10) shows the class structure for the constant pressure case. The class `vle-fix-p` share a new kind of relationship with its superclasses, this is called *multiple inheritance*. Hence, the resulting class comprises all attributes coming from both branches shown in figure (C.10). This class is finally what we were looking for: the class consists of a set of general features—equations and variables—and a set of mixture dependent attributes—data and correlations. Here it is to be pointed out that classes with a general character are completely reusable, while those including data and correlations contain expressions implemented also in general terms, hence reusable, as well as a set of parameters, whose implementation, although mixture dependent, is structurally similar to classes in which the same kind of parameters were defined. At the very beginning it was said how important is to type the less lines of code as possible. With the proposed class structure the modeler can dispose of a set of reusable classes and implement only those classes with particular data and correlations. If this implementation pattern is found each time a problem is required to be implemented, future modelers shall reuse classes and implement new ones in a systematic way.

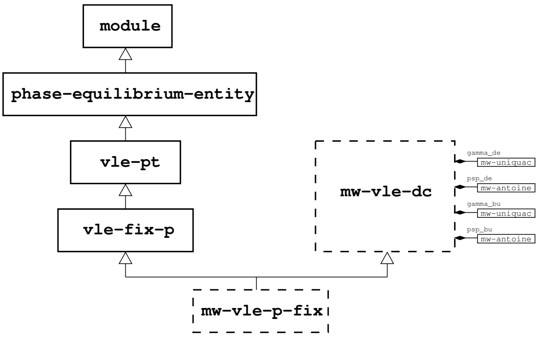


Figure C.10: Multiple inheritance allows to inherit attributes from classes in different branches of the hierarchy. Boxes with solid lines represent general classes, which are completely reusable. Boxes with dashed lines represent classes with mixture dependent attributes.

C.3.8 The closed model.

The properties of the mixture upon which VLE are to be calculated, namely, the composition and either the pressure or the temperature, will be supplied through a module connected to the terminal called `mx`. This input module should consist of variables defined as parameters, output variables, and equations between parameters and equations. Additionally, two output modules are required for getting the calculated data for dew and bubble points. The class containing the closed model comprises the required modules through aggregating them and grouping the connections of their terminals using the MDL reserved word `:links`. Connections between terminals can be performed using the graphical editor of ProMoT. In our example, terminals consist of three variables *nc*, *dt*, and the array called *composition*. When linked, with just one operation, all variables defining a terminal are correspondingly connected. Moreover, it is ensured that only connections between input-output variables be established. We arrive here at the point where all the efforts dedicated for implementing terminals is rewarded. Although not used in this example, the direction of the information through linked terminals can be bidirectional, feature which adds a very flexible frame, specially when terminals transmit signals, instead of information about process streams. This signals can be for example, couplings between controllers and process devices, for which the information flows in both directions.

Figure (C.1) shows the aggregated modules of two classes comprehending the twofold case considered here. Links between terminals are represented with lines. This is called a closed model, because all terminals of the aggregated modules are connected.

In what follows some calculations are going to be performed for different mixtures for illustrating the use of the classes developed above.

C.4 Using the same modules for different mixtures

Both phase diagrams shown in figure (C.11) were generated using the modules described in the last sections for the mixture methanol-water.

The diagrams of figure (C.11) were generated plotting the calculated bubble temperature

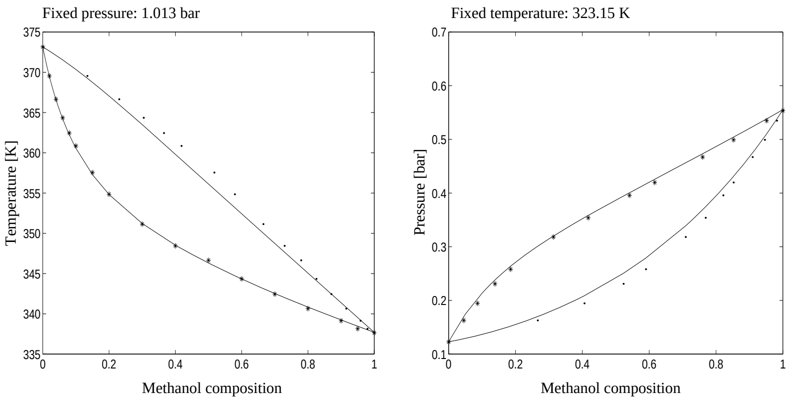


Figure C.11: Phase equilibrium diagrams calculated with ProMoT/DIVA models for the methanol-water mixture. The stars represent experimental bubble point data, while the dew point experimental data are represented with small points. Solid lines joint calculated values.

(left diagram) and dew point temperature (right diagram) at different compositions. The experimental data were taken from Seader and Henley (1998) p.168. The agreement of experimental data with calculated values is not discussed here, since the accuracy of the implemented model will depend on the parameters used in each case.

Reusing modules.

Let's consider now to use the same correlations, but a different mixture with its respective parameters, just to illustrate how to take advantage of the *structural reuse* of the classes where parameters are defined, changing only the required data for a new mixture. With structural reuse it is meant that a class its self will not be reused, but a modified copy of it will be created. This will be illustrated with the classes where the parameters of the new mixture are defined, since the same kind of parameters is going to be used, i.e. parameters for the Antoine equation and for the UNIQUAC method. Let's consider the system water-acetic acid and do it step by step.

- Classes for parameters and data. The best way to start is making copies of the classes `mw-antoine` and `mw-uniquac` which contain the required respective parameters for the methanol-water mixture. The copies are called `w-ac-antoine` and `w-ac-uniquac`. With these two classes, all that has to be done is to replace the old data with the new ones, the class structure of the former classes is also shared by new created classes.

- The class which aggregates data and correlations. Again, we proceed creating a copy of the class `mw-vle-dc`, let's call it `w-ac-vle-dc`. Originally, this class aggregates those classes for the last mixture. For the new system, the `:is-a` statements have to be changed replacing the names of the classes created in the step above.

- The class which puts together equations and aggregated correlations. Recall that our problem statement comprehends the case which considers the pressure as parameter and the case with the temperature as parameter. The classes for each case are `vle-fix-p` and `vle-fix-t`, respectively. This two classes are not mixture depend and can be reused through multiple inheritance. Specifically, for the case of constant pressure, we create the class `w-ac-vle-p-fix` which inherits all the attributes from `vle-fix-p` and `w-ac-vle-dc`. For the case of constant temperature the class `w-ac-vle-t-fix` is created, which has `vle-fix-t` and `w-ac-vle-dc` as superclasses.

- The closed model. Finally, a class aggregating and coupling the last modules has to be created for each case as illustrated in figure (C.1) for the methanol-water mixture.

After generating an output to DIVA, the work in ProMoT has finished and calculations can be performed in DIVA for different compositions and P, T conditions. Figure (C.12)

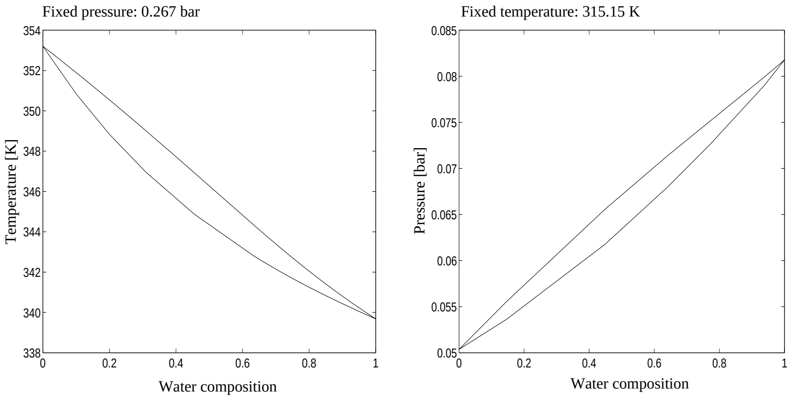


Figure C.12: Phase equilibrium diagrams calculated with ProMoT/DIVA models for the water-acetic acid mixture.

shows the calculated phase equilibrium diagrams for the binary mixture water-acetic acid. Since our purpose is centered in implementation issues, no further comparison with experimental data was performed.

Reusing and creating classes.

As a final application of this sequence of classes, let's consider the case of working with a different mixture and a different method for calculating the activity coefficients using the Wilson equation, which correlation is implemented in the library of physical properties and correlations. Let's consider to calculate VLE phase diagrams for the binary mixture n-butanol-water. Although this mixture exhibits liquid immiscibility at different compositions, only the equilibrium of vapor and liquid will be considered.

The procedure is very similar to the previous one and is listed in the following sequence of steps:

- A new class for parameters. Regarding the parameters for the Wilson equation for this mixture, we do not have a previous class which structure can be reused replacing the old data for new ones like was done in the previous example. Now, a new class with the required parameters for the Wilson correlation has to be implemented. Let's call this class `nbuol-w-wilson`, which has to be a subclass inheriting the correlation defined in the class `wilson-dechema`. The Antoine parameters for the new system are implemented in a similar fashion as in the previous examples, i.e., creating a modified copy with the new set of data.This class is called `nbuol-w-antoine`.

- The class aggregating data and correlations will, again, be very similar to the previous class of this kind and can also be created with a modified copy, changing only the `:is-a` statements which aggregate the data and correlations. The modified copy is called `nbuol-w-vle-dc`.
- The class `nbuol-w-vle-p-fix` puts together the model equations and the required correlations for the case of constant pressure and the class `nbuol-w-vle-t-fix` for the case of constant temperature. The implementation of these classes proceeds as in the last examples.
- Finally, two classes for the closed models are required: the class `nbuol-w-vle-f-p` when the pressure is given and `nbuol-w-vle-f-t` when the temperature is given.

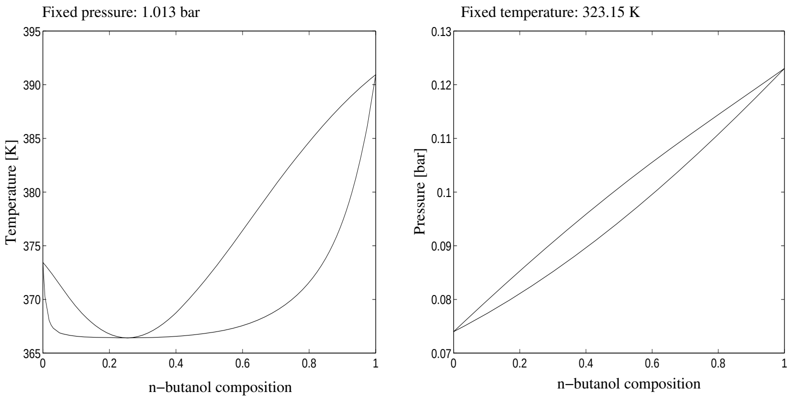


Figure C.13: Phase equilibrium diagrams calculated with ProMoT/DIVA models for the n-butanol-water mixture.

Figure (C.13) shows the calculated diagrams for this mixture.

C.5 Summary

In this appendix very attractive and functional features of the modeling tool ProMoT were described. It was shown that getting the best results of using ProMoT not only assumes managing its Model Definition Language, but also developing some habits while implementing the different modules, which include thinking on the reusability of them and the way the class structure allows this objective. The class structure proposed here was described through model entities that perform calculations of vapor liquid equilibrium for multicomponent mixtures. The same manipulations could be exemplified even though we were dealing with a different problem statement. Vapor liquid calculations were chosen for illustrative purposes, since it is a well understood topic and rather useful in many chemical engineering applications.

Appendix D

Remarks about implementation issues of the library

D.1 Naming conventions in the implementation notation

The notation for variables written in this dissertation is shown in the notation table at the beginning of this work. This notation is called *notation in the text* just to differentiate it from the *notation in the implementation* that is used for the implementation of the library, but is not used extensively in this text.

Giving names to parameters and values is fundamental when complex and highly structured models are developed, extensively used, and debugged. Following a systematic strategy for generating names, facilitates the development and manipulation of models. The same name can be used for designating variables if they are defined in different modules. For example, the modeler can name the temperature of the reactor with the same name used for the temperature of the next process unit. Names have a local scope. Creating names for variables is very important if we consider that a model will be manipulated by different users and variables are retrieved over and over during the life-cycle of the modeling-simulation process. Consider for example a model consisting of some thousands of variables which have already numerical values that represent the initial state of the system. Changing names of some variables is error prone and much difficulties might be encountered if names and their respective values do not match anymore.

Systematic naming brings many advantages: much documentation work can be saved;

the readability of modules is increased facilitating their manipulation; and the modeler is released of having to create names very often.

Variables

The following guideline was applied for generating names of variables in most of the cases, but when an ambiguity was generated, some other criterion might have been used:[1]

- Names of variables do not indicate if they are dependent or independent variables.
- The use of capital letters is avoided in general. When groups of letters have to be separated from others, an under score is used, for example, `delta_p`.
- Isolated letters are avoided when possible. The tendency exists, however, to name some variables of frequent use with just one letter. For example, when compositions designate dependent variables, `x` or `y` is used. Also `n` for designating the total number of moles is used.
- The *input/output character* of a variable refers to the fact that the property that this variable designates, belongs to a stream getting *into* the system or getting *out* of it. Letters `e` and `a` are used to define the input and output character of a variable, respectively.[2]
- Vapors, liquids and solids are characterized with `v`, `l`, or `s`, respectively.
- Variables that designate flows are designated with `j`.
- When the a stream has to be characterized as a mass stream, a heat stream, or flow of mechanical work letters `n`, `q` or `w` are used respectively. For example, a mass flow is designated as `jn` and a flow of heat as `jq`.
- The name of a variable is composed, in general, of the following parts:

[1]`This type character` was used to indicate code implemented in a computer language, in this work, usually MDL. Names of variables that are not coded in a programming language are written using typographical facilities, as shown in the notation table at the beginning of this work.

[2]Letters 'i' and 'o', would be more suitable for expressing the input/output character of a variable if these letters were not used very frequently for other purposes. Instead we used `e` and `a` which come the German: *Eingang* (input) and *Ausgang* (output).

1. A prefix that designates the variable. Table D.1 shows prefixes used for designating variables. The particular case of the temperature deserves a short comment. For the temperature, `te` is used since `t` is reserved for the independent variable time.
2. The input/output character of the variable, i.e. `e` or `a`.
3. Additional characteristics. For example, vapor (`v`), liquid (`l`), solid (`s`), or feed stream (`f`).

Table D.1: Prefixes used to name variables.

Variable	Prefix
Fraction mole of liquid	`x`
Fraction mole of vapor	`y`
Overall fraction mole	`z`
Molar enthalpy	`h`
Molar holdup	`n`
Parameter	`p`
Pressure	`p`
Flow	`j`
Temperature	`te`
Volume	`v`

With the above mentioned considerations, names of variables are systematically generated by the user. Table D.2 lists examples of names of variables generated systematically. It is important to point out that ProMoT cares of ensuring that no naming conflicts emerge, allowing that each name will be unique in the whole equation system. This fact releases the modeler of having to generate new names when a new model is created.

Equations

Naming equations is important in ProMoT. An equation can be redefined from subclasses, by just referring to the name and implementing the new version of the equation. Table D.3 shows names used frequently. Names of equations have also a local scope and the same names can be used in different modules. Again, this simplifies the notation and readability of models.

Table D.2: Examples of names of variables generated according to the conventions defined.

Description	Prefix	Input/output character	Additional characteristics	Name in the implementation	Name in the text
Temperature of the input vapor	te	e	v	`teev`	T''_e
Pressure of the output liquid	p	a	l	`pal`	P'_a
Parameter for the temperature	p		te	`pte`	pte
Input molar flow of liquid	j	e	nl	`jenl`	$J'_e[n]$
Molar fraction of the input vapor	y	e		`ye`	y_e
Molar enthalpy of the input vapor	h	e	v	`hev`	$h''_{e,v}$
Molar holdup of liquid	n		l	`nl`	n'

Table D.3: Examples of names of equations

Equation description	Name
Total material balance	`bal_tm`
Component material balance	`bal_cm`
Energy balance	`bal_e`
Vapor-liquid equilibrium	`eq_vl`
Summation condition for the liquid	`sum_x`
Summation condition for the vapor	`sum_y`
Pressure loss	`p_loss`

Terminals

Table D.4: Examples of names of terminals

Terminal description	Name
Input of liquid	`inl`
Input of vapor	`inv`
Output of liquid	`outl`
Output of vapor	`outv`
Input of heat	`inq`
Data output	`dt_out`

Terminals are connections between modules and their function consists of coupling variables between modules. For example, the temperature of the outlet stream of a module might be the temperature of the inlet stream of another one. Since compatibility between terminals has to be ensured, naming terminals is also important. Table D.4 lists names of terminals used in most of the cases. Terminal names have also a local scope and can be used again in different modules.

D.2 File structure of the library

Table D.5: MDL files that compose the library

File description	File name
Terminals	terminals.mdl
Physical property correlations	ppc.mdl
Elementary process entities	epe.mdl
Coupled process entities	cpe.mdl
Signal transformers	signal_tr.mdl
Phase equilibrium entities	pee.mdl

Table D.5 lists the files that create the whole structure of the library. File KDB.mdl loads all files and new applications can be started from this basic structure.

The applications described in this work can be loaded independently from each other. All applications require that the basic file structure of files listed in table D.5 be loaded.

Table D.6 lists the main file of the applications discussed in this contribution.

Table D.6: MDL files that compose the library

Application	Main file name
Plant for the production of butyl acetate	BUAC.mdl
Plant for the production of acetic acid	AZOT.mdl
Vapor-liquid equilibrium calculations	vle_app.mdl

Bibliography

Aris, R. (1993). Ends and beginnings in the mathematical modelling of chemical engineering systems. *Chem. Eng. Sci.*, *48*, 2507–2517.

Aris, R. (1999). *Mathematical Modeling. A Chemical Engineering Perspective*. Process Systems Engineering. Academic Press, USA.

Bahrami, A. (1999). *Object oriented systems development*. Irwin McGraw-Hill, USA.

Basmadjian, D. (1999). *The Art of Modeling in Science and Enginnering*. Chapman and Hall/CRC, USA.

Batres, R., Aoyama, A., and Naka, Y. (2002). A life-cycle approach for model reuse and exchange. *Comp. Chem. Engng.*, *26*, 487–498.

Baumgarten, B. (1992). *Petri-Netze: Grundlagen und Anwendungen* (2 edition). Spektrum-Hochschultaschenbuch, Germany.

Bayer, B., Schneider, R., and Marquardt, W. (2000). Integration of data models for process design–first steps and experiences. *Comp. Chem. Engng.*, *24*, 599–605.

Bogusch, R. (2001). *Eine Software-umgebung für die rechnergestützte Modellierung verfahrenstechnischer Prozesse*. Ph.D. thesis, RTWH Aachen.

Bogusch, R., Lohmann, B., and Marquardt, W. (2002). Computer-aided process modeling with ModKit. *Comp. Chem. Engng.*, *25*, 963–995.

Bogusch, R. and Marquardt, W. (1997). A formal representation of process model equations. *Comp. Chem. Engng.*, *21*, 1105–1115.

Cha, P. D., Rosenberg, J. J., and Dym, C. L. (2000). *Fundamentals of Modeling and Analyzing Engineering Systems*. Cambridge University Press, USA.

Dattatri, K. (1997). *C++: Effective Object-Oriented Software Construction*. Prentice Hall.

Denn, M. (1986). *Process Modeling*. Longman, England.

Douglas, J. (1988). *Conceptual design of chemical processes*. McGraw-Hill, USA.

Eggersmann, M., von Wedel, L., and Marquardt, W. (2002). Verwaltung und Wiederverwendung von Modellen im industriellen Entwicklungsprozess. *Chemie-Ing.-Techn.*, *74*, 1068–1078.

Ezran, M., Morisio, M., and Tully, C. (2002). *Practical software reuse*. Springer-Verlag, England.

Foreman, J. (1996) Software Technology Conference, Salt Lake City, USA. More information is found in the URL http://www.sei.cmu.edu/str/descriptions/deda.html.

Gamma, E., Helm, R., and Johnson, R. (2002). *Design patterns : elements of reusable object-oriented software*. Addison-Wesley, USA.

Gilles, E. (1997). Netzwerktheorie verfahrenstechnischer Prozesse. *Chemie-Ing.-Techn.*, *69*, 1053–1065.

Hangos, K. M. and Cameron, I. T. (2001). A formal representation of assumptions in process modelling. *Comp. Chem. Engng.*, *25*, 237–255.

Hartig, H. and Regner, H. (1971). Verfahrenstechnische Auslegung einer Veresterungskolone. *Chemie-Ing.-Techn.*

Holl, P., Marquardt, W., and Gilles, E. D. (1988). DIVA–a Powerful Tool for Dynamic Process Simulation. *Comp. Chem. Engng.*, *12*, 421–426.

Kahlbrandt, B. (2001). *Software-Engineering mit der Unified Modeling Language* (2nd edition). Springer, Germany.

Köhler, R., Räumschüssel, S., and Zeitz, M. (1997). Code Generator for Implementing Differential Algebraic Models Used in the Process Simulation Tool DIVA. In *15th IMACS World Congress*, Vol. 24-29 Germany.

Kurzok, A., Pahl, M. H., and Schulz, A. (2001). Software zur wissensbasierten Prozessmodellierung -WIP. *Chemie-Ing.-Techn.*, *73*, 1184–1188.

Lefkopoulos, A. and Stadtherr, M. (1993). Index analysis of unsteady-state chemical process systems–I. An algorithm for problem formulation. *Comp. Chem. Engng.*, *17*(4), 399–413.

Li, X. and Kraslawski, A. (2003). Conceptual process synthesis: past and current trends. *Chem. Eng. Proc.*, *43*, 589–600.

Mäkilä, P. and Waller, K. (1981). The energy balance in modeling gas-phase chemical reactor dynamics. *Chem. Eng. Sci.*, *36*, 643–652.

Mangold, M., Angeles-Palacios, O., Ginkel, M., Kremling, A., Waschler, R., Kienle, A., and Gilles, E. (2004). Computer-Aided Modeling of Chemical and Biological Systems, Methods, Tools, and Applications. *Ind. Eng. Chem. Res.* Accepted.

Mangold, M., Kienle, A., Gilles, E., and Mohl, K. (2000). Nonlinear computation in DIVA – methods and applications. *Chem. Eng. Sci.*, *55*, 441–454.

Mangold, M., Motz, S., and Gilles, E. (2002). A network theory for the structured modelling of chemical engineering processes. *Chem. Eng. Sci.*, *57*, 4099–4116.

Marquardt, W. (1996). Trends in computer-aided process modeling. *Comp. Chem. Engng.*, *20*, 591–609.

Marquardt, W., von Wedel, L., and Bayer, B. (2000). Perspectives on lifecycle process modeling. In *Foundations of Computer-aided process design*, Vol. 96, pp. 192–214.

McCabe, W. L., Smith, J., and Harriot, P. (1985). *Unit operations of chemical engineering*. McGraw Hill, Singapore.

Osigus, H. (1998) Modellierung einer Reaktivdestillationskolonne für den Butylacetatprozess mit dem Modellierungswerkzeug ProMoT. Dissertation, Universität Stuttgart.

Otter, M. (1999). Objektorientierte Modellierung physikalischer Systeme, Teil 1. *Automatisierungstechnik*, *47*.

Pantelides, C., Gritsis, D., Morison, K., and Sargent, R. (1988). The mathematical modelling of transient systems using differential-algebraic equations. *Comp. Chem. Engng.*, *12*(5), 449–454.

Polke, M. (Ed.). (1994). *Process Control Engineering*. VCH, Germany.

Quantrille, T. E. and Liu, Y. A. (1991). *Artificial intelligence in chemical engineering*. Academic Press, USA.

Quibeldey-Cirkel, K. (1999). *Entwurfsmuster*. Springer, Germany.

Sargent, R. (2004). Introduction: 25 years of progress in process systems engineering. *Comp. Chem. Engng.*, *28*, 437–439.

Schneider, R. and Marquardt, W. (2002). Information technology support in the chemical process design life cycle. *Chem. Eng. Sci.*, *57*, 1763–1792.

Seader, J. and Henley, J. (1998). *Separation Process Principles*. Wiley, USA.

Shah, P. B. and Kokossis, A. C. (2001). Knowledge based models for the analysis of complex separation processes. *Comp. Chem. Engng.*, *25*, 867–878.

Simon, H. A. (1996). *The sciences of the artificial*. MIT Press, England.

Smith, J., Ness, H. V., and Abbot, M. (1996). *Introduction to Chemical Engineering Thermodynamics*. Mc Graw-Hill, Singapore.

Stephan, K. and Mayinger, F. (1998). *Thermodynamik* (15 edition)., Vol. 1. Springer, Germany.

Stephanopoulos, G. and Han, C. (1996). Intelligent systems in process engineering: a review.. *20*, 743–791.

Stephanopoulos, G., Johnson, J., Kriticos, T., Lakshmanan, R., and Siletti, C. (1987). DESIGN-KIT: an object-oriented environment for process engineering. *Comp. Chem. Engng.*, *6*, 655–674.

Stephanopoulos, G. and Mavrovouniotis, M. (1988). Artificial intelligence in chemical engineering research and development. *Comp. Chem. Engng.*, *12*, v–vi.

Taylor, R., Krishna, R., and Kooijman, H. (2003). Real-world modeling of distillation. *Chem. Eng. Prog.*, 28–39.

Tränkle, F. (2000). *Rechnerunterstützte Modellierung verfahrenstechnischer Prozesse für die Simulationsumgebung DIVA*. Ph.D. thesis, University of Stuttgart, Germany.

Tränkle, F., Gerstlauer, A., Zeitz, M., and Gilles, E. (1997a). Application of the Modeling and Simulation Environment ProMoT/DIVA to the Modeling of Distillation Processes. *Comp. Chem. Engng.*, *21*, 5841–5846.

Tränkle, F., Gerstlauer, A., Zeitz, M., and Gilles, E. (1997b). PROMOT/DIVA: A prototype of a process modeling and simulation environment. *In Troch, I. and Breitenecker, F. (Eds.): IMACS Symposium on Mathematical Modeling, 2nd MATHMOD, TU Viena., 11*, 341–346.

Tränkle, F., Kienle, A., Mohl, K., Zeitz, M., and Gilles, E. (1999). Object-Oriented Modeling of Distillation Processes. ESCAPE-9 23:743-746, Budapest, Hungary, 1999.. *Supplement to Computers and Chemical Engineering, 23*, 743–746.

Tränkle, F., Zeitz, M., Ginkel, M., and Gilles, E. (2000). Promot: a Modeling Tool for Chemical Processes.. *Mathematical and Computer Modelling of Dynamical Systems, 6*, 283–307.

Waschler, R. (1996). *Entwicklung einer Modellfamilie für die Destillation/Rektifikation*. Diplomarbeit, Universität Stuttgart.

Waschler, R., Angeles-Palacios, O., Ginkel, M., and Kienle, A. (2003). Application of the Process Modeling Tool ProMoT to Large–scale Chemical Engineering Processes. In *Proceedings 4th MATHMOD, IMACS Symposium on Mathematical Modelling, February 5–7, 2003*, Vol. 2 of *ARGESIM Report no. 24, ISBN 3-901608-24-9*, pp. 1113–1121 Vienna, Austria. Vienna University of Technology.

Waschler, R., Angeles-Palacios, O., Ginkel, M., and Kienle, A. (2006). Object-oriented modeling of large-scale chemical engineering process with ProMoT. *Mathematical and Computer Modeling of Dynmical Systems, 12*(1), 5–18.

Waschler, R., Kienle, A., Anoprienko, A., and Osipova, T. (2002). Dynamic Plantwide Modelling, Flowsheet Simulation and Nonlinear Analysis of an Industrial Production Plant. In Grievink, J. and van J. Schijndel (Eds.), *European Symposium on Computer Aided Process Engineering–12–ESCAPE-12* Amstemdam. Elsevier.

Yang, A., Schlüter, M., Bayer, B., Krüger, J., Haberstroh, E., and Marquardt, W. (2003). A concise conceptual model for material data and its applications in process engineering. *Comp. Chem. Engng., 27*.

Zadeh, L. and Desoer, C. A. (1979). *Linear System Theory. The State Space Approach*. Rober E. Krieger Publishing Company, USA.

Zadeh, L. and Polak, E. (1969). *System Theory*. McGraw-Hill, New Delhi.